Krimtrack: An integrated coordination system with a virtual machine learning advisor

Author: David Gómez
Supervisor: Alfonso Rojas

A project submitted in fulfilment of the requirements for the Degree in Telecommunications Engineering in

Escola Tècnica Superior d’Enginyeria de Telecomunicacions de Barcelona (ETSETB)
Universitat Politècnica de Catalunya

October 2014
"The only real security that a man can have in this world is a reserve of knowledge, experience and ability."

Henry Ford
# Contents

<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>vi</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Introducing Krimtrack ........................................... 1
1.2 Security and the need for coordination ..................... 2
1.3 Big Data and the need for information ........................ 2
1.4 Interdisciplinarity and the need for cooperation .......... 2
1.5 Document structure ............................................... 3

## 2 Motivation and Objectives

2.1 Current situation .................................................. 4
2.2 Common complaints ................................................ 4
2.3 Enter Krimtrack ................................................... 5
2.4 Objectives .......................................................... 6

## 3 Krimtrack architecture

3.1 Overview ........................................................... 8
3.2 Mobile terminal .................................................... 9
3.2.1 Device comparison ............................................ 9
3.3 NFC Tags ........................................................... 10
3.4 Mobile terminal application ..................................... 13
3.4.1 Features ....................................................... 14
3.4.2 Navigation map ................................................ 15
3.4.3 Architecture .................................................. 15
3.4.4 Cache database schema ....................................... 18
3.5 Communication server ............................................. 19
3.5.1 Architecture .................................................. 19
3.5.2 Krimtrack protocol (KTP) .................................... 19
3.6 Database .......................................................... 21
3.7 Coordination web application .................................... 21
3.7.1 Features ....................................................... 21
3.7.2 Interface ....................................................... 23
3.8 Machine learning advisor ........................................ 23
3.8.1 Architecture .................................................. 24
4 Machine Learning

4.1 Machine learning 101
4.1.1 Concepts and terminology
4.1.2 Types of learning
4.1.3 Metrics to evaluate a model
4.1.4 Machine learning limitations
4.1.5 Learning procedure

4.2 Data sets used
4.2.1 Audiology
4.2.2 Column
4.2.3 Breast cancer
4.2.4 Fourier
4.2.5 Credit
4.2.6 Nursery
4.2.7 Imports

4.3 Filters
4.3.1 Attribute filters
4.3.1.1 Correlation-based feature selection (CFS)
4.3.1.2 CFS per class
4.3.2 Instance filters
4.3.2.1 Fast Condensed Nearest Neighbour rule (FCNN rule)
4.3.2.2 Entropy-based instance selection

4.4 Classifiers
4.4.1 Naive Bayes
4.4.2 C4.5 decision tree
4.4.2.1 C4.5 random decision tree
4.4.2.2 C4.5 decision stump
4.4.3 K-nearest neighbours (k-NN)
4.4.4 Artificial neural network
4.4.5 Logistic regression

4.5 Ensemble classifiers
4.5.1 Bootstrap aggregating (bagging)
4.5.1.1 C4.5 random forest
4.5.2 Boosting
4.5.2.1 AdaBoost
4.5.2.2 AdaBoost.M1
4.5.3 Stacking
4.5.4 Original attempts
4.5.4.1 Class stacking
4.5.4.2 Soft class stacking
4.5.4.3 Reinforced class ensemble 1
4.5.4.4 Reinforced class ensemble 2
4.5.4.5 Iterative class ensemble

4.6 Regressors
4.6.1 Linear regression
4.6.2 k-NN regression

4.7 Testing methodology
4.7.1 Error measurement
4.7.1.1 Accuracy / MAE
4.7.1.2 Baseline accuracy
4.7.1.3 Confusion matrix
4.7.1.4 Precision / recall
4.7.2 Cost matrix
4.7.3 Receiver operating characteristic (ROC)
4.7.4 Bias / variance
4.7.5 Model testing
4.7.5.1 On training validation
4.7.5.2 Holdout
4.7.5.3 Cross-validation
4.7.5.4 Bootstrap validation
4.8 Hyperparameter selection
4.8.1 Grid search

5 Results
5.1 Use case validation
5.2 Communication server
5.2.1 Server specifications
5.2.2 Load testing
5.3 Machine learning results
5.3.1 Data sets
5.3.1.1 Audiology
5.3.1.2 Column
5.3.1.3 Breast cancer
5.3.1.4 Fourier
5.3.1.5 Credit
5.3.1.6 Nursery
5.3.2 Classifier algorithms
5.3.2.1 Naive bayes
5.3.2.2 C4.5 decision tree
5.3.2.3 Neural network
5.3.2.4 k-NN
5.3.2.5 Logistic regression
5.3.2.6 Random C4.5 forest
5.3.2.7 AdaBoost.M1
5.3.2.8 Stacking
5.3.2.9 Class stacking
5.3.2.10 Soft class stacking
5.3.2.11 Iterative class ensemble
5.3.2.12 Reinforced class ensemble 1
5.3.2.13 Reinforced class ensemble 2
5.3.3 Classifier meta-analysis
5.3.4 Regression algorithms
5.3.5 Filters
5.3.5.1 Feature filters
Acknowledgements

I would like to express my gratitude to my fellow team members Guille, Judith and Alex to grant me the opportunity to work in an interesting project. Who could have thought terrorism would be an interesting topic to deal with?

I would like to thank my supervisor Alfonso Rojas for his direction on the project and suggestions on new approaches and variants to algorithms. Even though we didn’t reinvent machine learning, I sure learned a lot along the way.
Chapter 1

Introduction

This first chapter introduces the Krimtrack project, its goal as a product and service as well as its most interesting features from an engineering point of view. Also, we present an overview of the chapters in this document as a guide on what to expect from each one.

1.1 Introducing Krimtrack

Krimtrack is an integrated coordination system aimed primarily at security companies. As we will see in detail in chapters 2 and 3, it is an orchestration of several subsystems working as a unified service to provide coordinators with a responsive set of tools to help them with their jobs. These coordinators oversee a group of employees (like security guards) and react to incidents and emergencies by working and communicating with them to solve any problems that might arise.

The idea behind this project came to us as a proposal of a criminologist friend, who formed a group with two other criminologists, to design and offer a service to improve the quality of security companies’ services. One of our future colleagues was working as a security guard and on his job he was able to witness all the problems and inefficient procedures that exist within the private security world and were solvable by the appropriate use of technology. Thus, Krimtrack was born.

As the author of the project, and beyond the mere engineering aspect, this project has had other interesting properties that made it particularly appealing to work with.
1.2 Security and the need for coordination

As we just said, security companies (and virtually any medium or large entity) has problems in their operations that are inherent to their size. Most of these problems are organizational in nature, which means that the larger the company the more important coordination is among its employees.

At first glance, coordination might seem just a necessary evil for a company, a prerequisite to function correctly. But a lacking organization often means poor performance in the company’s operations as well, which has impact on its work environment too, as well as its bottom line. The chain of causality may be difficult to perceive, but its consequences are very real: robberies, assaults, unguarded zones or injured guards are usually a by-product of an oversight and they are many times avoidable. This is why, in the case of security companies, an improvement on coordination results in better security for everyone.

1.3 Big Data and the need for information

In the last years and decades we witnessed a vast number of technological changes of all kinds: faster processors, bigger memories, better software, more sophisticated algorithms, ubiquitous hardware, ... All of these led to an unprecedented widespread deployment of technology, with many capable data sources all over the world. This heterogeneous, unstructured and massive volume of data is what is known as Big Data.

With the proper treatment, one can distill important knowledge from this large pile of data. Capturing all the available information generated by an orchestrated system, we can take advantage of that huge amount of knowledge and use it to help in the decision making process. Such an approach is already used by many fields such as banking, finance, medicine or science, with excellent results.

1.4 Interdisciplinarity and the need for cooperation

Turning our attention to the project’s development itself, it is interesting to note a fact that sometimes engineers forget: most interesting projects require specialized knowledge in different areas. The end goal of engineering is to design technological systems to fulfill a particular need in a particular domain; to be effective an engineer has to be able to extract, understand and translate these needs to technical terms.
In this project we’ll be working with two criminologists, each with different specializations (security direction and predictive policing). Thus, we have come to find, interdisciplinarity is an essential quality for an engineer. Cooperation with other specialists builds the foundation for a good work environment, as well as being a great learning experience about fields other than one’s own.

**1.5 Document structure**

This section describes the structure that we will follow throughout this document to present the work done on Krimtrack.

In first place we have this introduction in chapter 1, as an initial overview on the philosophy of Krimtrack.

Chapter 2 explains in more detail what is Krimtrack, what it does and the environment it tries to replace. After this functional explanation we delve deeper into the internal structure and subsystems of the project on chapter 3.

Since the project has a strong emphasis on data analysis, we dedicate the entirety of chapter 4 to explain the basics on machine learning, as well as the algorithms and methodology implemented on Krimtrack’s machine learning module.

Once all the project is presented, we discuss its empirical results on chapter 5. These results are mainly concerned about the machine learning inference, but we will see also some experimental tests on the rest of the software system.

One of the services we will offer as a company will be data analysis on the information generated by the system, so we thought it would be interesting to perform a small test on similar data to train ourselves in such an endeavour. For this rehearsal we have chosen the Global Terrorism Database (GTD), which includes terrorism incidents from all over the world since 1970. In chapter 6 we document this analysis, from acquiring the raw database to obtaining results.

At last, we will finish with some conclusions on chapter 7, self-evaluating the project and contrasting its initial objectives versus its current content. We will also use this chapter to sketch some guidelines for future work to expand Krimtrack and fulfill its vast amount of potential.
Chapter 2

Motivation and Objectives

2.1 Current situation

Security companies serve a crucial role in the effective operative of all kind of services. Many companies hire them to protect their facilities, their personnel or, in public services, even citizens using them. We started focusing on train security, where the importance of surveillance tasks is greater given the large affluence of people on a daily basis, and we were able to identify worrying procedures that might have ultimately compromised its security.

Guards are the field operatives, with strictly scheduled routes designed by a team leader. They follow these routes and intervene if there is any incident, on which case they add it on an incident report once it’s been dealt with. Guards also fill a different regular report with their actual arrival times on the scheduled areas.

Furthermore, a control center is working 24 hours a day to supervise any exceptional case that requires coordination with other external services such as the police, firemen or ambulances. To contact them guards must call them via phone. If they are not available, they can only rely on themselves or nearby partners.

2.2 Common complaints

To judge the previous situation, we collected several common complaints among both guards and coordinators:
• Guards are difficult to locate with precision on any given time. Moreover, many times they are not where they are scheduled to be and problems arise because of it.

• To check on the guards’ schedules, inspectors are sent periodically to all over the geography on which the company operates, which results in a significant expense in time and gasoline.

• The control center is sometimes unavailable due to a network overload, which can be potentially life-threatening for guards.

• Both regular and incident reports are often written on paper, which means that they take more time to write, send, store, read and analyze than digital documents. This current manual workflow is also expensive in both time and money.

• Those paper reports must be sent to the central headquarters, which is inconvenient and, again, expensive.

• Incidents are reported at the end of a guard’s schedule, which limits the reaction time of the company to attend incidents.

• Many preparation tasks are difficult to do: schedules, guard allocation and incident prevention all are complicated procedures that require a lot of data review to do optimally.

2.3 Enter Krimtrack

Krimtrack is an integrated coordination system aimed at security companies and other organizations with a large number of staff to manage in real time. There are three main parts: mobile terminals (rugged smartphones with an Android operating system), the communication server and the machine learning inference system.

The guards use the mobile terminals as a link to the control center (also called coordination center). Each terminal has several functionalities that replace most of the equipment the guards currently carry and, at the same time, it provides additional tools to optimize their jobs, such as geolocation and real-time bidirectional communication and feedback.

As the core of the system we have a communication server, interconnecting all the terminals. These connections are supervised by a coordinator via a dedicated web application, with access to real-time feedback on the guards’ position, routes and incidents, among others. The server also will have an administration interface where it will manage all its resources (e.g. guards, zones, schedules, incidents, ...).
The most interesting area of the system from an academic point of view, and the focus of this document, is the machine learning module. All the generated data in the system serves as input to this module that will advise the coordination staff about ways to improve their business operations, such as resource allocation or warnings on possible future incidents. Furthermore, we will use this module to perform data analysis on the data gathered by Krimtrack, which will be summarized in reports with figures, conclusions and possible courses of action for the future.

![Figure 2.1: Krimtrack as a black box.](image)

During this project’s development, we took part in an entrepreneurship contest on business ideas organized by the Innova program on the UPC. Among more than 90 projects, Krimtrack was awarded the prize for best business project by the contest sponsors: EADA business school, Caixa d’Enginyers, Solvay, Sener and KIC InnoEnergy.

### 2.4 Objectives

As a summary of the specific objectives of this project we can include:

1. Design and implement an Android mobile application capable of geolocation, assisted report creation and coordination-based communication.

---

2. Design and implement a communication server.

3. Design and implement a coordination web application.

4. Research, analyze, implement and evaluate alternatives in the state of the art in machine learning.

5. Research new approaches to machine learning.

6. Design and implement a machine learning suite, eventually integrating it with the coordination system.

7. Use the machine learning algorithms to analyse a given data set and extract relevant conclusions.

As we have mentioned, this academic project is part of a larger business operation, but we deliberately omitted the mention of problems, methods and results from other non-technical areas. Thus, the treatment of economic, commercial or marketing issues, for example, are not included as objectives of this document.
Chapter 3

Krimtrack architecture

3.1 Overview

Krimtrack is a relatively complex system, containing five main modules plus a relational database, as seen in figure 3.1. One of these modules, the administrative web application, will not be included in the description of this document since we believe it has little academic interest. It is included in the diagram for completeness’s sake; its role, however, is limited to the management (i.e. addition, deletion, update and query) of the resources used by the rest of the system (e.g. users, zones, schedules, reports, incidents, ...).

Figure 3.1: Krimtrack subsystems.
Of all the remaining four, we have focused most of our attention on the machine learning module. It is the most innovative and versatile subsystem, and can be seen as the realization of the most ambitious goal of Krimtrack: to extract information to ultimately aid the user in his job. In section 3.8 we will talk about its architecture but we dedicate the whole chapter 4 to study the algorithms and methodology it uses.

Considering the system’s requirements, the modules have been designed using Java EE 7 technology: websockets and JSP in particular. A big advantage of this technology, in our case, is the integration of all the modules in one single .war package, easily deployable to any application server such as JBoss, Tomcat or GlassFish among others.

During this chapter we will review each of these subsystem separately.

### 3.2 Mobile terminal

Before we talk about the mobile application itself, we need to choose an appropriate device for the purposes it will be used for. To be able to choose from a variety as large as possible, we will choose a smartphone with Android OS as our platform. Also, given its final users (security guards or similar) and how it will be used, the mobile terminal will have to comply with a series of requisites:

- Durable against accidental impacts (rugged).
- Strong protection against particles (such as dust and water).
- Long battery life (12+ hours of intensive usage). Since we will be using a lot of the phone’s subsystems this requisite is specially important.
- NFC support (explained below).
- Screen big enough to be comfortable to use, yet not too big to be unwieldy.

Additionally, Push-To-Talk (PTT) support would provide additional value as well.

#### 3.2.1 Device comparison

Since these requirements are not found on off-the-shelf smartphones we need to look elsewhere. A commercial agent in China sent us specifications about three different smartphones, directly from the manufacturer. All of them have the requirements specified
above, including PTT, and the relevant features are compared in figure 3.2. In order to maintain confidentiality we will call them device #1, #2 and #3 respectively.

IP grade specifies how resistant the device is against particles and liquids, as shown in figure 3.3.

Device #3 is clearly better in every way: better screen, better processor, more memory and a significantly larger autonomy. Of course, an important drawback is its high price. The other two devices are more similar in price, and #2 is better enough to justify the increase in $15. Hence, we will recommend device #2 as the one with best quality-price ratio, offer #1 as the budget alternative and #3 as the premium device. Ultimately, though, the decision will depend on the client, and our recommendation might vary depending on the circumstances (for instance, a client with very long schedules might need the 3800 mAh battery to last the whole service).

### 3.3 NFC Tags

Near-field Communication (NFC) is a protocol derived from the RFID standard, but designed to work on a very short range (less than 10 centimetres). Such protocol is useful in instances where physical proximity is a desired requirement, such as access control, stock management, product identification or private data exchange.
NFC tags are passive circuits with a tiny amount of memory. This memory can be written and read using special NFC hardware present in some modern smartphones.

We will begin our search for the appropriate tags from the newest NTAG21x series from the manufacturer NXP:

As seen in figure 3.4 there are several features to evaluate:
In our case, memory size is irrelevant, since we will be using the tag universal identifier (UID) as a zone identifier; we don’t need to read or write anything in memory.

The 32-bit password looks like a promising feature; it permits the tag owner to lock the tag under a symmetrical cryptographic scheme, which enables access only if the reader has the correct password. In our system, that would mean inserting this password into all terminals, and if the password was ever compromised it would require to change all terminals. In any case, the information we need in the tag (the UID) is not sensitive information so we don’t need to hide it.

UID Ascii mirror allows the information in the tag to reference its UID; e.g. a link to http://example.com?uid=X, where X is the current tag UID). A feature we don’t need.

The 24-bit counter includes a memory on the tag which keeps count on the number of times a tag has been read. A useful feature for marketing perhaps, but not for us.

The originality signature is a more interesting feature. We use the UID to identify each tag, counting on the fact that each one is unique. Unfortunately, a malicious attacker could potentially duplicate a tag with the same identifier. A chip with

<table>
<thead>
<tr>
<th>Feature</th>
<th>NTAG210</th>
<th>NTAG212</th>
<th>NTAG213</th>
<th>NTAG215</th>
<th>NTAG216</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usable Memory (Bytes)</td>
<td>48</td>
<td>128</td>
<td>144</td>
<td>504</td>
<td>888</td>
</tr>
<tr>
<td>32 bit Password</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UID Ascii Mirror</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>24 bit Counter</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Originality Signature</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fast Read Command</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NFC Forum Type 2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>7 Byte Unique ID</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
this originality signature has a mechanism to allow the reader to acknowledge that the tag is genuine.

- Fast read command allows the device to read several pages of data from the tag faster. Not necessary for us.

- NFC Forum Type 2 guarantees the compatibility of the tags with the NFC standard, a feature we obviously need.

- A 7-byte UID means there are \(2^7 \cdot 8 \approx 72 \times 10^{15}\) unique identifiers, so we don’t have to worry about duplicate UIDs for a long time (all those tags are enough to cover a third of the earth’s landmass!).

Judging by the specifications, any model from NTAG210 onwards is adequate for our purposes. Also, we have to consider that these tags are available in a lot of different supports: paper stickers, PVC disks, cards, keycards, fridge magnets and wristbands, among many others.

In Krimtrack, one of the NFC stickers uses will be as a way of tagging different zones, so the guards can scan them as proof that they reached that zone, for position reporting purposes. Therefore, we need tags that can be attached to buildings: stickers are more concealable and easy to attach and remove but PVC disks are more resistant against vandalism, although more expensive. We think paper stickers are our best option for this case until circumstances prove otherwise.

Another application of NFC tags will be user identification. Instead of the usual username / password combination we will increase the system’s security by using a two-factor authentication: an NFC card plus a PIN. We will use a keycard that the guards will carry as the authentication token. Unfortunately, there are no keycards with a NTAG21x chip, so we will settle for the NTAG203. However, the only loss is the originality signature, and in the case of guards is easier to keep them under control (schedules, GPS location, etc...).

### 3.4 Mobile terminal application

The mobile application must provide the guards with all the necessary tools for their job. The idea is to replace all their heterogeneous equipment with one single device, the mobile terminal. Furthermore, we seek to automate administrative tasks with three goals in mind: ease the administration workflow, accelerate it and prepare the environment to extract information from it (via machine learning, see section 3.8).
3.4.1 Features

The most relevant features included in the mobile application are:

1. Authentication
   
   1.1. Two-factor user authentication by NFC card and PIN
   
   1.2. Connection encrypted with 2048-bit RSA

2. Communication
   
   2.1. PTT communication between users
   
   2.2. Message receiving from coordination center
   
   2.3. Shortcut list to important phones (control center, police, hospital, ...)

3. Geolocation
   
   3.1. Visualization on map
   
   3.2. User and nearby peers’ location
   
   3.3. Schedule locations and “next zone on schedule” highlighted

4. Schedule reporting
   
   4.1. List of all schedule points and their estimated time of arrival
   
   4.2. NFC tagging on schedule points

5. Event reporting
   
   5.1. Real-time event reporting, including emergencies, incidents and other events
3.4.2 Navigation map

The mobile application has been designed to work optimally in 4.5+ inch smartphones. We tried to minimize cumbersome actions in a mobile terminal, like typing, to streamline the activity as much as possible. The interface, as well as a sketch of the navigation map, can be found in figure 3.6.

3.4.3 Architecture

The application uses a classical three-tier architecture which separates the user interface, the domain and the data/communication layers (figure 3.7). It also follows a thin client paradigm, making the domain layer very simple, and relying instead on the server to do any heavy data processing. This has several advantages:

The most important factor is security. Given Krimtrack’s nature, special care has been taken to prevent security breaches. Since the mobile terminal can be easily compromised (can be stolen, for instance), we don’t want any critical procedure to be present on its software. Therefore, a possible attacker is greatly impeded from exploiting any vulnerability affecting the mobile terminal that has repercussions on the global system.

Thin clients also have an inherent administrative advantage. By the specification, all business logic and communication must be controlled and monitored by the coordinator. By delegating the heavy processing to the server we make coordination aware of all the ongoing operations, which would be hidden if the terminals did it on their own. As another administrative boon, if this data processing needs to be changed, tuned or expanded, a single modification on the server would affect the whole system. Otherwise every terminal would need to be changed as well.

Lastly, it is interesting to note the variety of external systems the application has to interact with: a camera, the telephone network, GPS satellites, NFC tags, other terminals via PTT and, of course, the communication server. These varied systems form an orchestrated environment, which we can see as heterogeneous sources of data we can analyse to gather important hints to further improve the user job via machine learning. This paradigm works better if we use each mobile terminal as a "sensor aggregator" of sorts and the server as the inference agent of all that data instead of letting each terminal perform its own inference.
Figure 3.6: Krimtrack mobile application interface and navigation map.
Chapter 3. *Krimtrack architecture*

![Mobile application simplified class diagram](image)

**Figure 3.7:** Mobile application simplified class diagram.
3.4.4 Cache database schema

During its execution, the mobile application will need certain data that will be mostly unchanging (or very sporadically changed) such as information about users (name and picture), zones (name and location), event types (type and name) and phone numbers (name, number and icon). Since we want to minimize bandwidth usage and this data can be potentially very large, we will use Android’s internal database manager (SQLite) as a cache memory, as pictured in figure 3.8.

Once the user is logged in the initialization will proceed as follows: the terminal will ask the server for the SHA256 hash of a particular set of data (zones, event types or phone numbers). The server will send it, and the terminal will compare it to the one in the database; if it’s the same, the terminal will load the data it already has from the cache. Otherwise it will ask the server for the full data and, once it receives it, it will be stored in the cache along its hash.

The user cache works slightly different: it stores a hash for every single user, so a change in one user will not imply sending the full data set of users to all the terminals.

Figure 3.8: Mobile application SQLite cache database schema.
3.5 Communication server

This module is the core component responsible for the handling and communication between connections, as well as maintaining the state of the whole system (like users’ positions or current ongoing schedules).

3.5.1 Architecture

Connections are implemented by websockets, so access to the server is provided by a websocket endpoint (KTWSEndpoint). The simplified class diagram of the server is pictured in figure 3.9.

Websocket is a technology that provides a bidirectional, full-duplex connection over a TCP socket. It is used by highly responsive applications, often in web browsers (the latest versions of most web browsers support it) but any client/server application is possible.

The server, alongside the coordinator web application and the database, are encapsulated in a single JavaEE application. That allows us to easily deploy, maintain and update all the system with a single WAR (Web Application aRchive) file.

3.5.2 Krimtrack protocol (KTP)

The communication itself is done by a custom protocol we call KTP (KrimTrack Protocol). It is a simple message-based protocol, each message composed by different fields separated by a “line feed” (\f), a non-printable character. Characters on the message are encoded in UTF-8.

As we said before, websockets are full-duplex and asynchronous. This allows the protocol to be full-duplex and asynchronous as well, so connections don’t have to store information about the state of the conversation. This feature makes the communication more modular and, consequently, more extensible.

Other communication protocols were considered first and discarded:

1. At first we thought about using plain sockets directly, in order to minimize any communication overhead with more complex protocols. During development it became clear that the complexity of dealing with such low-level objects was not worth the marginal gain in performance.
2. Once sockets were discarded we considered the RMI protocol. RMI is also integrated in the Java EE API (like websockets and JSP) so it was a natural choice. Its most important advantage was the simplicity of the code: you communicate with a remote server with merely a function call, just like a local object. The problem,
though, is the complex configuration (via XML files) that is needed beforehand, plus the fact that the serialization needed to transmit objects has compatibility issues between Java virtual machine versions. Therefore it would be an option difficult to maintain.

A complete list of the KTP commands is found on appendix A.

3.6 Database

Krimtrack’s database is implemented as a MySQL relational database. This was a choice primarily based on ACID guarantees (Atomicity, Consistency, Isolation and Durability). Other alternatives include so-called noSQL databases, which have better performance on massive data but the software using them has to be more complex to make up for the lack of ACID guarantees. Details about the database implementation are pictured in figure 3.10.

3.7 Coordination web application

The coordination web application will be used by the coordinator to monitor and help the field users in their job. The coordinator will be using this module intensively, for hours at a time in a usual setting. Hence the importance of a usable, responsive graphical interface, to present the information as quickly and ergonomically as possible.

3.7.1 Features

The coordination application includes the following features:

6. Field user monitoring
   6.1. Field user position
   6.2. Field user route
   6.3. Coordination-to-user messaging

7. Zoning
   7.1. Zone coverage
   7.2. NFC tags location
7.3. Incident risk prediction (see section 5.4)

8. Event reporting

8.1. Real-time event reporting

8.2. Visualization of events

8.3. Support for incident’s pictures and related people information
9. Emergency warnings

9.1. Real-time emergency reporting

9.2. Acoustic alarm for new emergencies

3.7.2 Interface

The coordination web interface has been designed with two usability principles in mind:

1. Intuitive use and navigation
2. Immediate access and visualization of all important information.

This last point is specially important since a coordinator needs to have available all the relevant information at all times to be effective. A screenshot of the interface can be seen in figure 3.11.

![Krimtrack coordination interface](image)

Figure 3.11: Krimtrack coordination interface.

3.8 Machine learning advisor

Finally, the focus of the project is the machine learning module. It has been designed to be independent from the rest of the system, in order to be used in a variety of ways. The potential possibilities are endless: customized advise for field users, optimal course
of action for emergencies, advice on guard allocation, staff evaluation, route design and many more.

For now we will show a simple example of what could be done to illustrate its capabilities: we will use it as an advisor for zoning classification on the coordination subsystem and as a stand-alone program to analyse datasets manually.

The details of the predictive algorithms and models will be discussed in the next chapter.

3.8.1 Architecture

Internally this module is divided in three big parts: data representation, filters and predictive algorithms. A complete (yet simplified) class diagram can be seen in figure 3.12. For more information on all the terms that will appear in the following paragraphs, see the next chapter for a more thorough explanation.

The data representation package includes the building block of the module: the instance. Instances are collected in a data set, which has a header specifying each attribute: number, type, possible values and more. The counting tree was designed as an auxiliary data structure to optimize the module performance but it only was beneficial in very special circumstances, so it is currently disabled.

There are also a small collection of filters. As we will see, data preprocessing is critical for a good predictive accuracy, and one of the preprocessing possibilities is data filtering.

Finally, the predictive models and their training algorithms are the main interest of this project. To test a large range of prediction methods we implemented a total of eight well established classifiers, excluding auxiliary ones:

- Naive bayes
- C4.5 decision tree
- Artificial neural network
- K-nearest neighbour (k-NN)
- Logistic Regression
- AdaBoost.M1
- C4.5 random forest
- Stacking
A simplified class diagram of the module can be found in figure 3.12. The highlighted classes are the machine learning algorithms: classifiers in green and regressors in red.

One important note on these classifiers is that its implementation was based only on their theoretical basis. We wrote them from scratch, and never used or adapted any other code as a sample. One of the objectives was to learn the intricacies of these algorithms deeply and be able to replicate them only with a rough description. Also, besides these eight, we also tried to implement other algorithms designed by my supervisor and myself, with varying degrees of success that will be discussed later on this document.

Even though we focused mostly on classification we also implemented two regression models (linear regression and a k-NN regressor) for completeness’ sake. We will explain these in the next chapter, and study their results in chapter 5. In this last chapter we will also see how we integrate this module with the rest of Krimtrack, as a first application among the many possibilities it offers.
Chapter 3. Krimtrack architecture

Figure 3.12: Machine learning module simplified class diagram.
Chapter 4

Machine Learning

In this chapter we will introduce the concept of machine learning and explain all the related procedures we will use and implement for data analysis. Results from the experiments on these algorithms are found on the next chapter.

It needs to be stated that the algorithms within this chapter were chosen by following a particular research path. We started with just three classifiers (naive bayes, decision trees and neural networks) but in order to improve these results on areas where these models failed we progressively developed the rest of the algorithms. This progression led us to the implementation of new techniques, which produced new results to improve upon. This iterative feedback proved to be a highly beneficial tool that led us to new results that have ultimately resulted in the variety of algorithms herein.

Since this module aims to be part of a commercial product, we will not present actual code in this document to preserve confidentiality. However, all the important ideas and algorithmic steps are presented with enough detail to be easily understood and reproduced.

4.1 Machine learning 101

4.1.1 Concepts and terminology

Machine learning is a field of Artificial Intelligence that studies the construction of automated learning systems. This learning is done by induction: given a set of data it tries to build the best model possible that explains that data.

Before we start, we will define the next terms:
- An instance is a single example from the phenomenon we are trying to learn. Labeled instances have a label attached to them while unlabeled instances do not.

- The goal of a predictive model is to estimate a label given an (unlabeled) instance. If these labels are finite and discrete we have classification models and if they are either infinite or continuous we have regression models.

- A label, also called class in a classification context, is the value from an instance we are trying to predict. As abuse of notation, we also use the term “classes” to refer to the number of values the class has.

- A data set, and hence all of its instances, are described by a series of attributes, also called features. Each attribute has a type (nominal or numeric) as well as a list of possible values (if nominal).

Formally, given a data set \(D\) with a feature set \(F = \{X_1, X_2, ..., X_n\}\) and a set of labeled instances \(I = \{I_1, I_2, ..., I_N\}\), each instance has a value for each feature and for its label \((I_i = \{X_1 = v_1, X_2 = v_2, ..., X_n = v_n, Y = y\})\). These values can be missing, which usually means a special treatment in both preprocessing and model generation, as we will see.

For example, take the data set from figure 4.1. It has four features (outlook, temperature, humidity and windy) and the binary class with values N or P. Each row represents a weather forecast for an area and the class represents whether a tennis match will be played (P) or not (N). The second feature, for example, has three possible values: hot, mild and cool, besides the red cells which mean missing data.

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>N</td>
</tr>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>N</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>?</td>
<td>false</td>
<td>P</td>
</tr>
<tr>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>P</td>
</tr>
<tr>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>P</td>
</tr>
<tr>
<td>rain</td>
<td>?</td>
<td>normal</td>
<td>true</td>
<td>N</td>
</tr>
<tr>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>P</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>N</td>
</tr>
<tr>
<td>sunny</td>
<td>?</td>
<td>normal</td>
<td>?</td>
<td>P</td>
</tr>
<tr>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>P</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>P</td>
</tr>
<tr>
<td>overcast</td>
<td>?</td>
<td>high</td>
<td>true</td>
<td>P</td>
</tr>
<tr>
<td>overcast</td>
<td>not</td>
<td>normal</td>
<td>false</td>
<td>P</td>
</tr>
<tr>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 4.1: A data set example.
Note that there is no practical distinction between the features and the class; the class is simply the feature we are trying to predict. In figure 4.1 we could try to predict the temperature forecast as a function of the outlook, the humidity, whether it’s windy or not and whether the tennis match was played or not, even though it doesn’t make much practical sense to do so.

For the model generation stage or training we use these labeled instances to infer a model \( M \), which will be used to estimate labels \( \hat{y} \) from future unlabeled instances, that is \( \hat{y} = M(v_1, v_2, \ldots, v_n) \). Since we will be focusing mostly on classification, along this chapter we will refer most of the time to the labels as “classes”, even though many concepts apply to regression as well.

### 4.1.2 Types of learning

Within the AI field there are several types of learning:

**Deductive** systems work by deriving facts from a known set of premises. Those facts will become part of the new premises and we iterate this deduction cycle until we reach a desired fact. This process requires an initial axiomatic set elaborated by an expert in the field with the aid of a knowledge engineer. If this expertise is not available but previous examples are, we use **inductive** learning instead.

Another categorization is supervised vs unsupervised learning. **Supervised** learning takes into account the labels of the data set in the training, while **unsupervised** doesn’t. Machine learning often uses, as we will do in this project, supervised algorithms while data mining users are more interested in the output of unsupervised techniques such as clustering, which are beyond the scope of this document.

![Supervised learning workflow](image)
4.1.3 Metrics to evaluate a model

There are basically three properties that we must take in consideration to consider a model valuable.

The first one is clear: the accuracy or predictive power is the main reason why we are using these models. Our estimations must be as good as possible, with the lowest error we can achieve.

There is a subtle detail though: we are learning from a limited sample from a larger population, but the model’s accuracy must be consistent with any other sample from the same population. That’s why our learning algorithms must take generality as an important factor. If we try too hard to fit a model to a given training set we risk overfitting, which will give us an optimistic accuracy estimation on our training set but will have low accuracy on future unseen data.

Finally, we must look for simplicity. As Occam’s razor states, we should not add complexity to a model if it doesn’t return significantly better results. In other words, of unknown origin: “Everything should be made as simple as possible, but not simpler”.

These three principles are intrinsically related:

- A simple model will have fewer constraints, which makes it more likely to generalize to the rest of the population.
- Good generalization implies that the model will have good accuracy with many different samples.
- A simpler model makes fewer assumptions and therefore it’s more robust and stable with new data. This boosts accuracy on unseen data.

Take figure 4.3 for instance. The authors of the image made a function of a single attribute corresponding to its cosine (the dotted line), sampled some points uniformly and added a small random noise. The machine learning algorithm must find the best approximation to the original cosine. If we use an insufficient model we get the upper left result, a curve that doesn’t resemble the original model at all (underfitting). If we reach enough complexity (but not more) we have the upper right graph, a very good fit to the model. But if we increase the complexity we will get models like those in the bottom row: they have smaller mean squared error than the second one (the one in the lower right even has zero error!) but it’s clear they don’t fit the original model quite as good as the second one. Therefore, it is crucial to strike a balance between power and simplicity.
4.1.4 Machine learning limitations

At first sight, machine learning seems like an almost utopian field. Without knowing anything about its semantics, you can enter data in an algorithm and a model will predict the future for you. There are certain limitations to consider, though:

- Data preprocessing is incredibly important. In a context of Big Data, data sets are full with missing values, errors, unbalanced classes, irrelevant or redundant attributes, ... It is imperative to perform a first analysis before training to address these problems.

- Evaluation of the model can be misleading. We will discuss this subject later on (section 4.7.1), but if we only focus on accuracy percentages we might not see other problems in the model.

- Obviously we need a data set to learn from. That limits our learning to phenomena that have already occurred often enough to have a sufficient statistical source. Plus, the specific value (label) we are trying to estimate for this data has to be available as well.
If the class is a subjective value (see nursery data set further on) the algorithms will only learn the subjective evaluation done in the examples. The result will not be any more objective even if it’s a machine making the decision. It is important then that the data in these sets is as realistic as possible: an objective measure is preferable to a subjective one.

### 4.1.5 Learning procedure

Summarizing the whole procedure from the very beginning, machine learning analysis should follow these stages:

1. Obtain the data.
2. Perform a first visual assessment of the data set (number of attributes, instances and classes, class distribution, amount of missing values, scatter plots between variables, ...).
3. Filter the data either manually (e.g. removing obviously useless attributes) or using a filter.
4. Train the model with the processed data.
5. Test the model.
6. Review the different error measures.
7. Try to analyse the model (if feasible) to learn more about the data’s statistical nature.

### 4.2 Data sets used

The experiments and results we will document in chapter 5 will be done using six different data sets (plus another one for regression) to cover as much different scenarios as possible. These, as well as many others, are available in the UCI machine learning repository[1].

To prove how machine learning can be useful in very different areas, these sets are from fields as diverse as medicine (audiology, column, breast cancer), signal processing (fourier), finance (credit) or social sciences (nursery). Also, some of them have peculiar traits that are interesting to see how they affect the learning procedure. Simpler data sets exist, but we discarded them since they are only simple academic exercises that bear little importance for this project.
We will be using these data sets as a test for our machine learning algorithms, but we have to keep in mind that these sets are not representative of most applications in the real world. In one hand, they are small (a few thousands of instances at most), while for our purposes in Krimtrack, data will be generated constantly so instance count will never be a limiting factor in the long run learning. Also, the data is presented neatly packaged in predefined attributes already decided for us, which won’t be the case for this project. This is why we will dedicate chapter 6 to simulate the full process, with special focus on data gathering, preprocessing and model interpretation.

To visualize the data sets we will use the open source data mining toolkit Weka[2]. The full headers from these sets are available in their corresponding bibliography entries.

### 4.2.1 Audiology

- **Number of instances:** 226
- **Number of instances with missing values:** 222
- **Number of attributes:** 69, all nominal
- **Target:** 24 classes

This data set is built with the intention of automatically diagnosing audiology injuries[3]. There are 24 different diagnosis and 69 attributes to evaluate, most of them boolean. As it’s clear from figure 4.4, this set is extremely unbalanced: most of the instances belong to only five classes, and many others only have one example. This fact, along with the high amount of missing data, makes this data set specially difficult to work with.

![Figure 4.4: Audiology data set class distribution.](image)
4.2.2 Column

- **Number of instances**: 310
- **Number of instances with missing values**: 0
- **Number of attributes**: 6, all numeric
- **Target**: 3 classes

These instances represent the clinical history of different patients with vertebral column problems [4]. It is similar to the breast cancer set, medical in nature and quite small (310 instances with 9 attributes, with three classes (diagnostic results) this time). In this case it looks balanced enough though (figure 4.5).

![Figure 4.5: Column cancer data set class distribution.](image)

4.2.3 Breast cancer

- **Number of instances**: 286
- **Number of instances with missing values**: 9
- **Number of attributes**: 9, all nominal
- **Target**: 2 classes

In this case the set is formed by the clinical history of different women who have had breast cancer, and the objective is to predict whether the cancer will come back or not[5]. It is a quite small data set (286 instances with 9 attributes and just two classes) so it should be manageable in terms of running time. We can see in figure 4.6 that the data set is quite unbalanced: 201 negative instances versus 85 positive ones.
4.2.4 Fourier

- **Number of instances**: 2000
- **Number of instances with missing values**: 0
- **Number of attributes**: 76 (12 after filter), all numeric
- **Target**: 10 classes

This data set involves optical character recognition [6]; more specifically, handwritten digits (from 0 to 9) found on utility maps. Each digit has been represented by hundreds of features, but separate data sets were created with different subsets of these features. We deliberately chose the most difficult to use for prediction purposes, the one with the fourier coefficients of the digits.

Since it has a large amount of instances, attributes and classes we applied an attribute filter to make the data more manageable in execution time, reducing it to 12 attributes with no significant loss on accuracy. As we will discuss in 4.3 usually applying a filter to a data set before training introduces bias in the learning, but since the set is large enough we’ll assume the bias is negligible.

On the other hand, the set is perfectly balanced among all ten classes, though that doesn’t mean that all classes will be equally difficult to learn. The distributions are different for each digit, as can be seen in the scatter plot in 4.8. It’s easy to see that some classes, (class 1, for instance, which equals the digit 0) is notably isolated from the rest, while others (like class 7 and 10, that is, digits 6 and 9) are heavily overlapped. We can expect then to learn digit 0 easily while 6 and 9 will be more complicated (which is understandable, since fourier coefficients are rotation-invariant; 6 and 9 are virtually indistinguishable using those features only).
4.2.5 Credit

- **Number of instances**: 1000
- **Number of instances with missing values**: 0
- **Number of attributes**: 20, 13 nominal / 7 numeric
- **Target**: 2 classes

In this case we look at the field of finance and decide whether to grant a credit or not based on german banking data[7]. We have a good amount of instances, no missing data and just 2 classes (good or bad).

This data set illustrates how powerful and useful machine learning can be[8]: to determine whether to grant a credit or not they used to have a financial specialist that analyzed the difficult cases and took a decision. Of all the credits he granted, 50% defaulted. Since
these difficult clients are often the most active, they become a priority to the bank, so they can’t just deny all these petitions. When they made a machine learning advisor to take on these cases the accuracy raised to 70%, with all the increased revenue this implies.

![Credit data set class distribution.

4.2.6 Nursery
- **Number of instances:** 12960
- **Number of instances with missing values:** 0
- **Number of attributes:** 8, all nominal
- **Target:** 5 classes

The last classification set deals with data of a social nature. The labeling is a subjective measurement on the ranking of applications for nursery schools in Ljubljana, Slovenia[9]. The ranking has five levels, from lowest to highest priority.

The class balance is heterogeneous: three classes are very balanced while the other two have a very low presence, including one with just 2 instances (out of a total of 12960). In this case we can guarantee that this unbalance is inherent to the data: by looking at the attribute distribution we can see that all possible combinations of values are present. In other words, we don’t have just a sample but the whole population in the data set.

It is clear that this fact makes machine learning unnecessary in a practical sense, at least for prediction purposes, since we can just memorize the instances and we don’t need to generalize. It is nonetheless interesting to see what happens in this case and observe the created models to extract objective conclusions about how this subjective ranking works.
4.2.7 Imports

- **Number of instances**: 201
- **Number of instances with missing values**: 41
- **Number of attributes**: 25, 11 nominal / 14 numeric
- **Target**: Numeric

Besides the six previous data sets for classification experiments, we will consider an additional one for regression. Its subject is import cars: the objective is to estimate their price given a list of car characteristics[10].

The prices go from $5118 to $45400, but as we see in the histogram most car prices lie in the low end of this range.

4.3 Filters

Even if the data sets we are working with are already processed, they are not yet optimal. A perfect predictive model would be able to ignore irrelevant attributes, or
learn all instances equally well despite unbalanced classes, but the heuristical nature of the training algorithms imposes us some limits. Basically, we use filters to remove any spurious effect from the training, for example:

- Enhance generalization by reducing overfitting to irrelevant attributes.
- Make a model simpler, and therefore more easy to understand.
- Shorten the training time.
- Reduce bias and, consequently, improve accuracy.

We should be careful when applying filters. As we will discuss in section 4.7.5, for a good accuracy assessment we need to isolate test data from training data or else we may incur in a biased estimation. Thus, we must not apply filters on the original set; we first have to decide and execute the testing methodology, split the instances and then filter only the training set. Otherwise information from the test set will be used to filter the training set, and the isolation principle will be broken. This warning can be relaxed if we have a lot of instances, as the bias in this case can be considered negligible.

### 4.3.1 Attribute filters

Attribute filters, also called feature selectors, seek to remove irrelevant or redundant features of a data set. Besides filters, there are also *wrappers*. The objective of both is to evaluate how good a subset of features is; the difference is that wrappers use a training algorithm to do the evaluation while filters use other heuristics, independent of any machine learning algorithm. In this project we will only use filters.

As an example of feature selection we have figure 4.12. It has two attributes $X_1$ and $X_2$ and three classes represented as colors red, green and blue. We can see that if we want to separate the dots by color feature $X_2$ is irrelevant and can be safely removed.
4.3.1.1 Correlation-based feature selection (CFS)

This filter[11] uses correlation as a heuristic metric and it’s based on two principles:

1. Good attributes are highly correlated with the class (high relevance).
2. Good attributes have low correlation with other attributes (low redundancy).

With this in mind, the author of the filter derives the following fitness heuristic:

\[ h = \frac{k r_{zi}}{\sqrt{k + k(k-1) r_{ii}}} \]  \hspace{1cm} (4.1)

where \( r_{zi} \) is the average of the correlations between the attributes and the class, \( k \) is the number of features and \( r_{ii} \) is the average correlation between attributes. High correlation with the class \( r_{zi} \) increases \( h \) while high correlation between features \( r_{ii} \) lowers it.

There are different measures we can use as the correlation in equation 4.1. We decided to use symmetrical uncertainty for its simplicity and good empirical results in the original paper. Symmetrical uncertainty is defined as:

\[ U(X, Y) = 2 \left[ \frac{I(X;Y)}{H(X) + H(Y)} \right], \]  \hspace{1cm} (4.2)

where \( H(X) \) is the entropy of \( X \) and \( I(X;Y) \) is the mutual information of \( X \) and \( Y \). This measure is symmetrical and normalized between 0 and 1, and therefore is apt to be used as a correlation measure.

Algorithm 4.1 CFS

**Input:** a data set with \( M \) classes

**Output:** 
- initialize bestSet
- compute the heuristic \( h \) for bestSet

**repeat**
- update bestSet with hill climbing step
- compute the heuristic \( h \) for bestSet

**if** current set score \( \leq \) previous set score **then**
- \( bestSet \leftarrow \) the previous set

**until** current set score \( \leq \) previous set score

**return** bestSet
With function 4.1 we can test a subset of features and obtain a score. But we need a search method to determine which subsets we will test (since we can’t feasibly test them all) and keep the best among them. Among many alternatives we settled for three hill climbing variants:

1. Starting with an empty set, we add features one by one until the heuristic can’t increase any more.

2. Starting with the full set, we remove features one by one until the heuristic can’t increase any more.

3. Starting with a random set, we add or remove features one by one until the heuristic can’t increase any more.

### 4.3.1.2 CFS per class

We thought another alternative would be applying the previous filter by classes. The intuition we have is that perhaps by analyzing the data in a binary context (belongs to class, doesn’t belong to class) the filter will be more accurate.

**Algorithm 4.2 CFS per class**

**Input:** a data set with $M$ classes  
**Output:** the data set has attributes filtered

```plaintext
for each class $c$ do  
    apply binary transformation to data set on class $c$  
    use CFS on binary data set to get filtered attributes  
    mark values from filtered features as unknown

for each feature $f$ do
    if feature $f$ is filtered in all classes then
        remove feature $f$
```

![Figure 4.13: Geometric example of binary transformation of class green.](image-url)
4.3.2 Instance filters

Removing instances is not as important as removing features, but it might help too. Even though empirical research show that instance filtering doesn’t improve the model’s accuracy (it often decreases slightly), it is useful to shorten training times when the number of instances is very large. With that said, the only relevant application of instance filtering is in a k-NN classifier, since its classification time is dependent on the size of the training set and it’s difficult to predict the outcome of instance filtering in other algorithms.

Figure 4.14: Geometric example of instance filtering.

4.3.2.1 Fast Condensed Nearest Neighbour rule (FCNN rule)

This filter[12] was designed as a training step for the k-NN classifier. The goal is to choose a subset of instances just large enough to have comparable results with this algorithm. We will see in section 4.4.3 how this classifier works, but if we visualize each instance as a point in an euclidian geometric space we want to obtain a reduced instance set with the same rough global structure.

FCNN uses the concept of Voronoi cells of a point \( p \in S \), which are defined as the set of all points that are closer to \( p \) than to any other point in \( S \). From this set we need to choose a representative in each iteration. The author suggests two alternatives: either choose the closest point of the cell to \( p \) or its centroid (the point closest to the theoretical geometric “center” of the cell). For simplicity we will choose the former option.

Algorithm 4.3 FCNN rule

Input: a training data set \( T \)

Output: a consistent subset of \( T \)

\[
S \leftarrow \emptyset \\
\Delta S \leftarrow \text{centroids of } T \\
\text{while } \Delta S \text{ not empty do} \\
\quad S \leftarrow S \cup \Delta S \\
\quad \Delta S \leftarrow \text{representatives of the Voronoi cells from instances in } S \\
\text{return } S
\]
4.3.2.2 Entropy-based instance selection

The previous instance filters were centered on a distance metric. We’ll try to use an entropy measure this time: we’ll build a score for each instance (weighting each feature by its correlation as computed by the CFS) and we’ll remove a fixed percentage of them, \( p_R \), with the lowest scores (the ones with least uncertainty).

**Algorithm 4.4** Entropy-based instance selection

**Input:** a training data set \( T \)

**Output:** a filtered subset of \( T \)

\[
S \leftarrow T
\]

for each feature \( f \) do
    compute \( p_{r_j} \) using the CFS filter correlations and normalizing

for each instance \( i \) do
    compute instance score as \( \sum_{j=1}^{M} (p_{r_j} \cdot p_{v_j(i)}) \log (p_{r_j} \cdot p_{v_j(i)}) \)

remove \( p_R \% \) instances with the lowest scores

return \( S \)

where \( p_{v_j(i)} \) is the probability of the value of the \( j \)th feature of instance \( i \) and \( p_{r_j} \) is the percentage of the correlation of feature \( j \) among the rest of features.

4.4 Classifiers

This section will describe the variety of classification algorithms we developed for their use in Krimtrack. Each classifier specifies two algorithms: the training algorithm to create the model and the algorithm that specifies how to use such model for prediction on a given instance.

4.4.1 Naive Bayes

A naive bayes algorithm tries to classify an instance by choosing the target value that maximizes its probability:

\[
y = \arg \max_c \{p(Y = c|X_1, ..., X_n)\} \quad (4.3)
\]

By Bayes’ theorem, this can be rewritten as:
\[ y = \arg \max_c \left\{ \frac{p(Y = c) p(X_1, ..., X_n \mid Y = c)}{p(X_1, ..., X_n)} \right\} \] (4.4)

Since the denominator is independent of the class, we can ignore it:

\[ y = \arg \max_c \{ p(Y = c) p(X_1, ..., X_n \mid Y = c) \} \] (4.5)

Now, using the chain rule of conditional probability we have:

\[
y = \arg \max_c \{ p(Y = c) p(X_1 \mid Y = c) p(X_2, ..., X_n \mid Y = c, X_1) \} = \]
\[
= \arg \max_c \{ p(Y = c) p(X_1 \mid Y = c) p(X_2 \mid Y = c, X_1) p(X_3, ..., X_n \mid Y = c, X_1, X_2) \}
\]

Expanding the equation:

\[
y = \arg \max_c \{ p(Y = c) \cdot p(X_1 \mid Y = c) \cdot p(X_2 \mid Y = c, X_1) \cdot \ldots \cdot p(X_n \mid Y = c, X_1, ..., X_{n-1}) \}
\]

At this point we use the naive bayes simplification: we assume that the features \( X_i \) are statistically independent. With this, we get the final expression:

\[
y = \arg \max_c \{ p(Y = c) \cdot p(X_1 \mid Y = c) \ldots p(X_n \mid Y = c) \} = \]
\[
= \arg \max_c \left\{ p(Y = c) \prod_{i=1}^{n} p(X_i \mid Y = c) \right\} \] (4.6)

All we need are simple probabilities that can be easily estimated from the training set:
Chapter 4. Machine learning

\[ p(Y = c) = \frac{\# \text{ instances with } Y = c}{\# \text{ all instances}} \tag{4.7} \]
\[ p(X_i = v_i | Y = c) = \frac{\# \text{ instances with } Y = c \text{ and } X_i = v_i}{\# \text{ instances with } Y = c} \tag{4.8} \]

Nevertheless, the last probability estimation is not entirely appropriate for our purposes. If a value of a feature does not appear in the training set it will have a conditional probability of 0, so the product in equation 4.6 will discard all the rest of probabilities and return 0. Therefore, some correction factor needs to be introduced so all non-appearing feature values have a small, but greater than zero, probability.

To that end we will define a pseudocount, adding one to all the previous counts for each value:

\[ p(Y = c) \equiv \frac{\# \text{ instances with } Y = c}{\# \text{ all instances}} \tag{4.9} \]
\[ p(X_i = v_i | Y = c) \equiv \frac{(\# \text{ instances with } Y = c \text{ and } X_i = v_i) + 1}{\# \text{ instances with } Y = c} \tag{4.10} \]

Note that with this modification the numbers \( p(X_i | Y) \) we calculate are no longer proper probabilities but pseudoprobabilities. This is not problematic: we only want them as a comparison tool, and they are still useful to that end. Also take notice that if there are no instances of a particular class we don’t have any problem leaving \( p(Y = c) = 0 \), since we wouldn’t have any information about how to classify that value anyway.

**Algorithm 4.5** Naive bayes classifier training

**Input:** a training data set T

**Output:** a naive bayes classifier

for each class \( c \) do

    calculate \( p(Y = c) \)

    for each feature \( i \) do

        for each value \( j \) from feature \( i \) do

            calculate \( p(Y = c | X_i = j) \)

        \( M \leftarrow \{ \text{all the calculated probabilities} \} \)

    return \( M \)
Algorithm 4.6 Naive bayes classification

Input: a naive bayes classifier and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

Output: the class prediction for $I$

for each class $c$ do
  
calculate $s_c = p(Y = c) \cdot \prod_{i=1}^{n} p(Y = c | X_i = v_i)$

return class with highest $s_c$

This algorithm is very simple both in training and testing. In fact, it looks too simple to be accurate; the hypothesis of statistical independence between features is often false. Surprisingly, though, this learning method sometimes gives more accurate results than other, more complicated algorithms. This is due to basically two interesting properties:

1. Even if the features are not independent, considering them as such breaks the training set into a collection of manageable one-dimensional statistical distributions. Other algorithms have to consider the instances as multidimensional vectors and need a number of training examples that grow exponentially with the number of features. Naive Bayes needs a comparably lesser amount of training data to compute a good model.

2. Even though the probabilities might be inaccurate due to the assumption of independence, the only important result is how they compare to each other. As long as the right class has the greatest probability (as in equation 4.6) it will be classified correctly and the learning will be effective. In other words, naive Bayes will have good results as long as the error on the conditional probabilities does not distort the overall statistical class probabilities ranking.[13]

4.4.2 C4.5 decision tree

The C4.5 algorithm was created by Ross Quinlan[14], as an extension of ID3. It trains and builds a decision tree that helps in a decision making process in a very intuitive way. It consists on two types of nodes: decision nodes, which split the tree in as many branches as the number of values of a feature, and leaf nodes, which return a value as a prediction. The basic idea is to split the data set by different features repeatedly, obtaining new subsets, until we reach a point where we can identify a class with minimum uncertainty.

The measure the algorithm uses to determine which feature is the ”best” to split a data set is entropy. In this context we could understand entropy as a measure of the uncertainty or prediction difficulty in our data. If we see each feature as an information
source, we can calculate the entropy of $X_i$ in a set $S$, as well as conditional entropies, as follows:

$$H(S) = -\sum_{i=1}^{m_c} p(Y = i) \log(p(Y = i)) \quad (4.11)$$

$$H(S|X_k) = -\sum_{i=1}^{m_k} p(X_k = v_i) \sum_{j=1}^{m_c} p(Y = j|X_k = v_i) \log(p(Y = j|X_k = v_i)) \quad (4.12)$$

where:

$$p(Y = i) \equiv \frac{\text{# instances with } Y = i}{\text{# total instances}}$$

$$p(Y = i|X_j = v) \equiv \frac{\text{# instances with } Y = i \text{ and } X_j = v}{\text{# instances with } X_j = v}$$

Finally we have the concept of information gain. Given a set, we need to know how much uncertainty we remove by separating it using a particular feature. ID3 defines the gain (G) as:
The feature we choose is the one that maximizes the gain or, equivalently, the one that minimizes the conditional entropy. There is a problem with this metric, though, which is that it is biased towards features with many values. For example, a feature that uniquely identifies each instance would have a conditional entropy of 0, and thus a maximal gain, but it would be a poor choice for learning. C4.5 addresses this problem replacing the gain with the gain ratio (GR):

\[ GR(X_i, S) \equiv \frac{G(X_i, S)}{\text{SplitInfo}(X_i, S)} \]  

\[ \text{SplitInfo}(X_i, S) \equiv \sum_{j=1}^{m_i} \frac{|S_{ij}|}{|S|} \log \left( \frac{|S_{ij}|}{|S|} \right) \]  

where \( |S_{ij}| \) is the cardinality of the subset of instances where \( X_i = v_j \).

With these definitions and the training data as input, we use the following algorithm:

**Algorithm 4.7** C4.5 decision tree classifier training

**Input:** a training data set \( T \)

**Output:** a decision tree classifier

if more than a specified threshold of instances belong to the same class then
create leaf node predicting that class
else
for each feature \( i \) in \( T \) do
calculate gain ratio by splitting with feature \( i \)
if maximum gain ratio is below a minimum threshold then
create leaf node predicting the most popular class in set
else
split tree with decision node using feature providing maximum gain ratio
for each split do
apply algorithm recursively with split data set
return built tree (or subtree)
Algorithm 4.8 C4.5 decision tree classification

**Input:** a decision tree classifier and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

**Output:** the class prediction for $I$

```plaintext
currentNode ← root of decision tree

while currentNode is not a leaf do
    currentNode ← child of currentNode choosing the correct split

return class predicted on currentNode
```

The C4.5 algorithm makes some improvements over the ID3:

- Capability of treating continuous attributes by discretizing numeric values.
- Handling missing values.
- Post pruning: after the tree is built it tries to remove unnecessary branches to increase accuracy.

A particular strength of decision trees are their *accountability*: not only can they learn and predict a target value, but they can justify their decision in a human-friendly manner. In some environments that might be very desirable; you don’t have to rely blindly on an algorithm, you can see the tree structure, the branches it took to make that decision and how complex the decision process is.

As usual, to avoid overfitting and achieve good generalization properties, we must look for trees as simple as possible (with good accuracy).

A weakness in the algorithm lies in the treatment of its continuous features. C4.5 includes discretization, but how to determine the number of intervals and its borders for an optimal learning is not a trivial problem, and the performance of the algorithm might be critically dependent of it.

Beyond the standard C4.5 training, we’ll explore two variants that will be useful as the foundation of different ensemble classifiers we will see further below. The trees created will be standard C4.5 decision trees, it’s only the generation method that differs.

### 4.4.2.1 C4.5 random decision tree

The problem with standard decision trees is that they are deterministic and they choose always the best gain ratio for each split. This can be a problem if the gain ratio is not a good indicator of the overall best way to build an optimal tree. If this happens
the best apparent path can be misleading us from a better solution. To solve this we use random forests (described in section 4.5.1.1) whose basic elements are C4.5 random decision trees.

The generation of these trees is identical to regular decision trees except for one step: when deciding on a feature to split we won’t check all of them, just a different random subset for each split. This forces the tree to explore other solutions and diversify the variety of possible trees we can use with a given data set.

**Algorithm 4.9** C4.5 random decision tree classifier training

**Input:** a training data set $T$

**Output:** a decision tree classifier

*if* more than a specified threshold of instances belong to the same class *then*

create leaf node predicting that class

*else*

$T' \leftarrow$ random subset of features of $T$

*for each* feature $i$ in $T'$ *do*

calculate gain ratio by splitting with feature $i$

*if* maximum gain ratio is below a minimum threshold *then*

create leaf node predicting the most popular class in set

*else*

split tree with decision node using feature providing maximum gain ratio

*for each* split *do*

apply algorithm recursively with split data set

*return* built tree (or subtree)

### 4.4.2.2 C4.5 decision stump

The other C4.5 variant is the decision stump. Put simply, this classifier is just a decision tree that we allow to split at most once. It is not meant to be used as a single classifier (it is too simple to be accurate enough in most circumstances), but as a “rule of thumb” used in, for example, a boosting algorithm (section 4.5.2).
Chapter 4. *Machine learning*

Figure 4.16: An example of a decision stump used on sample data set seen on figure 4.1.

Algorithm 4.10 C4.5 decision stump classifier training

**Input:** a training data set \( T \)

**Output:** a decision stump classifier

if more than a specified threshold of instances belong to the same class **then**
  create leaf node predicting that class
else
  for each feature \( i \) in \( T \) do
    calculate gain ratio by splitting with feature \( i \)
    if maximum gain ratio is below a minimum threshold **then**
      create leaf node predicting the most popular class in set
    else
      split tree with decision node using feature providing maximum gain ratio
      for each split do
        create leaf node predicting the most popular class in split set
  return built tree

4.4.3 K-nearest neighbours (k-NN)

We can imagine (as we’ve been done before) data sets as geometric spaces where features are dimensions and instances are points in these spaces. A k-NN classifier[15] uses this representation as its basis to classify instances: it simply stores the training instances in the model and new instances are classified as the same class of the stored instance geometrically closest to it (its closest neighbour).

The algorithm is more general than that, though. We can set a parameter \( k \) to increase the number of neighbours (2-NN, 3-NN, ...) making the decision: when \( k > 1 \) each neighbour votes on a class and the majority vote is returned as the final prediction. To further improve the results, each vote is weighted with the inverse of the distance to the new instance, so closest points have more importance in the decision than further ones.
Another implicit parameter we should establish is which definition of distance we’ll be using. As it often happens in machine learning, no parameter is universally better since its performance depends on data peculiarities, but studies\cite{16} have shown that the euclidian distance works well as a default, even if several more sophisticated (and costly) methods exist to find an optimal distance for a specific data set.

We have to consider that a data set can have not only numeric attributes but nominal ones too. That is why we will use a hybrid euclidian metric with Hamming distance for nominal attributes: 0 if the values are equal, 1 if they are not.

\begin{equation}
    d(I_1, I_2) = \sqrt{\sum_{i=1}^{n} d_i^2(I_1, I_2)} 
\end{equation}

\begin{align}
    d_i(I_1, I_2) &= \begin{cases} 
    v_i^{(2)} - v_i^{(1)}, & \text{if } X_i \text{ is numeric} \\
    0, & \text{if } X_i \text{ is nominal and } v_i^{(2)} = v_i^{(1)} \\
    1, & \text{if } X_i \text{ is nominal and } v_i^{(2)} \neq v_i^{(1)}
    \end{cases} 
\end{align}

where \( v_i^{(j)} \) is the value of feature \( X_i \) of instance \( j \).

\textbf{Algorithm 4.11} k-NN classifier training

\textbf{Input}: a training data set \( T \)

\textbf{Output}: a k-NN classifier

\( T' \leftarrow \text{instanceFilter}(T) \)

store \( T' \) in a model \( M \)

\textbf{return} \( M \)
We mentioned in section 4.3.2 that for very large data sources this model is critically in need of a good instance selection filter. The larger the training set, the larger the amount of instances we have to compare each new instance with, so filtering is a critical step in the training process of this model.

**Algorithm 4.12** k-NN classification

*Input:* a k-NN classifier, a parameter $k$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

*Output:* the class prediction for $I$

1. for each instance $I_{stored}$ in the model’s stored instances do
   1. calculate distance between $I$ and $I_{stored}$
   2. $C \leftarrow k$ stored instances closest to $I$
2. return class as weighted vote of $C$

### 4.4.4 Artificial neural network

Artificial neural networks (ANNs) are computational models inspired by the structure of biological brains, capable of learning by reinforcement. An ANN (like in figure 4.18) is made of artificial neurons placed in several layers: the input layer (yellow neurons in the figure, associated to the input variables), a number of hidden layers (blue neurons) and an output layer (orange neurons, associated to the output variable(s)). ANNs only work with numeric values, so we need to represent all values of our instances (both input and output) in numeric form. Each neuron of a layer is connected to all neurons of the next layer.

![Figure 4.18: An example of a neural network.](image)

The codification of the inputs/outputs to numeric values, the number of hidden layers, and the number of neurons per hidden layer are all parameters that need to be established beforehand.
Once we know how an ANN is structured, we need to know how a single neuron works. A neuron has multiple inputs $p_i$ (i ∈ [1..R]), each one with an assigned weight $w_i$, and a single output $t$. Also, for reasons that will be explained later on, it has another parameter complementing this input called bias. With this structure (see figure 4.19) we can define:

$$ q = b + \sum_{i=1}^{R} w_i \cdot p_i $$  

(4.18)

$$ t = f(q) $$  

(4.19)

\[\text{Figure 4.19: The structure of an artificial neuron.}\]

where $f(q)$ is the activation function, which is yet another parameter of the algorithm. Usual choices for this function are shown in figure 4.20, but a necessary restriction for learning is that the function must be differentiable at all points.

Hence, the way to predict an instance using the network is simple: we insert the data in the input neurons and we calculate the outputs, layer by layer, until we get the output layer values.

\[\text{Figure 4.20: Examples of neuron activation functions: step functions (left column), bounded linear functions (middle column) and sigmoid functions (logistic function on the upper right and tanh(x) in the lower right).}\]
As we said, before we start the training process we need to determine the ANN structure: input and output representation, number of neurons and hidden layers and activation function, with the initial weights assigned as small, random values[17]. The goal of the learning stage is, then, adjusting these weights by following the next procedure:

1. Calculate the network output for a single instance.
2. Calculate the error (actual output - real output).
3. Adjust the weights of the ANN by propagating the error backwards, layer by layer.

This last step is called backpropagation and it’s the key to the ANN learning [18]. Backpropagation uses the gradient descent method for error minimization: it distributes the output error \((t - y)\) on a neuron among all its input weights, proportionally to the input value on that instance. In the output layer, we have:

\[
\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta (t - y) \frac{dy}{dq}
\]  

\[(4.20)\]

\[
\delta_{out} = (t - y) \frac{dy}{dq}
\]  

\[(4.21)\]

\[
w_i \leftarrow w_i + \Delta w_i
\]  

\[(4.22)\]

The equation 4.20 is called the generalized delta rule. \(E\) is the error, \(\delta_{out}\) is the error gradient, \(\frac{dy}{dq}\) is the derivative of the activation function’s output and \(\eta\) is the learning rate. The learning rate adjusts the accuracy and speed of the algorithm: a low learning rate slows the convergence toward a solution but a high rate decreases the accuracy of that solution. Typical values are between 0.2 and 0.4.

In a hidden layer we don’t have the target \(y\), so we must use the weighted sum of the next layer’s gradients to get the neuron error gradient, which we will call \(\delta_h\):

\[
\delta_h = \frac{dy}{dq} \sum_{k \in \text{outputs}(h)} \delta_k w_{hk}
\]  

\[(4.23)\]

where \(\text{outputs}(h)\) is the set of neurons to which the current neuron is connected to (i.e. all neurons in the next layer) and \(w_{hk}\) is the weight between the output of \(h\) and the input of \(k\).
Algorithm 4.13 ANN classifier training

**Input:** a training data set $T$

**Output:** an ANN classifier

split $T$ in $T_1$ (2/3) and $T_2$ (1/3)

build neural network $M$ with small random weights

repeat

for each training instance $I$ in $T_1$ do

classify $I$ in $M$ and obtain the outputs of the output layer

calculate errors on last layer as (calculated outputs - real values)

apply backpropagation algorithm with these errors

test $M$ on $T_2$

until the maximum number of epochs are reached or error on $T_2$ doesn’t decrease

return $M$

Algorithm 4.14 Backpropagation

**Input:** an ANN classifier $M$ and the errors of neurons of the output layer

**Output:** an updated ANN model

for each layer $l$ on $M$ starting from the last do

for each neuron $n$ of layer $l$ do

for each weight $w_i$ of $n$ do

calculate $\Delta w_i$ using the errors (eq. 4.20)

update $w_i$

calculate $\Delta$bias (eq. 4.20)

update bias

errors ← calculated errors from this layer (eq. 4.23)

return $M$

Algorithm 4.15 ANN classification

**Input:** an ANN classifier $M$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

**Output:** the class prediction for $I$

build neural network with small random weights

for each layer $l$ on $M$ starting from the first do

for each neuron $n$ on layer $l$ do

calculate output of $n$ with inputs from previous layer

$n_{max}$ ← neuron of the output layer with highest output

return class associated with $n_{max}$
ANNs are probably the most versatile non-ensemble models introduced in this document. With a good setup, neural networks are used with excellent results in many real world applications such as computer vision or resource management in companies like Google\(^1\).

An interesting feature of ANNs is their capability of online learning: being able to learn incrementally, one instance at a time, while it’s being used to predict instead of waiting to gather the whole set of data before the training starts.

Furthermore, neural networks’ modular structure brings several advantages more:

- **Simplicity**: each neuron output is easy to compute.
- **Locality**: each neuron only needs its inputs to produce an output, with no need to be aware of the network topology.
- **Parallelism**: neurons are inherently parallelizable and can be easily adapted to a distributed environment.

Like the rest of these algorithms, ANNs take a heuristic approach: the model it finds might not be the best one. The optimality of the network will depend greatly on the learning rate \(\eta\), but to avoid local minima in error optimization we introduce a possible improvement in equation 4.20 by adding another term: a **momentum rate** \(\alpha\).

\[
\Delta w_i(n) = -\eta \frac{\partial E(n)}{\partial w_i(n)} + \alpha \Delta w_i(n - 1) \quad (4.24)
\]

The idea is to prevent large oscillations in the learning process by making a weight update on iteration \(n\) dependent upon previous updates. With this method we initially boost the convergence speed towards a solution and slow down when we are close to it, accelerating the process without compromising its accuracy.[19]

A drawback of ANNs is their black box nature: it is difficult to analyze individuals neurons and how they affect the global result. It is challenging to work with a sound theoretical basis, having to rely instead on empirical results.

This opacity also works against the user. Unlike decision trees, ANNs have poor accountability properties. It is very difficult to understand (intuitively) why a neural network chooses a certain value or what is the contribution of a particular neuron to the whole network. Despite these inconveniences, neural networks are probably still the most widely used machine learning model, used in many circumstances for their excellent results.

\(^1\)http://googleblog.blogspot.ca/2014/05/better-data-centers-through-machine.html
4.4.5 Logistic regression

Logistic regression tries a more refined approach than other classifiers. It not only returns a class but the whole class probability distribution. And that is the reason why it has such a misleading name; despite its name it’s not a regressor, but a classifier that uses regression internally to estimate each class’ probability. Logistic regression by itself only works on binary data sets. As a previous step, then, we must perform a binary transformation for each class and estimate the logistic regression parameters for each one.

The conceptual basis for logistic regression starts with a linear regression estimate on probabilities (equation 4.25). The problem with linear regression in this case is that probabilities are bounded between 0 and 1, and therefore we would have to limit the output to this range (figure 4.21). In doing so, we no longer have the function that minimizes the squared error and we would have to redo the regression analysis with the new piecewise function. Such analysis can’t be done, though, since this function is not differentiable at all points: we would have to find another “smoother” function resembling this one. And that is exactly what the logistic function is (equation 4.26); sometimes it is referred to as “squashed linear regression”.

\[
p_{lin}(x_1, \ldots, x_n) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \quad (4.25)
\]

\[
p_{log}(x_1, \ldots, x_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}} \quad (4.26)
\]

**Figure 4.21:** Linear function, bounded linear function and logistic function.

The algorithm uses the principle of maximum likelihood (ML) to estimate the parameters \( \beta_i \). It starts with the assumption of a Bernoulli distribution, where \( p \) is the logistic function approximation (equation 4.26), \( m \) is the number of training instances and \( y_i \) is 1 if the instance \( i \) belongs to the class and 0 if it doesn’t.
\[ L(\beta; x) = \prod_{i=0}^{m} p_i^{y_i} \cdot (1 - p_i)^{(1-y_i)} \]

\[ \ln L(\beta; x) = \sum_{i=0}^{m} y_i \ln p_i + (1 - y_i) \ln (1 - p_i) \]

\[ \frac{\partial \ln L(\beta; x)}{\partial \beta_j} = \sum_{i=0}^{m} y_i \frac{1}{p_i} \frac{\partial p_i}{\partial \beta_j} + (1 - y_i) \frac{1}{1 - p_i} \frac{\partial (1 - p_i)}{\partial \beta_j} = \]

\[ = \sum_{i=0}^{m} y_i \frac{1}{p_i} p_i (1 - p_i) x_j - (1 - y_i) \frac{1}{1 - p_i} p_i (1 - p_i) x_j = \]

\[ = \sum_{i=0}^{m} x_j (y_i - p_i) = 0 \]

With the expression 4.27 we can construct a system of \( n + 1 \) equations, from \( \beta_0 \) to \( \beta_n \). Since \( p_i \) is an exponential function the system cannot be solved analytically and we must resort to numerical approaches. In this implementation we used the Newton-Raphson method[20].

**Algorithm 4.16** Logistic regression classifier training

**Input:** a training data set \( T \)

**Output:** a logistic regression classifier

for each class \( c \) do

apply binary transformation to data set on class \( c \)
calculate equation system (equation 4.27)
find solution \((\beta_0^{(c)}, ..., \beta_n^{(c)})\) to system using Newton-Raphson
store solutions \( \beta_i^{(c)} \) in model \( M \)

return \( M \)

**Algorithm 4.17** Logistic regression classification

**Input:** a logistic regression classifier \( M \) and an unlabeled instance \( I = (v_1, v_2, ..., v_n) \)

**Output:** the class prediction for \( I \)

for each class \( c \) do

calculate class \( c \) probability for instance \( I \) (equation 4.26)

optionally normalize probabilities to obtain distribution

return class with highest probability
This classifier is very useful specially if we need to use probability distributions instead of singular predictions. This will become very useful in stacking classifiers, where different layers of classifiers would propagate the error if we just used prediction values.

### 4.5 Ensemble classifiers

A more sophisticated approach to learning is to combine several of the previously mentioned base classifiers to complement each other and build a better model. These composite models are known as ensemble classifiers. Besides the algorithms my supervisor and I designed as new ensemble attempts, there are three main strategies we implemented: bagging, boosting and stacking.

#### 4.5.1 Bootstrap aggregating (bagging)

*Bootstrap aggregating*, often abbreviated as *bagging*, uses a set of different classifiers that vote on the final class decision. The final output is by majority vote. Bagging also has another particularity on its training: each base classifier is not trained on the training set but on a bootstrap sample of the set. This ensures that classifiers will have some diversity between them. We will explain in more detail what a bootstrap sample when we talk about bootstrap validation in section 4.7.5.4.

![Figure 4.22: A diagram of bagging classification.](image)

#### 4.5.1.1 C4.5 random forest

In section 4.4.2.1 we introduced the C4.5 random tree as a randomly limited version of the standard C4.5 decision tree. This limitation forces the tree to explore other paths in the solution space that appear to be less optimal using an information gain heuristic but can lead us to a better result in the end. We mentioned how a single random tree
is rarely better than a standard one, but gathering many of them in a bagging classifier improves noticeably the end model (called random forest for obvious reasons).

**Algorithm 4.18** C4.5 random forest classifier training

**Input:** a training data set \( T \)

**Output:** a random forest classifier

for each tree in the ensemble do

build a bootstrap sample from \( T \)

build C4.5 random tree with bootstrap sample

return \( M \)

**Algorithm 4.19** C4.5 random forest classification

**Input:** a random forest classifier \( M \) and an unlabeled instance \( I = (v_1, v_2, \ldots, v_n) \)

**Output:** the class prediction for \( I \)

for each tree \( t \) in the ensemble do

classify \( I \) using \( t \)

return class with the majority vote

### 4.5.2 Boosting

The research paper that introduced boosting\(^{[21]}\) posed a simple question: could a collection of weak learners (classifiers with an accuracy slightly greater than random guessing) become strong learners? Robert E. Schapire proved in this paper that it was indeed possible, and invented boosting as a method to exploit this possibility.

Boosting is based on the fact that some instances are harder to learn than others. By iteratively focusing on the harder instances we can construct an ensemble that gradually refines models to predict harder instances correctly, increasing the overall accuracy.

*Figure 4.23:* An example of boosting\(^{[22]}\). The first picture represents the first (poor) attempt at classification, but with more iterations (1, 3, 5 and 36 respectively) we can refine the classification.
4.5.2.1 AdaBoost

The original and the most popular boosting algorithm is AdaBoost, short for Adaptive Boosting, introduced by the same author[23]. It only works with binary data sets, whose two classes are represented by \{-1, +1\}, and its fundamental expression is:

\[
F(x) = \sum_{i=1}^{t} \alpha_i h_i(x),
\]

(4.28)

where \(h_i(x)\) is each base classifier hypothesis (prediction) and \(\alpha_i\) is the weight of each base classifier. From equation 4.28, we can get the boosting final hypothesis \(H\):

\[
H(x) = \frac{\sum_{i=1}^{t} \alpha_i h_i(x)}{\sum_{i=1}^{t} \alpha_i} \in [-1, +1]
\]

(4.29)

With this final hypothesis \(H\) we can predict the class \{-1, +1\} as sign\((H(x))\) and the confidence of the prediction as abs\((H(x))\).

To be used in AdaBoost, the classifiers must have two properties:

- They must be able to work with weighted instances.
- They must have an accuracy greater than 50% (but not much greater if possible).

We start with all instances with the same weight. We build a classifier, calculate its weighted error \(\epsilon_i\) (equation 4.30) on the training set and calculate the classifier coefficient \(\alpha_i\) (equation 4.31).

\[
\epsilon_i = \sum_{j=1}^{m} w_j I[h_i(x_j) \neq y_j]
\]

(4.30)

\[
\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \epsilon_i}{\epsilon_i} \right),
\]

(4.31)

where \(I[x]\) is an indicator function: 1 if \(x\) is true and 0 if it’s not. Finally, we recalculate each instance’s weight, increasing it if we misclassified the instance or decreasing it if
we didn’t, and lastly we normalize them all to form a probability distribution. If we are
on iteration \( j \), each instance \( i \) has its weight modified like:

\[
w_i \leftarrow w_i e^{-y_i \alpha_j h_j(x_i)}
\] (4.32)

Once finished, we start a new iteration with these updated weights. These weights and
coefficients are proved to reduce the training accuracy error exponentially, as long as we
can find a classifier with an accuracy greater than 50% in each iteration.

To avoid overfitting, it is recommended to choose very simple base classifiers. Usual
choices are decision stumps, introduced in 4.4.2.2.

**Algorithm 4.20** Boosting classifier training

**Input:** a two-class training data set \( T \)

**Output:** an AdaBoost classifier

initialize all instance weights to \( 1/m \)

while a classifier \( i \) with greater than 50% accuracy exists for \( T \) or the maximum
number of iterations have been reached do

train \( h_i \) with weighted instances

calculate weighted error (eq. 4.30)

calculate \( h_i \) coefficient, \( \alpha_i \) (eq. 4.31)

update instance weights (eq. 4.32)

normalize instance weights

\( M \leftarrow \) all \( h_i \) and \( \alpha_i \)

return \( M \)

**Algorithm 4.21** Boosting classification

**Input:** an AdaBoost classifier \( M \) and an unlabeled instance \( I = (v_1, v_2, ..., v_n) \)

**Output:** the class hypothesis for \( I \) ([−1, +1])

\( F \leftarrow 0 \)

\( S \leftarrow 0 \)

for each classifier \( i \) in the ensemble do

\( F \leftarrow F + \alpha_i h_i(I) \)

\( S \leftarrow S + \alpha_i \)

return \( \frac{F}{S} \)

The output value will be used in the AdaBoost extension, AdaBoost.M1, to make the
final prediction.
AdaBoost is used in many real world applications for its exceptional accuracy in many data sets [24]. The maximum amount of iterations (or classifiers) has to be adjusted appropriately though, too many classifiers can result in an overfit model with poor performance. Too much noise in the data set will also impact negatively the training by overfitting too.

### 4.5.2.2 AdaBoost.M1

AdaBoost.M1 is simply an extension of AdaBoost to work for data sets with more than two classes. This extension is not complex: we run the regular AdaBoost training for each class using a binary transform of the training set. At classification, we calculate the prediction for each class in regular AdaBoost classification, transform each prediction range from $[-1, +1]$ to $[0, 1]$, normalize them and predict the class with highest probability.

**Algorithm 4.22 AdaBoost.M1 classifier training**

**Input:** a training data set $T$

**Output:** an AdaBoost.M1 classifier

for each class $c$ do

construct binary data set $T_c$

train AdaBoost classifier with $T_c$

$M \leftarrow$ all AdaBoost classifiers

return $M$

**Algorithm 4.23 AdaBoost.M1 classification**

**Input:** an AdaBoost.M1 classifier $M$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

**Output:** the class prediction for $I$

$F \leftarrow 0$

$S \leftarrow 0$

for each class $c$ do

calculate AdaBoost prediction $p \in [-1, +1]$ for class $c$

calculate class $c$ probability as $\frac{p + 1}{2} \in [0, 1]$

normalize probabilities

return class with highest probability

### 4.5.3 Stacking

Both ensemble methods introduced at this point rely on the same principle. They build different classifiers and then combine them in a particular manner. *Stacking* [25] tries to
be more general than that (it’s also known as \textit{stacked generalization}): the output of its base classifiers forms the basis of a new metadata set, and it trains another metaclassifier as a prediction combiner. This way we can learn to blend the basic outputs in meaningful ways, without being constrained to a particular procedure, like voting or weighting.

The generalization extends beyond the existence of a combiner. We can potentially add any number of layers to further improve the results. In this case, though, we must be wary of error propagation and keep in mind the simplicity principle.

Also, empirical results suggest that stacking works better when the combiner doesn’t have only the prediction of the base classifier, but the whole probability distribution. This gives the combiner more information to decide the final output.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{stacking_classification}
\caption{An example of stacking classification with three base classifiers in a data set with classes A, B and C. The instance \((v_1, \ldots, v_n)\) is classified by all base classifiers and their probability distributions \((p_{ij}: \text{the probability of class } j \text{ according to classifier } M_i)\) form the metainstance that will be the input to the combiner classifier, which will compute the final prediction, B in this example.}
\end{figure}

Training the combiner is not trivial. We can’t just train it with the predictions of the base classifiers, since we used that same data to train them and using them again would undoubtedly overfit our model and violate the test/training isolation principle. To avoid this we have to use a procedure similar to cross-validation (4.7.5.3) detailed in figure 4.25.

\begin{algorithm}
\caption{Stacking classifier training}
\textbf{Input:} a training data set \(T\)
\textbf{Output:} a stacking classifier
\begin{algorithmic}
\For{each base classifier \(M_i\)}
\State train \(M_i\) classifier
\EndFor
\State train combiner (see figure 4.25)
\State \(M \leftarrow\) base classifiers and combiner
\State \textbf{return } \(M\)
\end{algorithmic}
\end{algorithm}
Figure 4.25: To train the combiner model we must be sure that the data we input in the base classifiers is not used in the training of said classifiers. By splitting the training set in $k$ folds (3 in the figure, though 10 is the usual number in practice) and following this scheme we ensure that the metadata set $D'$ is not biased.

Algorithm 4.25 Stacking classification

Input: a stacking classifier $M$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

Output: the class prediction for $I$

for each base classifier $M_i$, do

- calculate class probability distribution $pdf_i$ with $M_i$

- predict final class with the combiner with all $pdf_i$ as its input

return predicted class

In theory the combiner should learn how to combine the outputs by itself, but in many cases it might not be enough. In many cases its final accuracy is equal to the best base classifier, just like diversity by selection in antenna arrays. It might be necessary to add some metafeatures to give the combiner more context on the instance at hand. These additional metafeatures depend heavily on the data set and its domain, so they must be specially designed for each set.

For example, the $1$ million dollar prize Netflix machine learning competition consisted in predicting the ratings (1 to 5 stars) a movie would get from a user in a particular date. It was finally won by “BellKor’s Pragmatic Chaos”\cite{26} algorithm which uses stacking: two contestants decided to join forces (and their algorithms) to win the first
prize. The second algorithm in the list, tied in accuracy with the first, was called “Feature-Weighted Linear Stacking” [27]. It is another stacking model using 119 base models using 24 metafeatures such as:

- The log of the number of times the movie has been rated.
- The log of the sum of the positive correlations of movies the user has already rated with the movie to be predicted.
- The standard deviation of the date-specific user means from a model which has separate user means (a.k.a biases) for each date.
- Among pairs of users who rated the movie, the average overlap in the sets of movies the two users rated, where overlap is defined as the percentage of movies in the smaller of the two sets which are also in the larger of the two sets.

4.5.4 Original attempts

Besides the classifiers seen so far, along this project’s development my supervisor and I tried to design and implement other algorithms to see if we could gain more insight into the inner working of classifiers. Many of our ideas happened to be existing classifiers, which gave us confidence that we were following the right path. The following three algorithms were thought by ourselves and, as far as we know, they didn’t appear in any research paper we could find.

4.5.4.1 Class stacking

Class stacking is a variant of stacking where there are as many base classifiers as classes in the set. Each classifier specializes in one class, training on a binary data set and trying to predict whether the instance belongs to that class or it doesn’t. Our intuition is that by splitting the problem in more manageable tasks the classification will become easier and the accuracy will improve. This algorithm is also the base upon the rest of our proposed algorithms are founded.
Algorithm 4.26 Class stacking classifier training

**Input:** a training data set $T$

**Output:** a class stacking classifier

for each class $c$
do
    construct binary data set $T_c$
    train $M_c$ classifier on $T_c$
train combiner (see figure 4.25)
$M \leftarrow$ base classifiers and combiner

return $M$

Algorithm 4.27 Class stacking classification

**Input:** a class stacking classifier $M$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

**Output:** the class prediction for $I$

for each base classifier $M_i$
do
    calculate whether the instance belongs to that class or not with $M_i$
predict final class with the combiner

return predicted class

4.5.4.2 Soft class stacking

This is a more specialized variant of the previous classifier; instead of the base classifiers returning whether the instance belongs to that class or not it returns the probability of belonging. The rest of the algorithm is exactly the same. With this little modification we expect to improve the results of the regular class stacking.

4.5.4.3 Reinforced class ensemble 1

With this other variant of class stacking we try to find out the effect of training each base classifier with a different set. Instead of training each base classifier with the full set we only use a random subset of two thirds the original size.

4.5.4.4 Reinforced class ensemble 2

This class stacking variant tries to apply the idea of boosting (reinforce our learning by focusing on the harder instances) in a different way.

For each classifier we split the training set randomly in training and testing. We build each model, we test it with the second set and we consider the misclassified instances.
Each of these instances are inserted into the training set of the instance’s class classifier, and the surplus created by this insertion will be moved to the test set to maintain the same sizes between the two. After this instance redistribution we start a new iteration, hoping to reduce the classification error.

The classification algorithm is identical to class stacking.

**Algorithm 4.28 Reinforced class ensemble 2 classifier training**

**Input:** a training data set $T$

**Output:** a reinforced class ensemble 2 classifier

repeat

$M \leftarrow \emptyset$

for each class $c$ do

construct binary data set $T_c$

split $T_c$ in training set $T_c^e$ and test set $T_c^t$

train $M_c$ classifier on $T_c^e$

test $M_c$ classifier with $T_c^t$

$M \leftarrow M \cup$ misclassified instances

for each instance $i$ in $M$ do

$c \leftarrow$ class of $i$

move $i$ into $T_c^e$

move random instance from $T_c^e$ into $T_c^t$

until maximum iterations have been reached or classification error is low enough

train combiner (see figure 4.25)

$M \leftarrow$ base classifiers and combiner

return $M$

---

### 4.5.4.5 Iterative class ensemble

Finally, the last variant on class stacking follows yet another strategy, this time without the need for a combiner. Its training is identical to class stacking, but classification is more involved. The base classifiers determine if the instance belongs to their class, and then:

- If only one class returns a positive, it returns that class.
- If no class returns a positive, it returns the most probable class from the previous iteration.
• If the number of positives is the same as the previous iterations, it returns the most probable class from the previous iteration.

• If more than one class returns a positive, it retrained the classifiers from the classes that returned positive with instances from those classes only and it starts a new iteration.

![Figure 4.26: Iterative class ensemble classification in a three class data set, in the best case. In the first iteration we discard class B, so we remove all class B instances from the training set and retrain only with instances with class A or C. On the second iteration we discard A so we predict class C.](image)

The expectation is that by removing the training instances that we are certain are not relevant in the current classification job the accuracy will be better. A worrying aspect of this algorithm is the need to retrain in each classification, which will probably incur in long testing times.

**Algorithm 4.29 Iterative class ensemble classification**

**Input:** an iterative class ensemble classifier $M$ and an unlabeled instance $I = (v_1, v_2, ..., v_n)$

**Output:** the class prediction for $I$

repeat
    for each base classifier $M_i$ do
        calculate whether the instance belongs to that class or not with $M_i$
    until a decision is made (see previous list)

return predicted class

### 4.6 Regressors

Even if the focus of this chapter is on classification, sometimes we found necessary in Krimtrack to predict numerical values (like expected number of incidents on a zone). That is why we included two regressors to complement the system.
4.6.1 Linear regression

Linear regression is a well-known estimation method in which the data is fit by a linear function, like we mentioned in the logistic regression classifier (equation 4.25). The objective is to minimize the squared error of the estimator $\hat{y}$.

$$\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$  \hspace{1cm} (4.33)

$$e^2 = \frac{1}{2} \sum_{i=1}^{m} (\hat{y} - y)^2$$  \hspace{1cm} (4.34)

$$\frac{\partial e^2}{\partial \beta_j} = \sum_{i=1}^{m} (\hat{y} - y) x_j = 0$$  \hspace{1cm} (4.35)

Unlike logistic regression, the equation system we have in this case is linear and easily solvable by the Gauss method, for instance. There is one more detail: in order to improve the model’s accountability, during both training and regression we normalize the training inputs, mapping each feature’s minimum value to 0 and each maximum value to 1. Thus we can compare coefficients and have a better intuition on the weight each feature has on the final estimation. Otherwise features with large values would have small coefficients to balance one another; similar values guarantee that differences between coefficients are significant of their importance.

**Algorithm 4.30** Linear regressor training

**Input:** a training data set T  
**Output:** a linear regressor  
- calculate equation system (equation 4.34)  
- find solution $(\beta_0, ..., \beta_n)$ to system using Gauss  
- store solutions $\beta_i$ in model $M$  

`return M`

**Algorithm 4.31** Linear regression estimation

**Input:** a linear regressor M and an unlabeled instance $I = (v_1, v_2, ..., v_n)$  
**Output:** the estimated value for $I$  
- normalize all $v_i$  
- calculate estimated value (equation 4.33)  

`return estimation`
4.6.2 k-NN regression

Training a k-NN model for regression is identical to the classification algorithm in 4.11: filtering instances and storing them. Estimation is similar to classification too: we find the \( k \) closest neighbours but we estimate the output as a weighted sum of the neighbours labels. Each neighbour’s weight is the normalized inverse of its distance to the new instance.

**Algorithm 4.32 k-NN estimation**

**Input:** a k-NN regressor, a parameter \( k \) and an unlabeled instance \( I = (v_1, v_2, ..., v_n) \)

**Output:** the estimated value for \( I \)

```
for each instance \( I_{stored} \) in the model’s stored instances do
    calculate distance between \( I \) and \( I_{stored} \)

\( C \leftarrow k \) stored instances closest to \( I \)

return class as weighted sum of \( C \)
```

4.7 Testing methodology

The machine learning procedure doesn’t stop at training; we have to be able to evaluate the model before deploying it in the real world. There are three main tools we developed that we need to consider: error measures, cost matrices and model testing schemes.

4.7.1 Error measurement

For a given model, error measures evaluate how well it predicts a particular test set. There are several figures available, judging different kinds of error and goodness of fit.

4.7.1.1 Accuracy / MAE

The simplest error measurement is accuracy percentage in classification or mean absolute error (MAE) in regression. This single number provides a condensed figure to compare models and it allows us to have an intuitive understanding of its performance. The problem is that it lacks more detailed information about the error distribution that other measures do provide.
4.7.1.2 Baseline accuracy

Of course, we need something to compare this accuracy measure with. For classification cases this number is the baseline accuracy: the percentage of the most popular class. Classifying always as that class (that kind of classifier is known as ZeroR) is a trivial way of establishing a minimum threshold that any good classifier should improve if they aim to provide any real information.

Take, for instance, a balanced coin. If we had a classification procedure with 50% of accuracy at predicting head or tails it wouldn’t be very impressive; both outcomes have a 50% chance, so no further information is gained. In the other hand, a 50% accuracy on the outcome of a 10-sided die roll can be considered as a significant improvement, since the maximum chance of guessing correctly (i.e. the baseline accuracy) is only 1 in 10: 10%.

4.7.1.3 Confusion matrix

In a classification model, a confusion matrix keeps the count of each kind of error (figure 4.27): how many instances from a given class (rows) have been classified in another class (columns). It is a square matrix with rank equal to the number of classes, and it can be very helpful to determine whether the accuracy error is distributed equally among classes or is more prevalent in a few ones. Using it we can have an intuitive understanding of the relative difficulty of each class, which can give us insight into the problem’s nature.

![Figure 4.27: An example of confusion matrix for the “fourier” data set. Notice how classes 6 and 9 are confused often, as we anticipated.](Image)
4.7.1.4 Precision / recall

Focusing on a particular class, we can summarize a confusion matrix in four values: those instances belonging to the class we classified as such (true positives, TP), those that don’t belong to the class yet we misclassified as positive (false positive, FP, also called type I error), those belonging to the class and we misclassified (false negative, FN, also called type II error) and finally those that don’t belong and we classified as such (true negative, TN).

With these four values we can define precision $P$ and recall $R$:

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$

Precision is a measure of correctness (i.e. excluding everything that should not be) while recall is a measure of thoroughness (i.e. including everything that should be). In unbalanced data sets it’s usually a challenge to have a good value on both measures at the same time.

For example, take figure 4.28. Precision would be calculated as 17 true positives among 24 positive instances: 70.8%. Recall would equal 17 true positives among 21 true instances: 80.9%. Finally, the accuracy would be 30 true positives or true negatives among 41 instances: 73.1%.
4.7.2 Cost matrix

On the previous three sections we made an implicit assumption: that all errors are equally relevant. This might not be the case; for instance, in the “credit” data set the author suggests that mistaking a bad client for a good one is worse than the other way around. Hence, we need a way to specify each error relevance in the accuracy calculation.

The cost matrix has the same structure than the confusion matrix and it stores each classification error (or success) relative weight. Therefore, the accuracy becomes:

\[ W\epsilon = \sum_{\text{misclassified instances } i, j} w_{ij} \]

\[ W = \sum_{\text{all instances } i, j} w_{ij} \]

\[ \text{accuracy} = \frac{W\epsilon}{W}, \]

where instances \( i, j \) are instances of class \( i \) classified as class \( j \), with weight \( w_{ij} \).

\[
\begin{array}{cc}
\text{classified as} & \text{good} & \text{bad} \\
\text{is} & \text{good} & 1 & 1 \\
\text{bad} & 5 & 1 \\
\end{array}
\]

Figure 4.29: The suggested cost matrix for the “credit” data set by its author[7].

4.7.3 Receiver operating characteristic (ROC)

The receiver operating characteristic (ROC) is a graphical plot used in detection theory that, in our case, represents the variability of true positives versus false positives when the classification threshold changes.

ROC analysis works by looking at the classifier output not as a discrete class, but as a probability of belonging to a certain class. In that case we would usually return that class as the prediction if that probability was greater than 0.5 (and hence the probability of not belonging was less than 0.5), but it doesn’t necessarily have to be that way. Depending on the class, we could choose to return a positive even if that number doesn’t reach 0.5, maybe because any hint that points to that class is enough and we want to minimize false negatives. Or viceversa, we could require a probability
higher than one half because we have to be sure before returning that value, minimizing false positives.

The ROC curve sweeps over probability thresholds, from 0 to 1, and for each point a different classification is made and a confusion matrix created. With this confusion matrix we calculate, as explained in the precision/recall section, the values \( TP, FP, FN, TN \). Then, we normalize the values to obtain the true positive rate \( TPR \) (also called sensitivity) and false positive rate \( FPR \) (equivalent to 1 - specificity):

\[
TPR = \frac{TP}{TP + FN}
\]
\[
FPR = \frac{FP}{FP + TN}
\]

Drawing all points, we get a ROC curve, like in figure 4.30.

We can identify different areas of interest in the ROC graph.

- The perfect classification would be the point at (0,1): maximum true positives and no false positives. A model’s curve should strive to be as close to this point as possible.

- The diagonal is called the line of no-discrimination. Points on this line provide no information and are no better than random guesses.

- The area above the diagonal contains the good models (better than random chance), where the true positive rate exceeds the false positive rate. The furthest from the diagonal, the better the model is.

- Conversely, the area below the diagonal has the bad classification results (worse than random chance). If a curve is consistently bad and is fully contained in this area we can invert its prediction to obtain a good classifier.
• The distance between the curve and the no-discrimination line is known as Youden’s $J$ statistic. Trying to maximize this statistic for the curve can give us the optimum cutoff point for predicting that class (the red line in figure 4.30).

• Finally, we can compute the area under the curve (AUC) as another estimator. Even though it is used as the standard method to assess predictive models, research has shown it can be a misleading measure[28].

4.7.4 Bias / variance

When evaluating a classifier we have to be aware that the method of testing (explained below) is just a mere approximation for the final model’s performance. Depending on how we test we might introduce bias and/or variance in the accuracy estimation.

As we will see, most test schemes use one or more proxy models built only for evaluation purposes. If these models are built with a set of instances with a substantially different statistical distribution the evaluation will introduce bias. On the other hand, if the testing data set is small we will have more variability upon the accuracy results, meaning an increased variance. Both extremes are bad since they both give us a wrong impression of our model’s performance, so a balance must be found between the two.

4.7.5 Model testing

4.7.5.1 On training validation

One first scheme to validate the model is to use the full data set to train the model and then test it with the same set. This scheme returns a over-optimistic estimation so it is never used unless there is a good reason to do so (like in boosting). The reason for this optimism is that we are not really testing its generalizing properties, just its memorizing capabilities. For example, k-NN and unpruned trees among others are called “perfect memorizers” because they would always score 100% with this scheme!

That is why it is so important to separate training and testing, and never use information from the testing set in the training algorithm. The rest of the schemes follow this training/test independence principle.

4.7.5.2 Holdout

Holdout is the simplest method to provide separation between training and test. We split the data into two exclusive subsets, use one for training and the other to test.
Empirical results (that we will test in the next chapter) show that a good rule of thumb for splitting a holdout set is to use two thirds for training and one third for testing[29].

The problem with the holdout scheme is that we only use part of the data to test which might not be representative of the larger set. This introduces higher variance, unless the test set is very large, in which case the bias might become a problem instead. All this becomes less important if we have available a large amount of instances, since even a small part can still be representative of the whole.

Lastly, in order to preserve each subset statistical distribution we make stratified splits, that is, splits that preserve the same proportion of classes in each fold.

### 4.7.5.3 Cross-validation

Cross-validation tries to solve the variance problem of holdout following the scheme at figure 4.33. By splitting with stratification the data into $k$ folds and eventually using all of them to train and test (preserving the training / test separation) we produce a much more robust result, reducing variance at the expense of a higher computational cost (since we have to train $k$ proxy models).

We still have to be careful with bias and variance though. If we split by too few folds, each proxy model will be trained by too few instances, so the results might not be
generalizable to the full set and bias may appear. On the other hand, too many folds will result in a poor test set that will give high variance estimations. Again, as a rule of thumb 10 folds is a popular option, but we will also test it in chapter 5.

### 4.7.5.4 Bootstrap validation

The last alternative’s goal is to obtain a low variance estimation without the computational cost of cross-validation. To accomplish this we create a training bootstrap sample from the original data set and build the test set with the remaining instances. Both sets are tested in the model and the final accuracy estimation is a weighted sum like indicates figure 4.34[30].

A bootstrap sample is a statistical technique that consists in sampling uniformly from the data (with replacement) to get a sample of equal size to the original. Since it is sampled with replacement there is a significant amount of duplicate instances (the sample will contain an expected 63.2% of distinct instances).

### 4.8 Hyperparameter selection

Training algorithms are the methods we use to obtain a model’s parameters (like neural network weights, regression coefficients or naive bayes probabilities) to be able to use them for classification purposes. But their effectiveness depends on another set of
different parameters, called training parameters or hyperparameters, that determine the generation of the model itself; i.e. the training. To maximize the model’s quality we must select these hyperparameters carefully.

To perform hyperparameter selection (also called model selection) we need a third data subset besides training and testing known as validation set. The application of this procedure splits the data into even smaller subsets, so for small data sets it might not be worth it to run a model selection procedure if the training is left considerably impaired. There are other, less damaging, techniques like using cross-validation to choose these hyperparameters, but the execution time would be notably increased and our time in this project’s development is limited.

There are many ways to do model selection, many of them using heuristics to find a good combination of hyperparameters in a reasonable time, but we will use the simplest one as a first attempt: grid search.

### 4.8.1 Grid search

*Grid search* consists in choosing a finite, discrete subset of all hyperparameters, training a model for each combination and finally testing them with the validation set. We finally return the hyperparameters that produced the most accurate model.

This selection has an exponential growth with the number of training parameters, so it’s not feasible to do with more than a few of them. That limits our possibilities but it is enough for our purposes at the moment. If our needs increase in the future we’ll be forced to implement another model selection algorithm such as *random search*, for example.

In the example in figure 4.35, for example, the optimal training parameters seem to be $\alpha=0.35$ and $\eta=0.2$, with an accuracy of 82.2% on the validation set.
Figure 4.35: Grid search for a neural network training.

<table>
<thead>
<tr>
<th>Momentum (α)</th>
<th>Learning factor (η)</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>77.7</td>
<td>78.9</td>
<td>79.4</td>
<td>79.5</td>
<td>79.1</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>77.5</td>
<td>78.6</td>
<td>80.2</td>
<td>79.9</td>
<td>79.5</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>78.5</td>
<td>78.7</td>
<td>80.7</td>
<td>80.2</td>
<td>79.8</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>79.3</td>
<td>80.1</td>
<td>81.3</td>
<td>81.3</td>
<td>80.3</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>80.2</td>
<td>79.9</td>
<td><strong>82.2</strong></td>
<td>81.7</td>
<td>80.1</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>79.1</td>
<td>78.9</td>
<td>82</td>
<td>81.6</td>
<td>79.9</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Results

This chapter will review which kind of tests we put Krimtrack through, which were their objectives and the results we got from them. We will limit the study to the two most critical areas: the communication server and the machine learning module. Nevertheless we will first outline how we defined the testing stages for the global system.

5.1 Use case validation

The mobile application and the coordination web application were also thoroughly tested for bugs and other necessary adjustments. To do so, we established several stages of testing necessary to validate Krimtrack’s function as a whole, with these different goals for each stage:

• **First stage:** preliminary testing
  
  – **Who:** the developer (me)
  
  – **When:** from the beginning of the development
  
  – **Objectives:**
    
    * Deploy Krimtrack on test server environment
    
    * Identify and remove critical bugs
    
    * First use case evaluation and adjustment

• **Second stage:** internal field testing
Chapter 5. Results

- **Who:** my colleagues and myself
- **When:** August 2014
- **Objectives:**
  * Deploy NFC tags on test field
  * Identify and remove bugs
  * Load test with client simulator
  * Receive feedback from colleagues

**Third stage:** real field testing

- **Who:** real workers / potential clients
- **When:** 2015
- **Objectives:**
  * Deploy Krimtrack on production server environment
  * Load test on production server with client simulator
  * Identify and remove any remaining bugs
  * Adjust use cases according to specific business needs
  * Receive client feedback for new functionalities
  * Receive client’s suggestions for improvement

5.2 Communication server

The most important module of Krimtrack is the communication server, the backbone of the whole system. In particular we must be able to guarantee maximum stability in any situation. That is why the highest priority is to perform a load test to confirm that the server can withstand a large amount of persistent websocket connections simultaneously.
5.2.1 Server specifications

The test server we used is a home computer with the following specifications:

- Intel Core i7-2600 3.4 GHz
- Memory of 8 GB
- ADSL 100 Mbps
- Debian 7.6 Linux
- Java application server Tomcat 7

For production we plan to hire a cloud hosting with layershift using Jelastic. Jelastic is a technology that provides a Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) paradigm for server management\(^1\). That means that the cost is based only on the hourly usage of server resources such as CPU, memory, disk space or network bandwidth, and they all scale automatically to support virtually unlimited usage.

The reason we chose layershift in particular is that they are located in the United Kingdom. We need our server to be located in European territory to comply with the European Data Protection Directive. There are other alternatives like the so-called “safe-harbor” frameworks\(^2\): American hosting companies that are certified to work under compliance of said European directive.

5.2.2 Load testing

Load testing is important for all kinds of server analysis, but in this case there is an additional reason to do it. Each user will use websockets, which implies a persistent open connection and, consequently, a socket. Operating systems have an internal limit on the number of open sockets, which can impose us an insurmountable limit on the amount of concurrent users.

On the other hand, Krimtrack is not meant to be used by a massive user base. A single coordinator has only so much power to supervise a large amount of people before he gets overwhelmed. To test a typical situation we have developed a simple “client simulator”, an auxiliary web application that implements the communication protocol used by the terminals and simulates the connection and random walk of \(N\) users.

\(^1\)http://jelastic.com/features/scalability/
\(^2\)http://www.export.gov/SafeHarbor/
We run into a problem with this simulation, though. Each connection uses a websocket, and browsers have a low built-in maximum limit (given our needs) on the number of open websockets in a browsing session. Additionally, even if we open different browsers some connections are dropped by the clients before we reach 400 connections. This hints at a problem within the simulation; since it’s a relatively new technology, it doesn’t seem like current web browsers are optimized to support a large volume of open websockets yet.

In any case, so far the test server has been able to effortlessly handle hundreds of connections in a home network. The production server, with more available resources, will undoubtedly offer a far better performance. The only problem lies within the simulation; a future challenge would consist in a way to test lots of connections in a large distributed scale to simulate more accurately a real scenario.

5.3 Machine learning results

To evaluate the different concepts introduced in the previous chapter, we designed several experiments to confirm and learn more about the data sets, algorithms, filters and methodologies used.
5.3.1 Data sets

Our first experiment was a full exhaustive sweep of all learning algorithms applied to all data sets. Since time was an important issue for this experiment, we limited it to 10-iteration holdout tests with no model selection (2/3 for training and 1/3 for testing). All necessary discretization will be performed in 4 equal-sized intervals and all data sets will be evaluated with a default cost matrix (i.e. all errors will be considered equal). Finally, for this first section we will not use any filter.

We present the results of the five base classifiers plus the eight ensemble learners implemented and the two regressors, with the following details:

1. Naive bayes
2. C4.5 decision tree with a 95% prepruning threshold
3. Neural network with one hidden layer with a number of neurons equal to two thirds of the number of features, learning rate of 0.3, momentum of 0.2 and 500 epochs
4. k-NN classifier with k=3
5. Logistic regression
6. Random C4.5 Forest with 20 trees with the same parameters as above
7. AdaBoost.M1 using decision stumps, with a maximum of 20 iterations
8. Stacking with a Naive bayes, a random forest, a neural network and a 3-NN as base classifiers and logistic regression as a combiner, all of them with the same parameters as above
9. Class stacking using neural networks
10. Soft class stacking using neural networks
11. Iterative class ensemble using neural networks
12. Reinforced class ensemble 1 using neural networks
13. Reinforced class ensemble 2 using neural networks
14. Linear regression
15. k-NN regressor with k=3
The accuracy bar graphs show the algorithm average accuracy percentage with its 95% confidence interval (2 standard deviations, assuming a normal distribution). In the case of training time bar graphs, given the disparity of values (from tenths of a second to several hours), we plot a logarithmic scale of log(\text{seconds}) since we are more interested in the order of magnitude than the exact value anyway. For both graphs we highlight in green the “best” algorithm (the one with the best average accuracy and lowest mean training time).

Also, we plot a horizontal line with the baseline accuracy: the constant model returning the most probable class in the data set. It serves as the minimum threshold a model needs to score in order to be useful.

5.3.1.1 Audiology

![Audiology accuracy graph](image)

Figure 5.2: Accuracy results for the audiology data set.

In this first set we see good results for most of the classifiers in comparison to the baseline accuracy. The best classifier is AdaBoost.M1 closely followed by a C4.5 decision tree.

Comparing this set to the other five we use for classification, several differences stand out:

- It’s the only set where a random forest doesn’t improve the accuracy of a single decision tree. With that said, the forest still manages to reduce the confidence interval.
Neural networks usually have very good performance in all sets (second highest on most of them), yet it has quite a mediocre result here in comparison to the rest. These points indicate that there is something very strange about this set’s statistical distribution. By trying to diversify it (with a random forest, for example) we end up spoiling the model even further and the models behave erratically compared to the other sets.

![Audiology training time](image)

**Figure 5.3:** Training time results for the audiology data set.

The training times make more sense. The large number of features makes most of the algorithms to last more than a second, even though there are few instances to train. The only algorithm whose training time is mostly influenced by the number of instances, the 3-NN, is the fastest this time with slightly more than 10ms.

On the other hand, we have the soft class stacking with an execution time of almost 4 days. It is clear that would become a major liability even if it showed worthwhile global accuracy results (which it doesn’t).

### 5.3.1.2 Column

The highest accuracy in the “column” data set is found in the logistic regression classifier, narrowly beating neural networks (by less than 0.1%). Even then, neural network’s lowest variance probably gives them an edge against logistic regression.
Many ensemble classifiers show a large variance. My interpretation is that we lack enough instances to construct a composite model providing stable results, which is a worrying aspect if we intend to use them in a real application.

Surprisingly, good algorithms in other sets fail to offer good results on this one. Random forests (barely outperforming single decision trees) or AdaBoost.M1 struggle to reach 70% accuracy. Even the worst models (reinforced class ensembles 1 and 2) surpass the baseline accuracy.

All base classifiers are very fast: less than a second or even less than a tenth of a second in the case of the 3-NN, the fastest algorithm. The reason for the overall speed is probably the low amount of instances and attributes. The few exceptions are the stacking and soft class stacking, still taking a few minutes each.

We can see how modifying the class stacking to soft class stacking pays off. With an increased execution time we also get around a 15% of accuracy improvement.

### 5.3.1.3 Breast cancer

This time the 3-NN algorithm is the most accurate classifier, followed once again by neural networks. In third place we have naive bayes, outranking most of the others that don’t even reach the baseline accuracy. This observation suggests, unsurprisingly, that cancer is actually quite difficult to predict after all.
The inability of some models to reach the baseline accuracy is probably due to the presence of irrelevant features in the set. This intuition is confirmed when we retrain the C4.5 decision tree with CFS filtering enabled: from 64.58% we go to 69.69%, roughly the baseline threshold of 70.28%.
That is also the reason why our algorithm attempts look comparable to the rest. It’s not that they are as good as the established algorithms, it’s just that all of them have similarly bad performance.

We have to note that in cases like these where the model has the same accuracy as the baseline model (returning always the most probable class) doesn’t mean they are equally useful. That is precisely the reason why a cost matrix is so important to adjust in practice; other less probable classes might be more critical to predict, and a baseline model would fail in all of them.

![Breast cancer training time](image)

**Figure 5.7:** Training time results for the breast cancer data set.

As expected, the small size of the set makes the training fast (less than a second for all base classifiers). Ensemble classifiers are not as fast, but they are still significantly faster than they are in other data sets.

### Fourier

In the “fourier” set the winner is the 3-NN followed by stacking, which uses 3-NN as a base classifier anyway. The others don’t reach 80% and the most likely explanation is that, as we saw in figure 4.8, there is a lot of overlap between classes that entropy or other “global” measures has difficulty separating. The k-NN classifier uses a local heuristic to better discriminate these difficult cases.
Figure 5.8: Accuracy results for the Fourier data set.

The confidence intervals are significantly smaller than the previous sets, fact that can be explained with the larger data set size of 2000 instances. This size and the perfect class balance (200 instances for each class) explains also how far above the baseline accuracy we managed to reach in comparison to other sets like breast cancer or credit.

Figure 5.9: Training time results for the Fourier data set.
Naive bayes is the fastest algorithm this time training in slightly more than a tenth of a second, which along with his good accuracy makes it a very good overall classifier for this set. Nevertheless, the 3-NN runs for roughly a second, which is not (in most cases) a significant difference in time.

This time the ensemble learners are slower than other sets. Again, as a relatively large set this makes sense.

5.3.1.5 Credit

By the results we see that this set seems more difficult than the others. Even the best algorithms, AdaBoost.M1 and Naive bayes, only raise 3% over the baseline accuracy of 70%. Some of them don’t even reach it, probably due to the presence of irrelevant features just like in the breast cancer set. Again, retraining the C4.5 decision tree with feature filtering we go from 68.38% to 70.99%, barely rising above the baseline accuracy.

Once again Naive bayes is the fastest learner while providing the best results (tied with AdaBoost.M1). As far as training time is concerned, the results are pretty analogous to the fourier data set.
5.3.1.6 Nursery

Two results from this graph are easily noticed: the accuracy results are very high (often surpassing 90%) and confidence intervals are very small due to the large amount of
instances (12960). The only exceptions to the last statement are the last four ensemble algorithms, which we would conclude are not focused enough to provide stable and repeatable performances.

This set brings forth another interesting fact: we can see stacking with an almost perfect accuracy (99.65%) and a confidence interval that assures us that the difference with the second is statistically significant. As research papers about stacking warn, this algorithm tends to improve the result of its base classifiers when there is a large amount of data to learn from.

The size of the set is also the responsible for the notably increased training times, even if Naive bayes remains very fast at roughly one second. Most other classifiers, though, have increased in one or two orders of magnitude their training duration.

In particular we could observe how the best classifier this time, stacking, needs a long time to conclude: more than twelve hours. Given the set’s nature as a nursery admission rules, it is unlikely that this long time is a problem but in other more immediate applications we may need to discard it in favor of other faster, although worse, classifiers.

5.3.2 Classifier algorithms

In this section we will present the same information from the previous experiment grouped by algorithm, with additional information to contextualize and analyse each
model’s performance.

The plots are similar to the ones in the previous section, with six bars this time (one for each data set). In the accuracy plots we added a red point as the data set average accuracy among all classifiers, to compare each model in relation to the rest.

5.3.2.1 Naive bayes

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>57.65%</td>
<td>0:00.087</td>
</tr>
<tr>
<td>Max</td>
<td>90.03%</td>
<td>0:01.706</td>
</tr>
<tr>
<td>Mean</td>
<td>73.12%</td>
<td>0:00.628</td>
</tr>
</tbody>
</table>

Table 5.1: Naive bayes results

We already concluded that naive bayes is not usually the best classifier, but it is good enough in most cases. In four of the six data sets it improves the overall mean accuracy; only in the column and audiology sets falls below the average. This might be a hint that the independence approximation of the naive bayes is not a good hypothesis on these sets.

![Naive bayes accuracy](image1)

![Naive bayes training time](image2)

**Figure 5.14**: Results for the naive bayes model.

Furthermore, we have seen that naive bayes is always a very fast algorithm among all six classification data sets. Audiology is not as fast though, due to the large number of classes and attributes, which need a lot of conditional probability estimation.

5.3.2.2 C4.5 decision tree

Standard decision trees have shown subpar performance on these experiments. Accuracy drops below the mean in four data sets and is not particularly fast in comparison to
Chapter 5. Results

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>68.38%</td>
<td>0:00.134</td>
</tr>
<tr>
<td>Max</td>
<td>96.46%</td>
<td>1:52.117</td>
</tr>
<tr>
<td>Mean</td>
<td>75.05%</td>
<td>0:29.695</td>
</tr>
</tbody>
</table>

Table 5.2: C4.5 decision tree results

other base classifiers (several seconds). That is precisely the reason why decision trees are rarely used on their own; random forests are trained instead.

As we mentioned in the previous chapter there is one situation where a decision tree might still be preferable despite these results: accountability. Depending on how we use the model, it might be desirable to have an explanation about the decision. Other classifiers lack such a quality, and the risk involved in deciding “blindly” might not be worth the increased accuracy.

5.3.2.3 Neural network

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>65.53%</td>
<td>0:00.438</td>
</tr>
<tr>
<td>Max</td>
<td>97.16%</td>
<td>0:34.414</td>
</tr>
<tr>
<td>Mean</td>
<td>78.84%</td>
<td>0:09.426</td>
</tr>
</tbody>
</table>

Table 5.3: Neural network results

It’s not surprising that neural networks have good results, they are arguably the most popular algorithm used in machine learning. They offer good results consistently, a good property for a classifier.
We can also observe how the confidence intervals are smaller than other algorithms, which indicates stability. Different executions of the same neural network using different random seeds provides a very similar model.

An unexpected result is the training time. The execution takes roughly the same time as decision trees, even though neural networks are supposed to be costly to train. The most likely explanation is that a long time in neural networks is spent optimizing its parameters, a procedure we skipped in this experiment.

### 5.3.2.4 k-NN

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>66.00%</td>
<td>0:00.022</td>
</tr>
<tr>
<td>Max</td>
<td>87.22%</td>
<td>0:23.688</td>
</tr>
<tr>
<td>Mean</td>
<td>76.71%</td>
<td>0:05.054</td>
</tr>
</tbody>
</table>

Table 5.4: k-NN results

On the sets chosen, k-NN reports average results. The fourier data set is an exception, surpassing the rest of models by more than 6% (not including the stacking classifier, which uses a k-NN internally). We can confirm then that the fourier set has a very relevant local structure that we need to take in consideration to maximize accuracy.

Even if it’s not as fast as a naive bayes classifier, it is still a very fast algorithm. The audiology set is trained faster, though, since the large of number of classes bear no relevance in k-NN, unlike naive bayes.
5.3.2.5 Logistic regression

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>65.18%</td>
<td>0:00.125</td>
</tr>
<tr>
<td>Max</td>
<td>91.86%</td>
<td>0:26.832</td>
</tr>
<tr>
<td>Mean</td>
<td>75.93%</td>
<td>0:08.575</td>
</tr>
</tbody>
</table>

Table 5.5: Logistic regression results

Logistic regression shows slightly above average results, with the exception of the fourier and breast cancer sets. In the case of the column set it is also the highest ranked classifier with 82.21%, while many other algorithms struggle to reach 70%.

During testing we found an interesting case worth mentioning. Unlike regular linear regression, sometimes finding a logistic function that minimizes the error is impossible since it doesn’t exist. Those cases are not a problem, on the contrary[31]: if this happens it means that we can separate perfectly the points and the coefficient found tends to infinite to asymptotically reduce the error to approximate a step function, as we can see.
in figure 5.19. Therefore, the solution is to limit the coefficient to a sufficiently large number.

\[\text{Figure 5.19: One-dimensional logistic functions with increasing coefficients and diminishing error}\]

5.3.2.6 Random C4.5 forest

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>69.29%</td>
<td>00:0841</td>
</tr>
<tr>
<td>Max</td>
<td>96.6%</td>
<td>20:56.469</td>
</tr>
<tr>
<td>Mean</td>
<td>77.25%</td>
<td>05:16.751</td>
</tr>
</tbody>
</table>

\[\text{Table 5.6: Random C4.5 forest results}\]

As expected, random forests improve the results offered by single decision trees (except in the strange case of the audiology set), but they still fail to reach the accuracy mean in the column set. On the other hand, we lose the accountability that trees offer, or at least we reduce it since it is more difficult to trace the decision made over many different trees.

\[\text{Figure 5.20: Results for the random C4.5 forest model.}\]

As usual in ensemble classifiers, the training time is considerably increased since all the different base classifiers (20 trees in this case) have to be trained.
5.3.2.7 AdaBoost.M1

Looking at the accuracy graphs, AdaBoost.M1 results look disappointing. Sometimes it works very well; for example, it ranks the highest in the audiology and credit data sets, which we have seen are specially difficult judging by the results. But except for these, the rest return average results, including a below average accuracy on the column data set. From this fact we understand that boosting works very well on some sets and not as well for others; it lacks consistency.

On the other hand it is a fast algorithm compared to other ensemble classifiers. The tradeoff between accuracy and speed might make it worthwhile, depending on the application.

To see in more detail how AdaBoost works we have performed an additional experiment with more base classifiers on the fourier data set to see how it influences the final error. In figure 5.22 we plotted the average of the classification error of 10 AdaBoost models. We see how the error decreases exponentially when we increase the number of iterations just like the theory predicts. We also note how the error on the training set (used internally as a stopping criterion) is always lower than on the test set.

As we said on the previous chapter, AdaBoost.M1 uses a regular AdaBoost classifier for each class internally. If we dissect each internal AdaBoost classification similarly to what we have just done, we obtain the plot in figure 5.23. Since it is just one realization this time, the lines are more jagged, but it’s still clear that they follow an exponential
trend. Also, each line (that is, each class) converges to a different error value that can be interpreted as the “difficulty” of each digit. Thus we confirm what we said about the data set in the previous chapter, for instance: digits 6 and 9 are the most difficult and digits 0 (as we could intuitively see in figure 4.8) are very easy, reaching perfect classification when using more than 27 classifiers.

5.3.2.8 Stacking

In academic research, they point basically to three cases where stacking is useful:
Chapter 5. *Results*

### Table 5.8: Stacking results

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>67.40%</td>
<td>1:52.796</td>
</tr>
<tr>
<td>Max</td>
<td>99.65%</td>
<td>12h:00:04.227</td>
</tr>
<tr>
<td>Mean</td>
<td>80.15%</td>
<td>3h:25:10:871</td>
</tr>
</tbody>
</table>

1. In many situations stacking works as a selection method. It shows the accuracy of the best base classifier on its own, often slightly lower due to the error introduced by the combiner.

2. As a combiner of the strengths of its classifiers to improve on their individual accuracies. For this, usually a lot of instances (like in the nursery set) have to be available.

3. Like the previous point, but with the addition of metafeatures specific to the data set. As we mentioned in the last chapter, this has excellent results on professional competitions like Netflix[26].

Point 1 is the reason why stacking has the highest mean accuracy of all classifiers. By selecting its best base classifier, stacking never has bad results unless all of them fail. This helps to maintain consistency among different problems.

Point 2 is proved by looking at the nursery data set. Its large size of 12960 instances gave the opportunity to obtain high accuracies from all classifiers, but stacking was the most accurate model with a clear statistically significance. It makes us confident that stacking will be very useful for future large data sets we will encounter.

![Figure 5.24: Results for the stacking model.](image)

Of course there is a downside to this high accuracy, the training time. It needs to train for a long time, 12 hours for the nursery set. As always, if time is not an issue this is not a problem but it disqualifies stacking for its use for fast, on-the-fly machine learning situations.
5.3.2.9 Class stacking

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>48.00%</td>
<td>0:02:925</td>
</tr>
<tr>
<td>Max</td>
<td>93.77%</td>
<td>17:32.461</td>
</tr>
<tr>
<td>Mean</td>
<td>69.05%</td>
<td>5:39.074</td>
</tr>
</tbody>
</table>

Table 5.9: Class stacking results

This is the first of our own implemented ideas. The basic intuition in all of them is that splitting the data set in binary classes makes the training easier by reducing the original problem to a group of easier ones. Given the results on all these classifiers we have to conclude that this assumption might not be valid.

![Figure 5.25: Results for the class stacking model.](image)

There are several reasons that we think explain why they don’t work:

1. Some algorithms not only don’t take advantage out of reducing the number of classes but it is probably harmful for them to do so (figure 5.26).

2. Transformed data sets often become unbalanced. This unbalance can heavily bias the “easier” classification problems. As an example, the fourier set has ten equiprobable classes. By transforming it, it becomes a two-class set (belongs / doesn’t belong) with 10%/90% proportion. When training, it will be biased towards “doesn’t belong” (the simplest model predicting always “doesn’t belong” would have a 90% accuracy!). Hence, many inputs to the combiner will look like figure 5.27 (very low recall as defined in section 4.7.1.4).

3. A combining classifier can have difficulty if more than one class predicts “belongs”. It introduces further errors that will decrease even more the final accuracy measure.
With all these facts and looking at the results, we can’t say the result is satisfactory. Besides, all these classifiers’ training time tends to grow sharply with the number of classes until they become unmanageable.

5.3.2.10 Soft class stacking

From all our attempts at creating a classifier, the soft class stacking model is probably the best one. Its accuracy is greater or equal than the other algorithms, with the surprising exception of the nursery set performing a lot worse than the mean, by a difference of 15%.
Chapter 5. Results

Table 5.10: Soft class stacking results

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>62.60%</td>
<td>8:29.189</td>
</tr>
<tr>
<td>Max</td>
<td>81.43%</td>
<td>3d:19h:18:51.889</td>
</tr>
<tr>
<td>Mean</td>
<td>72.36%</td>
<td>19h:37:14.590</td>
</tr>
</tbody>
</table>

Additionally, it improves the outcome of the regular class stacking, proving that the “soft” paradigm (combining class probabilities instead of discrete class predictions) is an excellent practice.

Figure 5.28: Results for the soft class stacking model.

Nevertheless, it has a big drawback: the largest training time of all algorithms by far with a maximum of almost four days for the audiology set. As usual, this time is further increased by the number of classes so in a real data set with lots of them this algorithm would quickly become prohibitively expensive to run.

5.3.2.11 Iterative class ensemble

Table 5.11: Iterative class ensemble results

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>65.20%</td>
<td>0:01.296</td>
</tr>
<tr>
<td>Max</td>
<td>81.95%</td>
<td>19:33.126</td>
</tr>
<tr>
<td>Mean</td>
<td>71.52%</td>
<td>7:00.622</td>
</tr>
</tbody>
</table>

The outcome is slightly better than regular class stacking, but not too much. The problem here is, from what we have seen in debugging traces, that performing several rounds of classification doesn’t really improve the prediction. The result in the first round is often the result in the second, and hence the idea of “refining” the classification by eliminating irrelevant classes doesn’t work in practice.
Furthermore, as we mentioned before, it is usual to have a heavy bias towards predicting “doesn’t belong”. As we saw in figure 5.27 it is very rare that more than one model predict “belongs” at the same time, making the whole objective of this algorithm unnecessary.

5.3.2.12 Reinforced class ensemble 1

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>33.40%</td>
<td>0:01.986</td>
</tr>
<tr>
<td>Max</td>
<td>70.24%</td>
<td>5:07.229</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td><strong>58.89%</strong></td>
<td><strong>1:33.885</strong></td>
</tr>
</tbody>
</table>

Table 5.12: Reinforced class ensemble 1 results

As expected, this classifier offers worse results than class stacking. The advantage of training with different sets is overshadowed by the significantly lower amount of instances to train with.

Figure 5.30: Results for the reinforced class ensemble 1 model.
### Reinforced class ensemble 2

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>46.80%</td>
<td>0:04.508</td>
</tr>
<tr>
<td>Max</td>
<td>75.21%</td>
<td>2h:36:25.231</td>
</tr>
<tr>
<td>Mean</td>
<td>61.75%</td>
<td>32:26.814</td>
</tr>
</tbody>
</table>

**Table 5.13:** Reinforced class ensemble 2 results

Accuracy in this case is also bad, but the most alarming result is actually the very large variance it has shown in all data sets. Even if the average accuracy is the first measure we look, variance is important too: if the accuracy varies wildly between realizations it means that it is highly dependent on the training set. That has unfortunate implications, since we need to guarantee that a model works for all kinds of samples from the same population. An unreliable model is as troublesome as a bad one.

![Graph](image)

**Figure 5.31:** Results for the reinforced class ensemble 2 model.

The reason for this variance is difficult to pinpoint. My opinion is that the algorithm is not focused enough in different areas:

1. We are assuming that failed instances are always more necessary than correct ones, but for a good classification both need to be present in the appropriate quantities. Boosting solves this problem by using weighted instances, but our binary approach (add or remove) doesn’t seem to work.

2. When we remove instances from the training set we do so randomly. This lack of focused criteria might end up introducing uncertainty in the model.

3. Finally, our suspicion is that by changing iteratively the training and test set we might be “cheating” and putting more effort into finding a good test set for our algorithm than fitting our model to a given set. This probably ends with an overfit model that works poorly on external test sets.
5.3.3 Classifier meta-analysis

To finish our classifier analysis, we have condensed all the information in single average values between all data sets, for each algorithm. With this averages we can build the following ranking:

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Average accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacking</td>
<td>78.46%</td>
</tr>
<tr>
<td>Neural network</td>
<td>77.70%</td>
</tr>
<tr>
<td>3-NN</td>
<td>76.15%</td>
</tr>
<tr>
<td>Random C4.5 forest</td>
<td>75.87%</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>74.51%</td>
</tr>
<tr>
<td>AdaBoost.M1</td>
<td>74.37%</td>
</tr>
<tr>
<td>C4.5 decision tree</td>
<td>73.30%</td>
</tr>
<tr>
<td>Naive bayes</td>
<td>72.88%</td>
</tr>
<tr>
<td>Soft class stacking</td>
<td>72.37%</td>
</tr>
<tr>
<td>Iterative class ensemble</td>
<td>70.47%</td>
</tr>
<tr>
<td>Class stacking</td>
<td>69.16%</td>
</tr>
<tr>
<td>Reinforced class ensemble 2</td>
<td>63.29%</td>
</tr>
<tr>
<td>Reinforced class ensemble 1</td>
<td>60.74%</td>
</tr>
</tbody>
</table>

Table 5.14: Overall classifier accuracy ranking

An unexpected surprise: stacking is the winner despite not being very impressive in four out of six data sets. The reason is clear and it was mentioned in the first point in section 5.3.2.8: its inherent consistency is due to acting as a selector that uses the best base
classifier for each set. While this means that it’s not usually the best, it is rarely bad and thus we can consider it the most reliable classifier of all those seen in this project. A significant drawback exists: it can take a considerable amount of time, specially in large or complicated sets.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Average training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive bayes</td>
<td>0.628 s</td>
</tr>
<tr>
<td>3-NN</td>
<td>5.054 s</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>8.575 s</td>
</tr>
<tr>
<td>AdaBoost.M1</td>
<td>8.866 s</td>
</tr>
<tr>
<td>Neural network</td>
<td>9.426 s</td>
</tr>
<tr>
<td>C4.5 decision tree</td>
<td>29.695 s</td>
</tr>
<tr>
<td>Reinforced class ensemble 1</td>
<td>1 min 33.885 s</td>
</tr>
<tr>
<td>Random C4.5 forest</td>
<td>5 min 16.751 s</td>
</tr>
<tr>
<td>Class stacking</td>
<td>5 min 39.074 s</td>
</tr>
<tr>
<td>Iterative class ensemble</td>
<td>7 min 1.622 s</td>
</tr>
<tr>
<td>Reinforced class ensemble 2</td>
<td>32 min 26.814 s</td>
</tr>
<tr>
<td>Stacking</td>
<td>3 hours 25 min 10.871 s</td>
</tr>
<tr>
<td>Soft class stacking</td>
<td>19 hours 37 min 14.590 s</td>
</tr>
</tbody>
</table>

Table 5.15: Overall classifier training time ranking

In second place we have neural networks. These networks are widely used with excellent performance in many applications, and these results confirm that judgment. Also, in our study they were not specially slow, though we considered relatively small sets. This can change if we need to learn from bigger sets in the future.
In third position we have 3-NN, which probably has the greatest accuracy/training time ratio. It has good accuracy with a fast training time relative to the others.

AdaBoost.M1 lies in the middle of the ranking, despite the expected high performance in other research. We have to note that this less-than-stellar position is not result of low accuracy, but inconsistency. AdaBoost.M1 can be very good in some sets but not as much in others, so before using it we must make sure that it’s appropriate for that set.

Naive Bayes has worked as expected, acceptable accuracy with an extremely fast training. It will be a very useful algorithm when circumstances require time-sensitive learning.

As a final observation, all our attempts at creating new algorithms have failed, for the most part. This was expected; it was not our intention to create any new revolutionary algorithm but to try intuitive procedures and analyze why they don’t work. This analysis was the reason we looked for the other error measures discussed in the previous chapter, in order to explain the reason they fail. We think this analysis was worth it, and our conclusions on the previous sections of this chapter for each algorithm will be useful for Krimtrack’s future.

### 5.3.4 Regression algorithms

Moving onto regression, we present the results of the imports data set to make sure the algorithms return sensible results.

<table>
<thead>
<tr>
<th></th>
<th>Linear regression</th>
<th>k-NN regressor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAE</td>
<td>1753.59</td>
<td>2840.35</td>
</tr>
<tr>
<td>Training time</td>
<td>0:00.462</td>
<td>0:00.012</td>
</tr>
</tbody>
</table>

Table 5.16: Results for imports data set

![Results for the imports data set.](image)

Figure 5.34: Results for the imports data set.

Linear regression gives us estimates with an absolute mean error of approximately $1700 while the 3-NN errs by $2800. We established previously that the range for the car
prices in this set is between approximately $5000 and $45000, which makes the error seem reasonable.

With that said, error significance in regression is not as straightforward to evaluate as in classification. Ultimately the final user of this prediction will have to assess whether the model is accurate enough depending on the domain he deals with.

Lastly, both algorithms are very fast for this small data set. But even in this case we can observe that the k-NN regressor is notably faster by more than an order of magnitude. As always, the trade-off between error and training time is something that must be considered in a case by case basis.

5.3.5 Filters

5.3.5.1 Feature filters

In the next experiment we will compare how feature filtering affects the accuracy of each algorithm and data set. We have two options: the regular CFS filter as introduced in the original paper[11] or our option of applying this same filter separately to each class. We present the results in figure 5.36.

The first thing we learn is immediately obvious: applying CFS per class doesn’t work. In some particular cases the accuracy remains the same or even increases a little, but globally we can see that it is not a good filtering method.

That leaves us with the regular CFS filter. It provides no significant improvements in accuracy terms, and it degrades notably the k-NN since it doesn’t rely on the same correlation metric the filter uses. The most important use of this filter is to dramatically speed up the training stage by reducing the set dimensionality.

![Figure 5.35: Data reduction for CFS filtering.](image)
Chapter 5. Results

<table>
<thead>
<tr>
<th></th>
<th>Original features</th>
<th>Average features after CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audiology</td>
<td>69</td>
<td>4</td>
</tr>
<tr>
<td>Column</td>
<td>6</td>
<td>3.9</td>
</tr>
<tr>
<td>Breast cancer</td>
<td>9</td>
<td>3.4</td>
</tr>
<tr>
<td>Fourier (original)</td>
<td>76</td>
<td>12</td>
</tr>
<tr>
<td>Fourier (second filtering)</td>
<td>12</td>
<td>10.8</td>
</tr>
<tr>
<td>Credit</td>
<td>20</td>
<td>4.4</td>
</tr>
<tr>
<td>Nursery</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.17: CFS feature reduction

Note how the fourier set is barely affected. Since we already applied the filter once, a second selection doesn’t alter the set much more, but the original set was reduced from 75 to just 12. Conversely, the audiology feature set is drastically reduced, with only 5% of the features remaining after the filter.

Figure 5.36: Results for feature filtering.
5.3.5.2 Instance filters

This test will be performed on the k-NN algorithm, the only one where instance filtering can be critical in very large sets. First we will consider the filter published in academic research, the FCNN rule, in figure 5.37.

Our output is coherent with the author’s results[12]. We see a slight decrease in accuracy (5% at most) but a radical filtering of data, from a minimum of 50% to a 80% reduction in the nursery set. In this case not only it removes instances drastically but this filtering actually increases accuracy a little bit, even if it might not be statistically significant. All things considered, we think it is a more than useful filter with excellent results.

Once seen the performance of the FCNN rule, we will compare it to our own attempt at entropy-based instance filtering in figure 5.38. We will set the instance deletion ratio at 50% to have a comparable filter to FCNN.

Our entropy-based filter gives a strange outcome. Sometimes it produces good accuracy, like in the breast cancer and credit set but in the others it shows poor performance (exceptionally poor in, again, the nursery set). Curiously enough, those two sets that
work correctly with this filter are the ones in which all the classifiers have trouble and show approximately the same results. We suspect that in these sets there is an abundance of instances that cannot be taken advantage of for learning, and removing those makes no difference. In the large sets (nursery and fourier) the accuracy drops, probably because we are removing a lot of instances and we have more risk of filtering the “wrong” ones than in smaller sets.

Anyway, it is obvious that we cannot rely on such an inconsistent filter, so we won’t use it in a production environment.

5.3.6 Test methodologies

In this section we will detail the experiments to evaluate the different methodologies and how they compare to each other. Given how many different permutations exist, we only present the results and plots that are worth a discussion.

5.3.6.1 Holdout ratio

The standard practice in the field is to use a holdout split of two thirds for training and one third for test. We tried different split points to see how it affects the estimation’s bias and variance. The experiment consisted in estimating the mean and variance of 10 iterations for several percentages of training sizes and for different data sets and algorithms.

The overall conclusion among all data sets is the expected: at first, the larger the training set the better the model becomes. For higher training percentages the curve starts to decrease; we have a small test set that is no longer representative of the overall statistical distribution and it has a negative impact on the estimation.

Furthermore, the curves are more or less parallel between different algorithms so we can conclude that holdout methodology provides a good estimation, and a 66.6% is a good trade-off between bias and accuracy (see below).

In the breast cancer case we can confirm once again that the set is difficult to learn. It doesn’t matter much how we split the set, the classifiers are not able to improve with more instances.

A large amount of instances, like in the fourier or nursery sets, produces a smoother curve. Also, we don’t have a decrease in accuracy for high training percentages. Even small test sets are enough to be statistically representative of the whole distribution.
We can observe another problem, mostly on the last two points, which is that the confidence interval grows larger even if the accuracy keeps increasing. Thus we validate the bias/variance tradeoff we mentioned in chapter 4.

As a curiosity, plot 5.41 also shows more information about the models themselves. We can infer, for instance, that decision trees need a lot of instances for the slope to reach an almost constant value around 60%, while naive bayes stabilizes around 30%. It is
noticeable too how the accuracy improvement between forests and trees is constant along the curve.

5.3.6.2 Cross-validation folds

Similarly to the previous experiment, the standard number of folds in cross-validation is 10. We swept through several number of folds and present the following relevant results, next to a line representing the 66%-holdout estimate to compare.
All sets report similar results, so we choose the audiology results in figure 5.43. With 2 folds the cross-validation estimate is lower than the holdout’s, but for more folds it has a higher expected accuracy.

It looks like at least one of them must be wrong, but the reality is more subtle: they are both evaluating a different model as a proxy for the final model. In this holdout we are using two thirds of instances to train, while in cross-validation we are using an averaging of models built with a $\frac{k-1}{k}$ proportion of all the instances, where $k$ is the number of folds. For $k = 2$ we use one half of the set, which is less than the two thirds of the holdout and explains the lower estimate in cross-validation. But in the other points we use more instances and hence we are evaluating a better model. This observation shows that the holdout estimate can have a large bias, since the proxy model tested might be very different from the final one.

Another fact we can validate is that cross-validation has lower variance than holdout, as long as $k$ is not too large. As we explained, when $k$ grows the bias tends to zero ($\frac{k-1}{k}$ tends to one, and therefore the proxy model tends to have the same size as the final one), but the variance tends to infinity and an individual estimation can vary greatly if we are not careful. In figure 5.43 we can see as well that 10 is a good number of folds with a reasonable bias/variance balance.

As conclusion for this experiment, then, we can see why most research is performed by cross-validation testing: it allows the use of more data (we don’t discard anything
for training like holdout does) with a reduced bias and variance, even if it is costly to compute.

5.3.6.3 Cost and confusion matrices

As a final experiment for testing methodologies we will see the usefulness of cost and confusion matrices in a real world application. As we said in chapter 4, the credit data set author suggests a cost matrix that penalizes bad clients classified as good five times more than the other way around.

<table>
<thead>
<tr>
<th></th>
<th>good</th>
<th>bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>206</td>
<td>28</td>
</tr>
<tr>
<td>bad</td>
<td>61</td>
<td>39</td>
</tr>
</tbody>
</table>

Table 5.18: Confusion matrix of a k-NN classification for the credit data set

For the confusion matrix at table 5.18, the standard accuracy measurement would be 73.4%. Adjusting the same test to the suggested cost matrix, the accuracy would be 42.3%, reflecting the new information about the domain we introduced. Such a low score means that our classifier is making a lot of grave mistakes that in the real world would have a bigger repercussion than what the first accuracy measurement would suggest.

For completeness’ sake we will compare a part of the study we obtained in section 5.3.1.5 with the cost matrix-adjusted accuracies in figure 5.44.

For the third time we can see this set is specially difficult, and now we can be aware how poor these classifiers are. More importantly, when we first saw the original results we concluded that AdaBoost.M1 was the most accurate model, but with this adjustment we see a neural network would have a more positive outcome.
One last thing to keep in mind is that cost matrices are only used to calculate a weighted accuracy to reflect real world constraints, but they are not able to affect the training stage. Cost-sensitive learning is an interesting object of study that lies beyond the scope of this project at the moment.

### 5.3.7 ROC analysis

In this experiment we will visualize the error of the logistic regression classifier on the fourier set using ROC analysis. This analysis should be consistent with the conclusions on figure 5.23 using boosting. After extracting the test set prediction probabilities for each class and performed the necessary calculations for a sufficient number of points, the ROC curves obtained are in figure 5.45.

![ROC analysis for logistic regression on fourier set.](image)

Results are definitely similar to what boosting tells us. Digits 0 and 8 are almost perfect on classification, while digits 1, 6 and 9 are quite poor. In the particular case of digit 9, at some points the curve almost falls below the no-discrimination line. Digit 6 doesn’t seem as bad as the boosting graph suggested, but note how the slope near the (0,0) point (threshold equal to 1) is less steep than most other digits. This suggests that even
if we relax the threshold a little bit a significant amount of other digits (probably 9s) get classified as 6 (false positives).

Continuing with a deeper analysis, we can calculate the optimum cutoff point for each digit and compare how the binary statistics change when we use it.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Optimal threshold</th>
<th>Threshold=0.5</th>
<th>Threshold=optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3</td>
<td>1</td>
<td>0.957</td>
</tr>
<tr>
<td>1</td>
<td>$10^{-8}$</td>
<td>0.548</td>
<td>0.259</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>1</td>
<td>0.786</td>
</tr>
<tr>
<td>3</td>
<td>$10^{-8}$</td>
<td>0.849</td>
<td>0.685</td>
</tr>
<tr>
<td>4</td>
<td>$10^{-4}$</td>
<td>0.828</td>
<td>0.684</td>
</tr>
<tr>
<td>5</td>
<td>$10^{-4}$</td>
<td>0.905</td>
<td>0.808</td>
</tr>
<tr>
<td>6</td>
<td>$10^{-7}$</td>
<td>0.425</td>
<td>0.426</td>
</tr>
<tr>
<td>7</td>
<td>$10^{-10}$</td>
<td>0.839</td>
<td>0.621</td>
</tr>
<tr>
<td>8</td>
<td>0.3</td>
<td>1</td>
<td>0.901</td>
</tr>
<tr>
<td>9</td>
<td>$10^{-4}$</td>
<td>0.195</td>
<td>0.134</td>
</tr>
</tbody>
</table>

Table 5.19: Precision and recall comparison for binary classification of fourier set using optimal thresholds

This threshold’s optimization goal is to maximize the difference between TPR and FPR. It is not guaranteed to improve accuracy, but it tries to balance the different types of error. In table 5.19 we see that in most digits precision and recall are in balance so increasing one decreases the other. These new thresholds favor recall over precision, which in practice means that we classify more instances as positive and the model becomes more inclusive. By becoming more inclusive it adds more positive instances previously left out, but it also adds false positives. The optimal “size” of this inclusiveness is the one determined by this threshold.

Note how digit 6 is a special case. With the optimum threshold we improve both precision and recall, a hint that the default 0.5 threshold might not be good enough and in a real application it would be necessary to adjust it accordingly.

5.3.8 Model selection

We noted in the first experiment in this chapter that we didn’t use model selection to select the training algorithm’s hyperparameters because it would slow down the tests. Now we will see whether model selection improves the final model and the level of improvement.

We will use k-NN for this experiment. With model selection the algorithm tries to find the optimal $k$ for the current set. The results of the executions (again, the average of
ten iterations) will be compared to those obtained with a standard $k = 3$. A split with 50% training, 25% validation and 25% test will be used for the experiment.

![Model selection in k-NN](image)

**Figure 5.46:** Model selection performance for k-NN.

Model selection improves noticeably the accuracy in four sets, specially in nursery. We have to remember that by using model selection we are using 16% less instances for training (corresponding to over 2000 instances in the nursery set!) and yet the accuracy is increased. It is easy to see that model selection is definitely valuable if we want to maximize performance.

The breast cancer set is slightly increased, but on the fourier set model selection actually degrades the algorithm result. Probably the reason for this decline is the same as we mentioned before: we discard too many instances for training (34 instances per class out of 200) and the selection process cannot improve the model enough to be worth it. Therefore, we must be careful about the need for instances of each algorithm and data set before using model selection blindly.

### 5.4 Machine learning integration with Krimtrack

We have insisted that machine learning is useful in a large variety of situations and interactions with a vast amount of different systems. We will use it “manually” in data analysis (like we demonstrate in the next chapter), but as a simple example of the potential of this module we have integrated the machine learning inference into the coordination interface.
In order to have a visual result, we have decided to paint the zones in the system as a function of the number of expected incidents that day, using a gradient ranging from green (no incidents) to red (more than eight incidents). Obviously this threshold and the following theoretical model is completely arbitrary and have been chosen as a proof of concept. Since we are trying to predict a numerical value we will use regression, specifically a 5-NN regressor.

To accomplish this we need to establish the features we will use. Each instance will represent a zone, and to make things easy we will define five features describing them:

1. Latitude of the zone’s center
2. Longitude of the zone’s center
3. Radius of the zone
4. Day of the month
5. Month

Finally, the label to predict will be the number of incidents in that day and zone.

Next, we have to build a training set. We create 8 zones centered around the UPC north campus and 250 random distinct dates for each zone, with a total of 2000 instances. To simulate the number of incidents we will make up a theoretical stochastic model, using those five previous features. We won’t be concerned to have a realistic simulation model. With all these, we have defined the estimated number of incidents $x$ as follows:
\[ x_{\text{dist}} = U(0, 1) \cdot \left( 8 - \frac{\text{distance}(\langle \text{latitude, longitude} \rangle, P)}{30} \right) \]

\[ x_{\text{radius}} = U \left( 0, \frac{\text{radius}}{20} \right) \]

\[ x_{\text{day}} = U \left( 0, \frac{\text{day}}{10} \right) \]

\[ x_{\text{month}} = \begin{cases} 
U(0, 1) & \text{if month} \in \{1, 2, 3\} \\
U(0, 2) & \text{if month} \in \{4, 5, 6\} \\
U(0, 4) & \text{if month} \in \{7, 8, 9\} \\
U(0, 1) & \text{if month} \in \{10, 11, 12\} 
\end{cases} \]

\[ x = \max(0, x_{\text{dist}} + x_{\text{radius}} + x_{\text{day}} + x_{\text{month}}) \]

where \( U(a, b) \) is a uniformly distributed random value between \( a \) and \( b \) and \( P \) is a constant arbitrarily chosen point in the map near the zones. This model has generated instances with a number of incidents between 0 and 10.75 (figure 5.47). Even though the number of incidents should be discrete we will keep the decimals and pretend this number is an average of several years to have a more diverse data set.

\[ \text{Figure 5.47: Histogram for the incident data generated.} \]

Once we plug the machine learning module into the coordination system, we tweak the communication server to return additional information about the incident risk in zones to the coordinators. The predictive model is trained once a day, with a MAE of less than 1 incident. When the coordinator application requests zoning information the machine learning module is asked to perform regression on the zones and compute its incident risk prediction. Since we are using 5-NN the regression will be very fast, and the low amount
of coordinators means that these calculations will not be a performance bottleneck for the communication server.

The coordinators visualize the information in the interface pictured in figure 5.48. The first screenshot takes place in (a simulated) March 20th, the second in May 2nd and the third in October 7th. We can see how there are different incident risks depending on the zone and the day we are considering. Therefore, we have shown how machine learning can be integrated in Krimtrack, how it can be very useful to users and quite easy to accomplish once we have decided on one of its many, many possible uses limited only by our creativity.
Chapter 6

GTD database: an example of data preprocessing and analysis

In chapters 4 and 5 we studied different machine learning techniques, their applications and the results of our experiments. But there is more to this field than algorithms; a critical stage of data analysis is the acquisition, preprocessing and understanding of the data set, as well as making sense of the model output before it is used in the real world.

In this chapter I will show a brief example of a real machine learning application in a domain similar to the one we will be encountering in Krimtrack. The perfect example would be analyzing data generated by our system, but we don’t have a reliable source that represents the kind of statistical distribution we will be encountering, so instead we will use the GTD database as a data source with a domain close to ours.

We remark that the objective of this section is not the inference itself; we discussed that subject in the previous two chapters. My intention is to focus on the preprocessing stage in a real world case where we don’t have a nicely packaged data set like the ones we worked with in chapter 5. Also, the models generated will be used to extract conclusions that provide us (hopefully) with some knowledge about real world terrorism.

6.1 GTD database overview

The Global Terrorism Database (GTD) is an open source database maintained by the National Consortium for the Study of Terrorism and Responses to Terrorism (START) that includes terrorist incidents from all over the world since 1970[32]. It has a clearly defined methodology about the data acquisition, meaning and criteria for inclusion. The mandatory requisites for inclusion are:
• The incident must be intentional.

• The incident must entail some level of violence or threat of violence.

• The perpetrators of the incidents must be sub-national actors (i.e. state terrorism is excluded).

In addition, two of the three following items must apply too:

1. The act must be aimed at attaining a political, economic, religious, or social goal.

2. There must be evidence of an intention to coerce, intimidate, or convey some other message to a larger audience (or audiences) than the immediate victims.

3. The action must be outside the context of legitimate warfare activities.

In instances where these definitions are blurred and overlap with other types of crimes, the GTD allows an incident to be tagged as doubtful.

The version we will use of the GTD has 133 attributes and 113113 instances with incidents from 1970 to 2012, even though a new version with incidents from 2013 has been released while this project was being developed. The attributes and their possible values are explained in the GTD codebook[33], with a small sample attached in appendix B as an example.

There are also some more particularities to have in mind as well:

• Due to data loss (prior to the GTD compilation) only an estimated 15% of incidents were recovered from the year 1993. To avoid statistical misrepresentation, START decided to remove all instances from that year from the database although they are available in the GTD codebook as an appendix.

• Some attributes were only available from post-1997 instances, and they are missing in earlier data.

• Obviously, the mentioned criteria for considering an incident a terrorism act are not universal and many scholars in criminology have other definitions. START decided to err in the side of inclusiveness and expect researchers to filter the data for their needs using the appropriate features. That is precisely the reason why they document their coding decisions so thoroughly in the GTD codebook.
• In 2012 START adopted a new methodology for data gathering, using natural language processing (NLP) and machine learning techniques over worldwide media articles in order to identify terrorism acts. This new methodology resulted in a dramatic increase of incident detection from 2011 to 2012, and it’s unclear how many can be attributed to the methodology and how many represent an actual increase in terrorism in the year 2012.

6.2 Initial preprocessing

We start the process by downloading the 54MB data set in spreadsheet format and open it for a first review. Independently of the hypotheses we want to prove, we quickly see that many attributes are unnecessary or of little relevance and would only slow down the learning process, in particular:

1. **eventid** is a unique identifier for each row that provides no information.

2. **approxdate**, **resolution**, **provstate**, **city**, **location**, **gsubname**, **divert** and **kidhijcountry** are attributes whose values are too sparse (i.e. they have very low entropy) to provide useful information.

3. **country**, **region**, **alternative**, **attacktype1**, **targtype1**, **corp1**, **natlty1**, **targsubtype1**, **target1**, **claimmode**, **weaptype1**, **weapsubtype1**, **propextent** and **hostkidoutcome** are numerical codes for information that are already available as text in other attributes, so they become redundant.

4. **summary**, **motive**, **weapdetail**, **propcomment**, **ransomnote** and **addnotes** are strings that describe some information in natural language. In the future it would be useful to use natural language processing (NLP) to extract features for machine learning purposes, but at the moment we will exclude it.

5. **attacktype2**, **attacktype2_txt**, **attacktype3**, **attacktype3_txt**, **targtype2**, **targsubtype2**, **corp2**, **target2**, **natlty2**, **natlty2_txt**, **corp3**, **targtype3**, **targsubtype3**, **target3**, **natlty3**, **natlty3_txt**, **gname2**, **claim2**, **gname3**, **gsubname2**, **gsubname3**, **guncertain2**, **guncertain3**, **claimmode2**, **claim3** and a few other attributes represent different items from lists. In the few incidents where there are more than one of those items they are added in these fields. Unfortunately, limitations on the machine learning algorithms would make them try to assign meaning to each one of them individually, even though they are interchangeable; i.e. the ordering doesn’t matter. To avoid confusing the model...
training algorithm, we will only leave the first attribute from each of these elements 
\((\text{attacktype1\_txt}, \text{targtype1\_txt}, \text{targsubtype1}, \ldots)\).

6. \textbf{scite1, scite2, scite3} and \textbf{dbsource} are the sources where the instances were 
gathered, and they are not relevant to our study.

From 113 attributes we trim down the set to 52. We are doubtful about one of the 
remaining attributes, \textbf{gname}, that represents the name of the terrorist group. On one 
hand it might be useful to keep potentially profitable information, but on the other it is 
a quite sparse feature with 1487 distinct values that can dramatically slow down learning 
with little benefit.

### 6.3 Target type prediction

Our goal will be to build a prediction model for guessing target types for terrorist attacks. 
The GTD recognizes 22 types (even if 5 of them concentrate most of the probability) 
distributed as shown in figure 6.1. The baseline accuracy is 22.1\% and we can observe 
from this distribution that the five more likely targets for terrorists are, in order, private 
citizens, businesses, government, police and the military.

Before we begin our analysis we will remove the feature \textbf{targsubtype}. If we are trying to 
guess the target type it is not realistic to think that we will have its subtype available. 
Besides, we would be cheating since there is a deterministic correspondence between 
subtype and type, so we would have nearly 100\% accuracy if we included it.

We start by building a naive bayes model as a first quick approximation. We test it for 
an unimpressive accuracy of 32.96\%.

Since our first attempt didn’t work very well we try another fast approach by using a 
3-NN model. After 32 minutes of testing the model returned an accuracy of 46.66\%. 
Better, even if it’s still not too much. The 22x22 confusion matrix is too large to include 
in this document; however, we don’t notice any particular error concentration, they all 
look similarly balanced so we can’t think of a way we could improve the model this way. 
Another attempt was made by building a C4.5 decision tree, but it took many hours 
and we decided to abort the execution and follow other paths.

Perhaps we can improve the model and speed up training by filtering irrelevant features.
We will use the CFS filter to find out which attributes are considered more important. 
Once filtered, the data set is drastically reduced to just \textit{gname} and \textit{crit3}. It is suspicious 
that the model would only need these features, and we think the problem is that we
are overfitting the model to gname. There are so many terrorist organizations that for one given group there are only a handful of instances to learn from. But, in a real world scenario, knowing the name of the terrorist group could be difficult and it would change the prediction radically if we were wrong. Hence, we will try to keep as much information as possible by transforming the feature gname into another one we will call isindividual that will be 1 if gname is “Individual” or “Unknown” and 0 otherwise.

With this modification the decision tree is made in minutes, with an accuracy of 48.16%. Again, the whole tree is too large to fit in this document, but a partial representation of the first levels is pictured in figure 6.2.

6.4 Descriptive analysis

One of the reasons we made a decision tree is for its user-friendly structure that allows us to easily interpret the model. From that tree we can deduce some rough facts about
terrorism:

- When the third criterion (incident outside international humanitarian law) is not met, the target is usually the military. These incidents could be mistaken for war actions, except the two first two criteria are necessarily met (i.e. political, economic, religious or social goal and intention to coerce or intimidate larger audiences) and hence disqualify them from being legitimate war incidents.

- Discriminating by country is very common throughout the whole tree. That gives the impression that terrorism is very dependent on which country the incidents occur.

- The most used attributes are country, target’s nationality and type of weapon. These are features with a lot of values, so the tree grows wide very quickly.

- The previous point implies that a lot of branches are built without any instances and the algorithm is forced to infer based on the most popular class of the parent node. Thus a large portion of the tree is nothing more than educated guesses which explain the disappointing accuracy. We have to be careful to use those guesses to extract conclusions.

- The month and day of the month are rarely used in the tree. It looks like there is no significant correlation between the kind of target and the date of the incident.

We could analyze the tree further but the model sparsity suggests that we have too few instances to generalize to such a global context. Our deductions above are too general.
to be interesting. In order to improve our prediction we will try to limit our analysis to Spanish incidents and see the differences.

Removing all non-Spanish incidents leaves us with 3234 instances and a baseline accuracy of 31.47%, but unfortunately the model is even less accurate: 45.45%. Extending it to a 50-tree forest we reach the highest accuracy of 50.18%. Checking the trees in the forest, we confirm that the trees have the same problem we had previously. They tend to use day and month to classify, overfitting the model. By diversifying the model with several trees we mitigate the effect, but the final accuracy is still poor.

6.5 Analysis conclusions

All this process has shown us that terrorism is a very difficult subject to foresee. Obviously, the main obstacle to fight terrorism is this unpredictability, and even though we managed to extract information (we improved by 20% the baseline accuracy) we are far from a definitive solution.

![Accuracy over learning procedure stages](image)

**Figure 6.3:** Chronological evolution of the GTD analysis performed.

We could probably improve our predictive power by further processing the data set, cross-referencing other data sources with the GTD and using other statistical techniques. That kind of deep, domain-centric analysis is beyond the scope of this chapter, which
aimed to show how machine learning is not an automated procedure but a dynamic, creative process that needs expert knowledge, ample time and plenty of data.
Chapter 7

Conclusion

During Krimtrack’s development we have had to work on its several large subprojects at once. Each one of them required a significant amount of time: the mobile application, the communication server, the coordinator web application and the main focus of this document, the machine learning module.

The global system’s functionality is in place and working as expected, even if the interface needs some additional work to be smoother and more aesthetically pleasing. With the load test we guaranteed that even with a small, home server the system is able to comfortably fit hundreds of users simultaneously. Limitations on the simulation methodology have prevented us from securing a more realistic user limit in a production scenario, but the number estimated is a lower bound good enough to start.

The machine learning aspect of this project has allowed us to obtain interesting results that will be undoubtedly useful in their application to this system as well as many other potential needs. We have characterized a substantial variety of machine learning algorithms according to several metrics: accuracy, speed, complexity and accountability, among others. Each of them excels in a particular set of circumstances, and that permits us to satisfy a vast array of learning needs.

After studying the different machine learning data sets, we have seen that there is no optimal, general path to always obtain the best results. Each one of these sets has to be studied carefully, considering the different metrics we have mentioned along this document to be able to contextualize the results. Also, these approaches ought to be contrasted and verified by research literature when possible. Some of the algorithms we designed prove to be occasionally useful but show poor performance in some cases. Such unreliability is dangerous when we are trying to learn in unknown circumstances.
Finally, we have seen a brief example of how we can use this machine learning module in a larger data analysis context. The results confirmed us that terrorism is indeed a difficult subject to predict, but even then we managed to obtain some preliminary information about the most relevant features to predict target types. We have seen that data analysis is not a trivial process, and that many steps are required beyond machine learning to produce decent results.

### 7.1 Future research lines

With that said, Krimtrack’s current state is just the beginning. We have a vast amount of ideas for the future to continually improve the service we will be offering.

Starting by Krimtrack’s coordinator interface, a hard problem to solve is to generate clean and at the same time truthful routes from the field users’ GPS locations. Other applications have constraints that simplify this problem enough to be easily tackled (e.g. car navigation assumes a car always moves along roads without abrupt changes in velocity). We don’t have any constraint here: guards can be on roads or buildings, on foot or in any kind of vehicle, standing by or moving.

Also, locations for our clients can potentially be indoors, which excludes GPS geolocation. We have looked for some GPS-like indoor solution: there are many of them but none are standardized as far as we know. Besides, the visualization would need to change as well, a map API like Mapquest would be replaced with a building blueprints API, for instance.

To increase the communication options, it would be interesting to eventually add Voice over IP (VoIP) functionality. The coordinator would be able to communicate by voice with the field users in a more natural way, but the infrastructure needed is complex enough to require a more thoughtful design to implement.

As the system grows so will the data generated, and benchmarks show that simple relational databases don’t scale too well. That is why at some point we’ll have to migrate to a noSQL solution, with a whole different paradigm on data management. Research needs to be done and tests performed before changing such a critical subsystem.

Moving onto machine learning, possibly the most important would be the optimization of the classification methods implemented. We made an effort to decrease the running time of the training algorithms, but the reality is that Weka’s implementation run in some cases noticeably faster. This will imply code profiling and work with more optimal data structures, as well as taking more advantage out of multithreading CPUs.
Talking about classification, we have missed the opportunity (and the time) to include other classifiers that we found interesting, such as support vector machines (SVMs), other kinds of neural networks (like recurrent neural networks or restricted Boltzmann machines) and bayesian models (like bayesian networks).

As far as regression is concerned, we obviously just scratched the surface with the two regressors presented in this document. Regression is a tremendously useful technique that we will undoubtedly use in many occasions, so more work in this field will be imperative in the near future.

Machine learning is just one field among many coming from artificial intelligence. Other AI fields can be useful; for instance, we encountered in the previous chapter some database features that included strings that could not be used directly in classification. Using natural language processing (NLP) we would be able to extract useful information and include it in the machine learning operation, improving our results even further.

Yet another interesting machine learning application, specially in our case, is clustering. Clustering is an unsupervised learning technique that groups instances by similarity in different clusters. This technique is widely used in predictive policing and crime analysis, such as the identification of areas with greater incidences of particular types of crime, known as “hot spots”. Some work in clustering has already been implemented in the machine learning module (specifically, the k-means algorithm) but was left out of this document for the lack of a clear evaluation metric.

As the last point, there are other, more sophisticated techniques used in predictive policing that we haven’t had the time to study properly. For example, Dr. George Mohler developed a crime predictive system based on clustering and advanced statistics that considers crimes analogous to earthquakes, producing “aftershocks” in time that can be predicted[34]. This and other statistical approaches would be interesting and ultimately necessary to be competitive in a predictive policing market.

### 7.2 Closing words

Beyond the mere engineering aspect, many other challenges await. For starters, we have to learn to work as a company: budgets, time management, human resources, client negotiations and many other areas will need to be handled on our own.

Our intention is for Krimtrack to offer a valuable service to large companies, with a real need for coordination in all aspects of their operation. This coordination takes many forms: in the past (historical records), the present (real-time feedback) and the future
(resource optimization and allocation). Using state-of-the-art technologies in a Big Data paradigm we aim to provide our clients with all the necessary information to make their businesses thrive.

We have a long way ahead of us and there will be many obstacles to overcome. Thankfully, we have the education, capabilities and enthusiasm to keep moving forward and create our own future for ourselves.
# Appendix A

## Krimtrack Protocol (KTP)

### Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGIN</td>
<td>Terminal</td>
<td>Login petition by mobile terminals.</td>
</tr>
<tr>
<td>LOGOUT</td>
<td>Terminal / Coordinator</td>
<td>Logout petition.</td>
</tr>
<tr>
<td>GET_USER_LOCATIONS</td>
<td>Coordinator</td>
<td>Petition by coordinator for current position of connected users.</td>
</tr>
<tr>
<td>UPDATE_POSITION</td>
<td>Terminal</td>
<td>Notification about new position (latitude, longitude and standard deviation error).</td>
</tr>
<tr>
<td>WARNING</td>
<td>Terminal</td>
<td>Notification about new warning in the field, with its name, type and location.</td>
</tr>
<tr>
<td>END_WARNING</td>
<td>Terminal / Server</td>
<td>Notification about end of previously reported warning.</td>
</tr>
<tr>
<td>REQUEST_ZONES</td>
<td>Terminal / Coordinator</td>
<td>Request for information about zones in the system: a list of names and coordinates.</td>
</tr>
<tr>
<td>Command</td>
<td>Originator</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REQUEST_SCHEDULE</td>
<td>Terminal</td>
<td>Request for current schedule for active user: a list of items, each with a zone and the scheduled time of arrival.</td>
</tr>
<tr>
<td>REQUEST_PHONE.CONTACTS</td>
<td>Terminal</td>
<td>Request for phone contact information: a list of contacts, each with a name, a phone number and an icon.</td>
</tr>
<tr>
<td>REQUEST.ICE_TYPES</td>
<td>Terminal</td>
<td>Request for incidents types: a list of names and ids.</td>
</tr>
<tr>
<td>REQUEST_USERINFO</td>
<td>Terminal / Coordinator</td>
<td>Request for a single (connected) user information: name, picture and current location.</td>
</tr>
<tr>
<td>REQUEST_NFC</td>
<td>Terminal / Coordinator</td>
<td>Request for information about the NFC points: their coordinates and the zones they are associated with.</td>
</tr>
<tr>
<td>REQUEST_POSITION_MONITORING</td>
<td>Coordinator</td>
<td>Request for all the captured locations of a user on a given schedule, i.e. their route.</td>
</tr>
<tr>
<td>REQUEST_ZONES_HASH</td>
<td>Terminal</td>
<td>Request for SHA256 hash of the whole zoning information.</td>
</tr>
<tr>
<td>REQUEST_PHONE_HASH</td>
<td>Terminal</td>
<td>Request for SHA256 hash of the whole phone contacts information.</td>
</tr>
<tr>
<td>REQUEST.ICE_TYPES_HASH</td>
<td>Terminal</td>
<td>Request for SHA256 hash of the whole incident types information.</td>
</tr>
<tr>
<td>REPORT_UPDATE</td>
<td>Terminal</td>
<td>Notification about new checkpoint updated, with the number of schedule item reached and the real arrival time.</td>
</tr>
<tr>
<td>ICE_UPDATE</td>
<td>Terminal</td>
<td>Notification about new incident or event: subject, coordinates, time and description. Pictures or related people information are sent separately with INCIDENT_PICTURE and INCIDENT_PEOPLE respectively.</td>
</tr>
</tbody>
</table>
### Appendix A. KTP Commands

<table>
<thead>
<tr>
<th>Command: INCIDENT_PERSON</th>
<th>Originator: Terminal</th>
<th><strong>Description:</strong> Message attached to ICE_UPDATE describing people related to the incident: name, id, age, nationality, address and motive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command: INCIDENT_PICTURE</td>
<td>Originator: Terminal</td>
<td><strong>Description:</strong> Message attached to ICE_UPDATE with a picture related to the incident.</td>
</tr>
<tr>
<td>Command: SERVER_PING</td>
<td>Originator: Server</td>
<td><strong>Description:</strong> Ping to check if the terminal’s connection is working.</td>
</tr>
<tr>
<td>Command: COORDINATION_LOGIN</td>
<td>Originator: Coordinator</td>
<td><strong>Description:</strong> Login petition by coordinator.</td>
</tr>
<tr>
<td>Command: UPDATE_USERS_POSITION</td>
<td>Originator: Server</td>
<td><strong>Description:</strong> Notification to the coordinators that users have updated their position, with a list of users and their new positions.</td>
</tr>
<tr>
<td>Command: COORDINATION_MESSAGE</td>
<td>Originator: Coordinator</td>
<td><strong>Description:</strong> Message from coordination to be sent to one or more mobile terminals.</td>
</tr>
</tbody>
</table>
Appendix B

Extract of GTD Features

Definition
DATABASE VARIABLES

I. GTD ID and Date

**GTD ID**
*(eventid)*

**Numeric Variable**

Incidents from the GTD follow a 12-digit Event ID system.

- First 8 numbers – date recorded “yyyymmdd”.
- Last 4 numbers – sequential case number for the given day (0001, 0002 etc). This is “0001” unless there is more than one case occurring on the same date.

For example, an incident in the GTD occurring on 25 July 1993 would be numbered as “199307250001”. An additional GTD case recorded for the same day would be “199307250002”. The next GTD case recorded for that day would be “199307250003”, etc.

In rare cases, corrections to the date of a GTD attack are made. In order to maintain stable Event ID numbers, date changes are not reflected in the Event ID.

**Year**
*(iyear)*

**Numeric Variable**

This field contains the year in which the incident occurred. In the case of incident(s) occurring over an extended period, the field will record the year when the incident was initiated.

When the year of the incident is unknown, this will be recorded as “0”.

**Month**
*(imonth)*

**Numeric Variable**

This field contains the number of the month in which the incident occurred. In the case of incident(s) occurring over an extended period, the field will record the month when the incident was initiated.

When the exact month of the incident is unknown, this will be recorded as “0”.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Day</strong></td>
<td><em>iday</em></td>
</tr>
</tbody>
</table>
| Numeric Variable | This field contains the numeric day of the month on which the incident occurred. In the case of incident(s) occurring over an extended period, the field will record the day when the incident was initiated. When the exact day of the incident is unknown, the field is recorded as “0”.

**Approximate Date**  
(approxdate)  
Text Variable
Whenever the exact date of the incident is not known or remains unclear, this field is used to record the approximate date of the incident.

- If the *day* of the incident is not known, then the value for “Day” is “0”. For example, if an incident occurred in June 1978 and the exact day is not known, then the value for the “Day” field is “0” and the value for the “Approximate Date” field is “June 1978”.

- If the month is not known, then the value for the “Month” field is “0”. For example, if an incident occurred in the first half of 1978, and the values for the day and the month are not known, then the value for the “Day” and “Month” fields will both be “0” and the value for the “Approximate Date” field is “first half of 1978”.

**Extended Incident?**  
(extended)  
Categorical Variable

1 = "Yes"  
The duration of an incident extended more than 24 hours.

0 = "No"  
The duration of an incident extended less than 24 hours.

**Date of Extended Incident Resolution**  
(resolution)  
Numeric Date Variable
This field only applies if “Extended Incident?” is “Yes” and records the date in which the incident was resolved (hostages released by perpetrators; hostages killed; successful rescue, etc.)
Date Uncertain?
(dateuncertain)
Categorical Variable

1 = "Yes" The specific date of an incident is not clearly reported.
0 = "No" The specific date of an incident is clearly reported.

II. Incident Information

Incident Summary
(summary)
Text Variable

A narrative summary of the incident, noting the “when, where, who, what, how, and why.”

Note: This field is presently only systematically available with incidents occurring after 1997.

Inclusion Criteria
(crit1, crit2, crit3)
Categorical Variables

These variables record which of the inclusion criteria (in addition to the necessary criteria) are met. This allows users to filter out those incidents whose inclusion was based on a criterion which they believe does not constitute terrorism proper. Note that for each of the criteria variables a case is coded as “1” if source information indicates that the criterion is met and “0” if source information indicates that the criterion is not met or that there is no indication that it is met.

Criterion 1: POLITICAL, ECONOMIC, RELIGIOUS, OR SOCIAL GOAL (CRIT1)

The violent act must be aimed at attaining a political, economic, religious, or social goal. This criterion is not satisfied in those cases where the perpetrator(s) acted out of a pure profit motive or from an idiosyncratic personal motive unconnected with broader societal change.

1 = "Yes" The incident meets Criterion 1.
0 = "No" The incident does not meet Criterion 1 or there is no indication that the incident meets Criterion 1.
Criterion 2: INTENTION TO COERCE, INTIMIDATE OR PUBLICIZE TO LARGER AUDIENCE(S) (CRIT2)

To satisfy this criterion there must be evidence of an intention to coerce, intimidate, or convey some other message to a larger audience (or audiences) than the immediate victims. Such evidence can include (but is not limited to) the following: pre- or post-attack statements by the perpetrator(s), past behavior by the perpetrators, or the particular nature of the target/victim, weapon, or attack type.

1 = "Yes"  The incident meets Criterion 2.
0 = "No"  The incident does not meet Criterion 2 or no indication.

Criterion 3: OUTSIDE INTERNATIONAL HUMANITARIAN LAW (CRIT3)

The action is outside the context of legitimate warfare activities, insofar as it targets non-combatants (i.e. the act must be outside the parameters permitted by international humanitarian law (jus in bello) as reflected in the Additional Protocol to the Geneva Conventions of 12 August 1949 and elsewhere).

1 = "Yes"  The incident meets Criterion 3.
0 = "No"  The incident does not meet Criterion 3.

Doubt Terrorism Proper? (doubterrr)

Categorical Variable

In certain cases there may be some uncertainty whether an incident meets all of the criteria for inclusion. In these ambiguous cases, where there is a strong possibility, but not certainty, that an incident represents an act of terrorism, the incident is included in GTD and is coded as “Yes” for this variable.

1 = "Yes"  There is doubt as to whether the incident is an act of terrorism.
0 = "No"  There is essentially no doubt as to whether the incident is an act of terrorism.

Note: This field is presently only systematically available with incidents occurring after 1997. If this variable was not included in the data collection process at the time the case was coded, “-9” is recorded in the database.

Alternative Designation (alternative)

Categorical Variable
This variable applies to only those cases coded as “Yes” for “Doubt Terrorism Proper?” (above). This variable identifies the most likely categorization of the incident other than terrorism.

1 = Insurgency/Guerilla Action  
2 = Other Crime Type  
3 = Inter/Intra-Group Conflict  
4 = Lack of Intentionality

Note: This field is presently only systematically available with incidents occurring after 1997.

Part of Multiple Incident  
(multiple)  
Categorical Variable

In those cases where several attacks are connected, but where the various actions do not constitute a single incident (either the time of occurrence of incidents or their locations are discontinuous – see Single Incident Determination section above), then “Yes” is selected to denote that the particular attack was part of a “multiple” incident.

1 = "Yes" The attack is part of a multiple incident.  
0 = "No" The attack is not part of a multiple incident.

Related Incidents  
(related)  
Text Variable

When an attack is part of a coordinated, multiple incident the GTD IDs of the related incidents are listed here, separated by commas.

Note: This field is presently only systematically available with incidents occurring after 2011.

III. Incident Location

Country  
(country)  
Categorical Variable
This field identifies the country or location where the incident occurred. This includes non-independent states, dependencies, and territories, such as Northern Ireland and Corsica. If an incident occurs in an autonomous or geographically non-contiguous area, it is listed separately from the “home” country. However, separatist regions, such as Kashmir, Chechnya, South Ossetia, Transnistria, or Republic of Cabinda, are coded as part of the “home” country.

In the case where the country in which an incident occurred cannot be identified, it is coded as “Unknown.”

Note that the political circumstances of many countries have changed over time. In a number of cases, countries that represented the location of terrorist attacks no longer exist; examples include West Germany, the USSR and Yugoslavia. In these cases the country name for the year the event occurred is recorded. As an example, a 1989 attack in Bonn would be recorded as taking place in West Germany (FRG). An identical attack in 1991 would be recorded as taking place in Germany.

Thus, the following change dates apply:

**BREAKUP OF CZECHOSLOVAKIA:**
Czech Republic – independence: 1 January 1993
Slovakia – independence: 1 January 1993

**BREAKUP OF UNION OF SOVIET SOCIALIST REPUBLICS (USSR):**
Armenia – independence: 21 September 1991
Azerbaijan – independence: 30 August 1991
Belarus – independence: 25 August 1991
Estonia – independence: 17 September 1991
Georgia – independence: 9 April 1991
Kazakhstan – independence: 16 December 1991
Kyrgyzstan – independence: 31 August 1991
Latvia – independence: 21 August 1991
Lithuania – independence: 17 September 1991
Moldova – independence: 27 August 1991
Tajikistan – independence: 9 September 1991
Turkmenistan – independence: 27 October 1991
Ukraine – independence: 24 August 1991
Uzbekistan – independence: 1 September 1991


**BREAKUP OF YUGOSLAVIA:**
Bosnia and Herzegovina – independence: 11 April 1992
Croatia – independence: 25 June 1991
Kosovo – UNMIK established: 10 June 1999
Macedonia – independence: 8 September 1991
Yugoslavia becomes Serbia-Montenegro: 4 February 2003
Montenegro – independence: 3 June 2006
Serbia – independence: 3 June 2006
Slovenia – independence: 1 January 1992

**BREAKUP OF CZECHOSLOVAKIA:**
Czech Republic – independence: 1 January 1993
Slovakia – independence: 1 January 1993

**OTHER:**
Eritrea – independence: 24 May 1993
Germany – unification: 3 October 1990
### Country (Location) Codes

(Note: These codes are also used for the target/victim nationality fields. Entries marked with an asterisk (*) only appear as target/victim descriptors in the GTD.)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Afghanistan</td>
<td>46</td>
<td>Comoros</td>
<td>89</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>5</td>
<td>Albania</td>
<td>47</td>
<td>Congo (Brazzaville)</td>
<td>90</td>
<td>Hungary</td>
</tr>
<tr>
<td>6</td>
<td>Algeria</td>
<td>49</td>
<td>Costa Rica</td>
<td>91</td>
<td>Iceland</td>
</tr>
<tr>
<td>7</td>
<td>Andorra</td>
<td>50</td>
<td>Croatia</td>
<td>92</td>
<td>India</td>
</tr>
<tr>
<td>8</td>
<td>Angola</td>
<td>51</td>
<td>Cuba</td>
<td>93</td>
<td>Indonesia</td>
</tr>
<tr>
<td>10</td>
<td>Antigua and Barbuda</td>
<td>53</td>
<td>Cyprus</td>
<td>94</td>
<td>Iran</td>
</tr>
<tr>
<td>11</td>
<td>Argentina</td>
<td>54</td>
<td>Cape Verde</td>
<td>95</td>
<td>Iraq</td>
</tr>
<tr>
<td>12</td>
<td>Armenia</td>
<td>55</td>
<td>Denmark</td>
<td>96</td>
<td>Ireland</td>
</tr>
<tr>
<td>14</td>
<td>Australia</td>
<td>56</td>
<td>Djibouti</td>
<td>97</td>
<td>Israel</td>
</tr>
<tr>
<td>15</td>
<td>Austria</td>
<td>57</td>
<td>Dominica</td>
<td>98</td>
<td>Italy</td>
</tr>
<tr>
<td>16</td>
<td>Azerbaijan</td>
<td>58</td>
<td>Dominican Republic</td>
<td>99</td>
<td>Ivory Coast</td>
</tr>
<tr>
<td>17</td>
<td>Bahamas</td>
<td>59</td>
<td>Ecuador</td>
<td>100</td>
<td>Jamaica</td>
</tr>
<tr>
<td>18</td>
<td>Bahrain</td>
<td>60</td>
<td>Egypt</td>
<td>101</td>
<td>Japan</td>
</tr>
<tr>
<td>19</td>
<td>Bangladesh</td>
<td>61</td>
<td>El Salvador</td>
<td>102</td>
<td>Jordan</td>
</tr>
<tr>
<td>20</td>
<td>Barbados</td>
<td>62</td>
<td>Equatorial Guinea</td>
<td>103</td>
<td>Kazakhstan</td>
</tr>
<tr>
<td>21</td>
<td>Belgium</td>
<td>63</td>
<td>Eritrea</td>
<td>104</td>
<td>Kenya</td>
</tr>
<tr>
<td>22</td>
<td>Belize</td>
<td>64</td>
<td>Estonia</td>
<td>106</td>
<td>Kuwait</td>
</tr>
<tr>
<td>23</td>
<td>Benin</td>
<td>65</td>
<td>Ethiopia</td>
<td>107</td>
<td>Kyrgyzstan</td>
</tr>
<tr>
<td>24</td>
<td>Bermuda</td>
<td>66</td>
<td>Falkland Islands</td>
<td>108</td>
<td>Laos</td>
</tr>
<tr>
<td>25</td>
<td>Bhutan</td>
<td>67</td>
<td>Fiji</td>
<td>109</td>
<td>Latvia</td>
</tr>
<tr>
<td>26</td>
<td>Bolivia</td>
<td>68</td>
<td>Finland</td>
<td>110</td>
<td>Lebanon</td>
</tr>
<tr>
<td>28</td>
<td>Bosnia-Herzegovina</td>
<td>69</td>
<td>France</td>
<td>111</td>
<td>Lesotho</td>
</tr>
<tr>
<td>29</td>
<td>Botswana</td>
<td>70</td>
<td>French Guiana</td>
<td>112</td>
<td>Liberia</td>
</tr>
<tr>
<td>30</td>
<td>Brazil</td>
<td>71</td>
<td>French Polynesia</td>
<td>113</td>
<td>Libya</td>
</tr>
<tr>
<td>31</td>
<td>Brunei</td>
<td>72</td>
<td>Gabon</td>
<td>114</td>
<td>Liechtenstein*</td>
</tr>
<tr>
<td>32</td>
<td>Bulgaria</td>
<td>73</td>
<td>Gambia</td>
<td>115</td>
<td>Lithuania</td>
</tr>
<tr>
<td>33</td>
<td>Burkina Faso</td>
<td>74</td>
<td>Georgia</td>
<td>116</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>34</td>
<td>Burundi</td>
<td>75</td>
<td>Germany</td>
<td>117</td>
<td>Macau</td>
</tr>
<tr>
<td>35</td>
<td>Belarus</td>
<td>76</td>
<td>Ghana</td>
<td>118</td>
<td>Macedonia</td>
</tr>
<tr>
<td>36</td>
<td>Cambodia</td>
<td>77</td>
<td>Gibraltar</td>
<td>119</td>
<td>Madagascar</td>
</tr>
<tr>
<td>37</td>
<td>Cameroon</td>
<td>78</td>
<td>Greece</td>
<td>120</td>
<td>Malawi</td>
</tr>
<tr>
<td>38</td>
<td>Canada</td>
<td>79</td>
<td>Greenland*</td>
<td>121</td>
<td>Malaysia</td>
</tr>
<tr>
<td>40</td>
<td>Cayman Islands</td>
<td>80</td>
<td>Grenada</td>
<td>122</td>
<td>Maldives</td>
</tr>
<tr>
<td>41</td>
<td>Central African Republic</td>
<td>81</td>
<td>Guadeloupe</td>
<td>123</td>
<td>Mali</td>
</tr>
<tr>
<td>42</td>
<td>Chad</td>
<td>83</td>
<td>Guatemala</td>
<td>124</td>
<td>Malta</td>
</tr>
<tr>
<td>43</td>
<td>Chile</td>
<td>84</td>
<td>Guinea</td>
<td>125</td>
<td>Man, Isle of*</td>
</tr>
<tr>
<td>44</td>
<td>China</td>
<td>85</td>
<td>Guinea-Bissau</td>
<td>127</td>
<td>Martinique</td>
</tr>
<tr>
<td>45</td>
<td>Colombia</td>
<td>86</td>
<td>Guyana</td>
<td>128</td>
<td>Mauritania</td>
</tr>
<tr>
<td>46</td>
<td>Comoros</td>
<td>87</td>
<td>Haiti</td>
<td>129</td>
<td>Mauritius</td>
</tr>
<tr>
<td>47</td>
<td>Congo (Brazzaville)</td>
<td>88</td>
<td>Honduras</td>
<td>130</td>
<td>Mexico</td>
</tr>
<tr>
<td>49</td>
<td>Costa Rica</td>
<td>89</td>
<td>Hong Kong</td>
<td>90</td>
<td>Hungary</td>
</tr>
<tr>
<td>91</td>
<td>Iceland</td>
<td>92</td>
<td>India</td>
<td>93</td>
<td>Indonesia</td>
</tr>
<tr>
<td>94</td>
<td>Iran</td>
<td>95</td>
<td>Iraq</td>
<td>96</td>
<td>Ireland</td>
</tr>
<tr>
<td>97</td>
<td>Israel</td>
<td>98</td>
<td>Italy</td>
<td>99</td>
<td>Ivory Coast</td>
</tr>
<tr>
<td>100</td>
<td>Jamaica</td>
<td>101</td>
<td>Japan</td>
<td>102</td>
<td>Jordan</td>
</tr>
<tr>
<td>103</td>
<td>Kazakhstan</td>
<td>104</td>
<td>Kenya</td>
<td>106</td>
<td>Kuwait</td>
</tr>
<tr>
<td>107</td>
<td>Kyrgyzstan</td>
<td>108</td>
<td>Laos</td>
<td>110</td>
<td>Lebanon</td>
</tr>
<tr>
<td>111</td>
<td>Lesotho</td>
<td>112</td>
<td>Liberia</td>
<td>113</td>
<td>Libya</td>
</tr>
<tr>
<td>114</td>
<td>Liechtenstein*</td>
<td>115</td>
<td>Lithuania</td>
<td>116</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>117</td>
<td>Macau</td>
<td>118</td>
<td>Macedonia</td>
<td>119</td>
<td>Madagascar</td>
</tr>
<tr>
<td>120</td>
<td>Malawi</td>
<td>121</td>
<td>Malaysia</td>
<td>122</td>
<td>Maldives</td>
</tr>
<tr>
<td>123</td>
<td>Mali</td>
<td>124</td>
<td>Malta</td>
<td>125</td>
<td>Man, Isle of*</td>
</tr>
<tr>
<td>127</td>
<td>Martinique</td>
<td>128</td>
<td>Mauritania</td>
<td>129</td>
<td>Mauritius</td>
</tr>
<tr>
<td>130</td>
<td>Mexico</td>
<td>131</td>
<td>...</td>
<td>132</td>
<td>...</td>
</tr>
<tr>
<td>Code</td>
<td>Country</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>Moldova</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>Mongolia*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>Morocco</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>Mozambique</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>Myanmar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>Namibia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Nepal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>Netherlands</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>New Zealand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>Nicaragua</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>Niger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>Nigeria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>North Korea</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>Northern Mariana</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>Islands*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>Oman*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>Pakistan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>West Bank and Gaza Strip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>Panama</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>Papua New Guinea</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>Paraguay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>Peru</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>Philippines</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>Poland</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>Portugal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>Puerto Rico</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>Qatar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>Romania</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>Russia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>Rwanda</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>Saudi Arabia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>Senegal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>Serbia-Montenegro</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>Seychelles</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>Sierra Leone</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>Singapore</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>Slovak Republic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>Slovenia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>Solomon Islands</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>Somalia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>South Africa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>South Korea</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>Spain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>Sri Lanka</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>St. Kitts and Nevis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>St. Lucia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>St. Martin*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>Sudan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td>Suriname</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td>Swaziland</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>198</td>
<td>Sweden</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>199</td>
<td>Switzerland</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Syria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>Taiwan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>Tajikistan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>Tanzania</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>Togo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>Thailand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>Trinidad and Tobago</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>Turkey</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>Tunisia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>Turkmenistan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>Uganda</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>Uganda</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>Ukraine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>United Arab Emirates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>Great Britain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>United States</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>Uruguay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>Uzbekistan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>Vanuatu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>Vatican City</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>Venezuela</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>Vietnam</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>Virgin Islands (U.S.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td>Wallis and Futuna</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>Congo (Kinshasa)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>Zambia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>Zimbabwe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td>Northern Ireland</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>Yugoslavia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>Czechoslovakia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>Corsica</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>Somali</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>Asian*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>347</td>
<td>Timor-Leste</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>349</td>
<td>Western Sahara</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>351</td>
<td>Commonwealth of Independent States*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>359</td>
<td>Soviet Union</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>362</td>
<td>West Germany</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>377</td>
<td>North Yemen</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>403</td>
<td>Rhodesia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>406</td>
<td>South Yemen</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>422</td>
<td>International</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>428</td>
<td>South Vietnam</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>499</td>
<td>East Germany</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>520</td>
<td>Sinhalese*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>532</td>
<td>New Hebrides</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>Multinational*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Serbia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Montenegro</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Kosovo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>South Sudan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


List of Figures

2.1 Krimtrack as a black box. ............................................. 6
3.1 Krimtrack subsystems. ................................................. 8
3.2 Comparison of mobile terminal candidates. ...................... 10
3.3 IP grades for solids (top) and liquids (bottom). .............. 11
3.4 NFC tag models feature comparison. ............................. 12
3.5 NFC zone sticker (left) and NFC user authentication keycard (right). 14
3.6 Krimtrack mobile application interface and navigation map. 16
3.7 Mobile application simplified class diagram. .................... 17
3.8 Mobile application SQLite cache database schema. ............ 18
3.9 Communication server simplified class diagram. ............... 20
3.10 Krimtrack database schema. ...................................... 22
3.11 Krimtrack coordination interface. ............................... 23
3.12 Machine learning module simplified class diagram. ......... 26
4.1 A data set example. ................................................... 28
4.2 Supervised learning workflow. .................................... 29
4.3 Example of underfitting (upper left), good fitting (upper right) and overfitting (lower row) of a set of points. .......... 31
4.4 Audiology data set class distribution. ........................... 33
4.5 Column cancer data set class distribution. .................... 34
4.6 Breast cancer data set class distribution. ...................... 35
4.7 Fourier data set class distribution. .............................. 36
4.8 Fourier data set scatter plot of attributes att6 (X-axis) and att73 (Y-axis). 36
4.9 Credit data set class distribution. ............................... 37
4.10 Nursery data set class distribution. ............................. 38
4.11 Imports data set target histogram. .............................. 38
4.12 Geometric example of feature filtering. ....................... 39
4.13 Geometric example of binary transformation of class green. 41
4.14 Geometric example of instance filtering. ....................... 42
4.15 An example of a decision tree used on sample data set seen on figure 4.1. 47
4.16 An example of a decision stump used on sample data set seen on figure 4.1. 51
4.17 An example of 1-NN classification borders. .................... 52
4.18 An example of a neural network. ................................ 53
4.19 The structure of an artificial neuron. .......................... 54
4.20 Examples of neuron activation functions: step functions (left column), bounded linear functions (middle column) and sigmoid functions (logistic function on the upper right and tanh(x) in the lower right). .... 54
4.21 Linear function, bounded linear function and logistic function. . . . . . . . 58
4.22 A diagram of bagging classification. ........................................... 60
4.23 An example of boosting[22]. The first picture represents the first (poor) attempt at classification, but with more iterations (1, 3, 5 and 36 respectively) we can refine the classification. ........................................... 61
4.24 An example of stacking classification with three base classifiers in a data set with classes A, B and C. The instance \((v_1, ..., v_n)\) is classified by all base classifiers and their probability distributions \((p_{ij})\): the probability of class \(j\) according to classifier \(M_i\) form the metainstance that will be the input to the combiner classifier, which will compute the final prediction, B in this example. ........................................... 65
4.25 To train the combiner model we must be sure that the data we input in the base classifiers is not used in the training of said classifiers. By splitting the training set in \(k\) folds (3 in the figure, though 10 is the usual number in practice) and following this scheme we ensure that the metadata set \(D'\) is not biased. ........................................... 66
4.26 Iterative class ensemble classification in a three class data set, in the best case. In the first iteration we discard class B, so we remove all class B instances from the training set and retrain only with instances with class A or C. On the second iteration we discard A so we predict class C. . . . 70
4.27 An example of confusion matrix for the “fourier” data set. Notice how classes 6 and 9 are confused often, as we anticipated. .................. 73
4.28 An example of classification. ........................................... 74
4.29 The suggested cost matrix for the “credit” data set by its author[7]. . . . 75
4.30 An example of ROC curve. ........................................... 76
4.31 Test on training scheme. ........................................... 78
4.32 Holdout validation scheme. ........................................... 78
4.33 Cross-validation (CV) scheme. ........................................... 79
4.34 Bootstrap validation scheme. ........................................... 80
4.35 Grid search for a neural network training. ........................................... 81
5.1 Load testing visualized on coordination interface. .................. 85
5.2 Accuracy results for the audiology data set. ........................................... 87
5.3 Training time results for the audiology data set. ........................................... 88
5.4 Accuracy results for the column data set. ........................................... 89
5.5 Training time results for the column data set. ........................................... 90
5.6 Accuracy results for the breast cancer data set. ........................................... 90
5.7 Training time results for the breast cancer data set. ........................................... 91
5.8 Accuracy results for the fourier data set. ........................................... 92
5.9 Training time results for the fourier data set. ........................................... 92
5.10 Accuracy results for the credit data set. ........................................... 93
5.11 Training time results for the credit data set. ........................................... 94
5.12 Accuracy results for the nursery data set. ........................................... 94
5.13 Training time results for the nursery data set. ........................................... 95
5.14 Results for the naive bayes model. ........................................... 96
5.15 Results for the C4.5 decision tree model. ........................................... 97
5.16 Results for the neural network model. ........................................... 98
5.17 Results for the 3-NN model. ........................................... 99
5.18 Results for the logistic regression model. 99
5.19 One-dimensional logistic functions with increasing coefficients and diminishing error 100
5.20 Results for the random C4.5 forest model. 100
5.21 Results for the AdaBoost.M1 model. 101
5.22 AdaBoost.M1 classification error in test and training on fourier data set. 102
5.23 AdaBoost classification error per class (digit) on fourier data set. 102
5.24 Results for the stacking model. 103
5.25 Results for the class stacking model. 104
5.26 Example of 5-NN classification error due to binary transformation. 105
5.27 A subset of a training set of the combiner in class stacking (fourier set). 105
5.28 Results for the soft class stacking model. 106
5.29 Results for the iterative class ensemble model. 107
5.30 Results for the reinforced class ensemble 1 model. 107
5.31 Results for the reinforced class ensemble 2 model. 108
5.32 Final classifier accuracy ranking. 109
5.33 Final classifier ranking by training time. 110
5.34 Results for the imports data set. 111
5.35 Data reduction for CFS filtering. 112
5.36 Results for feature filtering. 113
5.37 Results for FCNN filtering. 114
5.38 Results for instance filtering. 114
5.39 Results for holdout training percentages for audiology data set. 116
5.40 Results for holdout training percentages for breast cancer data set. 116
5.41 Results for holdout training percentages for fourier data set. 117
5.42 Results for holdout training percentages for credit data set. 117
5.43 Results for cross-validation number of folds for audiology data set. 118
5.44 Results for credit data set, with cost matrix adjustment. 119
5.45 ROC analysis for logistic regression on fourier set. 120
5.46 Model selection performance for k-NN. 122
5.47 Histogram for the incident data generated. 124
5.48 Coordination interface with incident risk prediction. 125

6.1 Target type distribution for the GTD. 130
6.2 Partial C4.5 decision tree for the GTD. 131
6.3 Chronological evolution of the GTD analysis performed. 132
List of Tables

5.1 Naive bayes results .................................................. 96
5.2 C4.5 decision tree results ......................................... 97
5.3 Neural network results .............................................. 97
5.4 k-NN results .......................................................... 98
5.5 Logistic regression results ......................................... 99
5.6 Random C4.5 forest results ....................................... 100
5.7 AdaBoost.M1 results .............................................. 101
5.8 Stacking results ...................................................... 103
5.9 Class stacking results ............................................. 104
5.10 Soft class stacking results ....................................... 106
5.11 Iterative class ensemble results ................................. 106
5.12 Reinforced class ensemble 1 results ............................ 107
5.13 Reinforced class ensemble 2 results ............................ 108
5.14 Overall classifier accuracy ranking ............................. 109
5.15 Overall classifier training time ranking ......................... 110
5.16 Results for imports data set ..................................... 111
5.17 CFS feature reduction ........................................... 113
5.18 Confusion matrix of a k-NN classification for the credit data set ........................................... 119
5.19 Precision and recall comparison for binary classification of fourier set using optimal thresholds ................................. 121