UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DEL GRAU EN ENGINYERIA FÍSICA

Active Learning in Crowdsourcing Classification Problems using Sampling Theory for Graph Signals

---

Author:
Javier MAROTO

Director:
Dr. Antonio ORTEGA

Tutor:
Dr. Ferran MARQUES

May 15, 2017

Universitat Politècnica de Catalunya

Abstract

Engineering Physics

Active Learning in Crowdsourcing Classification Problems using Sampling Theory for Graph Signals

by Javier MAROTO

Before state of the art methods can be deployed to solve a classification problem, the first step is to collect and label a sufficient amount of training data. Given the time and cost associated to data labeling, crowdsourcing systems (e.g., Amazon Mechanical Turk) are often used. However, one of the key disadvantages of crowdsourcing systems is the presence of spammers or workers who are not as skilled or careful, thus leading to many false labels being assigned. This work addresses this problem by proposing novel algorithms that optimally assign data to different workers for labeling, taking into account the expected quality of labeling provided by each worker. Regarding the model used to classify and select our nodes, we propose a new methodology that is based on graph signal sampling theory for active learning problems. Our simulation of the labeling process using these schemes shows that the classification error can be reduced with respect to a random assignment of workers. We also give some insights about the advantages of using one or multiple labels per data point for labeling in crowdsourcing systems.

Contents

Chapter 1

Introduction

1.1 Motivation: crowd-sourcing labeling

With the development of the field of machine learning [1] and the recent increase in popularity of deep learning [2], there has been a lot of focus on how to improve methods to find patterns from complex data and use them to classify or predict behaviors in new data. Using these methods is commonly referred in machine learning as learning from data. However, what is always assumed in the majority of the literature is that we already start with some meaningful data. In practice, we must invest heavily in human resources in order to label data needed to learn the model parameters.

Take as an example the following classification problem [3]. We have a huge dataset with images of human faces. We want to know for every image if the person is a male or a female. Using only manual classification is unfeasible for huge datasets. Classifying using image processing by explicitly processing some patterns of the image, such as shape of the face or length of hair, proves to be excessively difficult, not robust and inefficient. The general approach is to use machine learning, which requires some previous manual labeling to create our training set, i.e., labeling a subset of images as either male or female.

In this work, our principal motivation is to reduce the cost and increase the efficiency of manual labeling for active learning semi-supervised classification problems. The cost of labeling, measured in terms of time or human resources, can become prohibitive given the increased sizes of datasets used for training. This could be a major bottleneck in deploying machine learning solutions. Semi-supervised learning can help reduce the burden of labeling, by using both labeled and unlabeled data. A key problem in semi-supervised learning is that of determining which items to label, given limited resources. Substantial progress has been made in this area with a focus on active learning, where the main objective is to select for labeling only those data points that best characterize the dataset. How to select which points to label without wasting resources labeling points that provide little to no information is one of the key questions of this study. There are state of the art studies [4, 5] including the one in which this work is inspired [6] that address this selection process and try to improve the class prediction error with the minimum number of labels possible. Optimality can be defined by quantifying the error in estimating the label for data that has not been manually labeled.

However, one assumption made in those works is that there are no errors in the labeling. This assumption is fair when the labeling is done by experts [7], people that invest abundant time doing the task [8] or when tasks are easy. But in practice, this is rarely the case. There are several important factors we have to take into account when using human resources to label a dataset, including the financial cost of labeling, the probability of having mistakes in the labels and the time required for the process.

Additional factors to address when trying to label a class that is subjective are the honesty of the labeler, the diversity of the group of labelers or the wording of the questions, which may influence the answer of the labeler [9].

Clearly, it would be better to hire highly motivated and expert individuals to minimize the number of errors. But it is easy to see that hiring experts or supervising people to ensure they are focused on the task not only will be expensive, but also time-consuming. This is why nowadays a common approach is to use crowd-sourcing platforms such as Amazon's Mechanical Turk [10] or Crowdflower [11]. Not only this approach is much cheaper, but the labeling quality is not much worse [9]. There are more advantages when the classification problem is subjective. First, the necessity of making a standardized process allows no bias from the people that request the tasks. Second, because each worker is anonymous and they are not supervised, they are able to give more honest answers [12]. However, because of this anonymity, we have to also be aware of the possibility of having some low-performing workers, or spammers, that could be a important source of error [13].

Our objective is to extend the work in [6] by no longer making the assumption that workers don't make mistakes when labeling. This extension is important due to the increased use of crowdsourcing. Our results are promising and leverage recent advances in Graph Signal Processing (GSP) for datapoint selection. We will elaborate in some of the concepts presented in that study [6], that we will use as base of this work. Furthermore, We will explain all the new methodology and algorithms designed in our study to adapt the previous proposed ideal scheme to the real crowd-sourcing labeling.

There are multiple approaches that minimize the effect of adding error to the labels of the training set. Some of the most popular are using multiple noisy labels for each data point [14] or adding a reduced group of experts to check some data points [15]. In general, the common approach is to add some sort of redundancy to improve the prediction error. However, these studies do not harness the advantage of using active learning to efficiently label and the fact that labels are similar across points that are neighbors in the feature space (cluster-like behavior), as our work does.

One additional idea that our study challenges in Chapter 3 is the assumption most literature makes: that changing the assignment order of the workers does not yield any improvement at all. To the best of our knowledge there is no work that considers the situation proposed in Problem 2 (which will be stated in Chapter 2) using GSP-based algorithms.

1.2 Objectives

The main goal of this project is to design new and better methodologies, based on GSP, that addresses the challenge of having wrong labels in active learning. The hypothesis is that taking into account the quality of work of each labeler, there are ways of assigning them samples that reduce the impact of spammers or low-quality workers. We will show algorithms that use vertex-domain consistency as a way to reduce the error after unreliable labeling. This has not been used in any other work to our knowledge. Our goal is to give insight about the whole process of labeling and describe it analytically.

Hence, the objectives pursued in this project are:

1. Deduce how to optimally distribute the work to different performing workers so that the prediction error is minimized.

2. Deduce when it is better to re-label already labeled samples (i.e, increase labeling redundacy) instead of continuing labeling unlabeled ones.

3. Estimate the error probability associated to the labeling decisions in the samples and the performance in the workers

4. Develop software in order to apply all the insights obtained and simulate the crowd-sourcing situation

5. Design relevant heuristics for a realistic scenario, which will be defined in the next chapter. We will simplify aspects like the dataset target behavior or the probability error of the workers.

1.3 Semi-supervised Learning

Supervised learning allows us to infer the relation between the features and the target from the training set and then use the learned function to predict the target of new data points from their features. Unsupervised learning gives us information of the distribution of data points, so there is no distinction between training and test data. We just classify all the information given but we cannot do any error evaluation or inference of any target from the data. Semi-supervised learning [16] is a special case that mixes the previous two types of learning. We usually apply semi-supervised learning methods when we have little labeled data. In this situation, we cannot learn well the underlying function that relates the features with the target, so instead of using a bad estimate of that function we group all data points by feature similarity as we would do in unsupervised learning and infer the unlabeled points from the few labeled points within the structure (see Figure 1.1).

In our classification problem we start with no labeled data points. Although we will label part of the data using noisy workers, we assume that we have a very limited budget, so during all the labeling process the number of labeled samples will be much lower than the number of unlabeled ones. Thus, our problem will use semi-supervised methods.

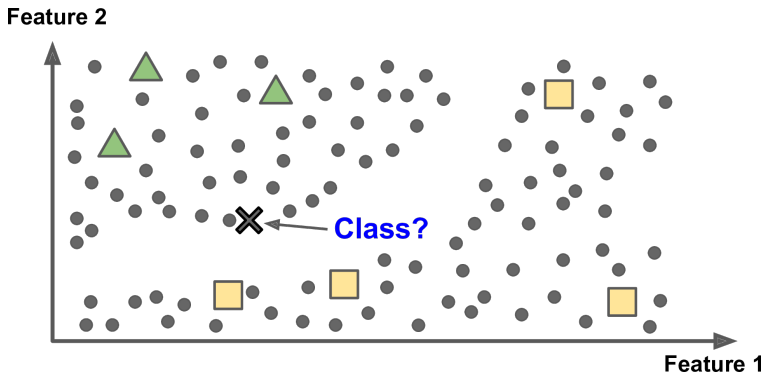


FIGURE 1.1: In this example we can see the importance of semi-supervised learning when there are few samples labeled. The triangular and square figures represent labeled data points that belong to one of two classes so they are part of the training set. The circular points are unlabeled points. They are part of the test set. If we were to use a supervised method we would predict that the marked point is of the square class because it is nearer to the square points than to the triangle points. However, using a semi-supervised method we know that it should belong to the triangular class because is on the same cluster as the triangular class points.

1.4 Active learning

When we have few or no examples to train the system, that is, we are in a unsupervised/semi-supervised regime, we normally cannot make good enough inferences because we lack knowledge of the function that relates features with target. If we want to further improve the quality of our inferences, we need more labeled samples. We can increment the number of labeling points by using pasive or active learning.

The difference between passive and active learning is that in the latter we choose what points we want labeled. The principal objective when using active learning is to add labels so that the prediction error in all the data points that we leave unlabeled is minimized. It is similar to a teacher solving exercises, and having to solve the most relevant exercises for the course because of lack of time. In active learning, the limiting factor is the cost of manually having to create examples for the machine. One added advantage of active learning is that because we get the same quantity of information as in passive learning but with fewer examples, generally we can obtain the same results but using a smaller training set. This allows us to use more complex methods to train because we will not be as limited by the execution time, which is dependent on the number of samples of the training set.

Some of the query strategies to optimally choose data points to add to the training set include uncertainty sampling [17], margin sampling [18], query-by-committee [19, 20], expected gradient length [21] or using Gaussian

Random Fields [22]. Instead, in our work we base our active learning strategy in Graph Signal Processing principles (see Section 2.3), but other active learning frameworks can be applied.

Chapter 2

Problem formulation

2.1 Definition of the problem

In our work, we ultimately want to propose an algorithm capable of assigning workers efficiently in crowdsourcing systems taking into account possible errors when labeling. In order to simplify the task we divide the original problem in two separate problems.

Problem 1 (Single-label problem) *Consider the following situation. We have an unlabeled dataset of size N, which has associated a graph $\mathcal{G}$ to be defined later. We have a limited pool of workers of size W. Each worker can label only one of the data points. Each worker, i, has a known probability ε_i of making a mistake when labeling. We select a subset of data points S with size W using the algorithm in [6]. This subset defines the set of data points that will be labeled. Each sample of S can only be labeled one time. Each The problem is choosing where to assign each worker so that when interpolating the rest of the dataset from the labeled set S the error is minimized.*

In Problem 1, we have simplified the real situation by imposing that any data point can only be labeled one time. Using this constraint, we can try algorithms that measure the different impact that errors in the labeling have depending on where we assign the noisier workers. Because we cannot label more than one time, the number of nodes labeled will be constant for a given number of workers. This simplifies greatly the problem as we do not have to worry about harder questions, e.g, when we should stop labeling new nodes and start adding redundancy.

In Chapter 3, we will develop two new algorithms to solve Problem 1. The first algorithm is the *covariance dispersion algorithm* (CDA), which is obtained from analyzing the conditional covariance matrix ($\mathbf{K}_{S^C|S}$). The second algorithm is the *node contribution algorithm* (NCA), which is obtained from interpreting the MAP estimator ($\boldsymbol{\mu}_{S^C|S}$) as a random walk process.

Problem 2 (Multi-label problem) *Same as Problem 1, but each sample of S can receive any number of labels.*

Problem 2 corresponds to a more realistic situation, in which we can make use of redundancy to reduce the error in certain data points. Because we can have multiple labels in the same data point, the number of labeled nodes can be less than the number of workers. In this case we have to take into account

that the sample class probability is not directly expressed by the quality of worker assigned to it as in Problem 1, but it is a function of the quality of each of the workers assigned to the sample and the class of each label given. This adds difficulty to the problem because the optimization will be dependent on the sample probabilities and not the worker probabilities.

In Chapter 4 we will see that this new degree of freedom will allow us to lower the prediction error below what can be achieved under Problem 1. We will adapt the *node contribution algorithm* for this new scheme and add several improvements to it. We will show in Chapter 5 that using this algorithm is much better than using other known common algorithms such as uniform labeling or adaptively labeling the noisier samples.

In both Problem 1 and Problem 2, we distinguish two different cases, binary and multiclass classification. In binary classification we will consider that the labels can only have two values: $\{+1, -1\}$. In multiclass classification we will consider that each label class is encoded as a one-hot vector. In case of labeling error, the label given could be any of the erroneous classes with equal probability.

2.2 Notation and preliminaries

We will start giving some notation used in the field of Graph Signal Processing (GSP) [23, 24]. The use of GSP in semi-supervised learning is not new, has already led to promising results [25, 26, 27]. The graph reflects the underlying feature and target distribution of a data set. Each node is a data point, and the edges represent similarity between data points. Each node has a value associated with it, which for classification problems is a categorical value. These values constitute the graph signal. A graph signal is *smooth* or low-pass if moving from one node to its neighbors, the graph signal is unlikely to change. This is analogous to saying that, when doing clustering in semi-supervised methods, data points from the same cluster are likely to belong to have the same class.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected, undirected and weighted graph of size N, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. If two nodes share a lot of categorical features or have numerical features with close values, the edge between them will have a high weight. In the opposite case, the edge will have low or zero weight (no edge). A typical example of this would be the Gaussian kernel weights, which are defined as $w_{ij} = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2)$, where $\mathbf{x}_i$, $\mathbf{x}_j$ are the feature vectors of data points i and j. It is easy to verify that if $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar their edge weight w_{ij} will be high.

The adjacency matrix $\mathbf{W}$ is given by the weights of the edges between the nodes of the graph. That is, we have that $\mathbf{W}_{ij} = w_{ij}$ where w_{ij} is the weight of the edge that connects i and j. We also know that $\mathbf{W} = \mathbf{W}^\top$ because the graph is undirected and that $\mathbf{W}_{ii} = 0$ because there are no self-loops. The degree matrix $\mathbf{D}$ is a diagonal matrix where $\mathbf{D}_{ii} = \sum_{j=1}^{N} w_{ij}$. The expression of the Laplacian matrix is $\mathbf{L} = \mathbf{D} - \mathbf{W}$. The Laplacian matrix can be shown to be positive and semi-definite. Hence, it has real eigenvalues $0 = \lambda_1 < \lambda_2 \leq ... \leq$

λ_N with corresponding set of eigenvectors $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N\}$ and can be written as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N)$ and $\mathbf{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_N)$.

The sets of eigenvalues and eigenvectors of the Laplacian matrix gives us a notion of frequency in the spectral graph domain. The eigenvalues would be related to the frequency each of the eigenvectors has. Eigenvectors associated with eigenvalues with low value, have similar values between nodes that are strongly connected and vice versa. Thus, in GSP, low frequency is related with closeness of values in neighboring nodes and high frequency would be equivalent to having very different values in strongly connected nodes. Any graph signal, can be projected into the base formed by the eigenvectors of the Laplacian and the resulting coefficients correspond to the spectral components of the signal. This projection operation is similar to the Fourier Transform, so in this field it is called Graph Fourier Transform or GFT.

2.3 Sampling theory using GSP

This subsection serves as a reference for the two active learning algorithms used in our experimental results (Chapter 5), that will be explained in more detail in Section 5.2.

Algorithm 1 has been used in [6]. It tries to maximize the cut-off frequency ω of the Paley-Wiener space of perfectly recoverable graph signals in the graph $\mathcal{G}$ that represents our dataset: $\mathbf{f} \in PW_\omega(\mathcal{G})$. This is analogous to maximizing the Nyquist frequency when sampling in traditional signal processing.

Algorithm 2 is a novel method of our work based on the k-medoids method [28], but applied to graphs. It chooses cluster representatives so that we maximize their similarity with the rest of the data points. These algorithms are unrelated to the ground of our work, but we use them to select the subset of representative nodes S that the algorithms we explain in Chapters 3 and 4 will require.

Algorithm 1 Greedy heuristic based on graph sampling theory

Input: Adjacency matrix $\mathbf{W}$
Output: Optimal subsets of n nodes S_n where $n = 1...N$, cut-off frequencies of each subset ω_n

1: $S \leftarrow \{\emptyset\}$
2: $\mathbf{L} \leftarrow obtainLaplacianMatrix(\mathbf{W})$
3: $\mathbf{L}^{(K)} \leftarrow \mathbf{L}^K$
4: **for** $i = 1 : N$ **do**
5: $\quad [\omega_i, \mathbf{v}] \leftarrow smallestEigenpair(\mathbf{L}_{SC}^{(K)})$
6: $\quad \text{nextNode} \leftarrow indexMaxValue(\mathbf{v})$
7: $\quad S \leftarrow addNode(S, \text{nextNode})$
8: $\quad S_i \leftarrow S$
9: **end for**

Time Complexity: $O((|\mathcal{V}| + 2|\mathcal{E}|)KN)$

Algorithm 2 Greedy heuristic based on k-medoids clustering

Input: Adjacency matrix **W**
Output: Optimal subsets of n nodes S_n where $n = 1...N$

```
S ← {∅}
D ← graphAllShortestPaths(W)
for i = 1 : N do
    v ← meanDistanceToEachNode(D)
    nextNode ← indexMinValue(v)
    S ← addNode(S, nextNode)
    S_i ← S
    D ← updateDistances(D, nextNode)
end for
```

Time Complexity: $O(|\mathcal{V}|log|\mathcal{V}| + |\mathcal{V}||\mathcal{E}|)$

2.4 Gaussian Random Field assumption

Let S be the subset of nodes chosen to be labeled by workers. This set will be obtained from the active learning algorithm. Let $S^C = \mathcal{V} \setminus S$ be its complementary set in the graph. Our main question is: to which node $i \in S$ should we assign a noisy worker so that, if the worker makes any mistakes labeling, it has the minimum possible impact in the prediction? To answer that, we will assume that, although the graph signal components l_i are categorical values, they are generated by hard thresholding from an underlying Gaussian graph signal f_i (for example, in face recognition there are faces that are more probable to belong to female class than others but both have the same categorical value). In binary classification the class labels of the target are obtained from the sign of the underlying Gaussian graph signal, while in multiclass classification the class is given by the component with greater value. We will see next that for any labeling in S, we can express the distribution of the graph signal on S^C, conditional on the observation, as a multivariate Gaussian density function, where its mean gives us the best estimation of the interpolated signal.

Consider $\mathbf{f} = (f_1, f_2, ..., f_N)^\top$ a graph signal generated by a GRF. For the sake of simplicity, let us assume binary classification and the prior $E\{f_i\} = 0$. The probability of obtaining a given graph signal will then be proportional to:

$$p(\mathbf{f}) \propto e^{-\mathbf{f}^\top(\mathbf{L}+\delta\mathbf{I})\mathbf{f}} = e^{-\mathbf{f}^\top\mathbf{K}^{-1}\mathbf{f}} \tag{2.1}$$

In the general case $E\{f_i\} \neq 0$, we would have to make the substitution $f_i \longrightarrow (f_i - E\{f_i\})$ in (2.1). In multiclass classification, we would have a GRF generated signal for each of the classes.

In GRF, signals that have most of its energy in the low frequency part of the spectrum are more probable, which can be verified by expressing the Laplacian in (2.1) as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$. This emulates the cluster-like behavior typical in machine learning problems.

Like in any other multivariate Gaussian process, here the notion of covariance matrix is given by $\mathbf{K} = (\mathbf{L} + \delta\mathbf{I})^{-1}$ where $\mathbf{I}$ is a identity matrix of size N x N and δ is a real positive with an arbitrarily small value. It is easy to see that the covariance matrix $\mathbf{K}$ has the same eigenvectors as the Laplacian matrix $\mathbf{L}$ but their corresponding eigenvalues are $\sigma_i = 1/(\lambda_i + \delta)$. Here we can see the importance of adding the $\delta\mathbf{I}$ term in the covariance matrix expression as it avoids having a singular σ_1 eigenvalue ($\lambda_1 = 0$). Henceforth, we can express the covariance matrix as:

$$\mathbf{K} = \sum_{i=1}^{N} \frac{1}{\lambda_i + \delta} \mathbf{u}_i \mathbf{u}_i^T = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top \tag{2.2}$$

where $\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, ..., \sigma_N)$. $\mathbf{K}$ can be written as in (2.2) because $\mathbf{L}$ is a positive semi-definite, real and symmetric matrix, so that $\mathbf{K}$ and $\mathbf{K}^{-1}$ will be positive definite, real and symmetric matrices, and thus, by the spectral theorem they are diagonalizable.

Let us select a subset of nodes $S \subset \mathcal{V}$. By simple permutation we can write without loss of generality $\mathbf{f} = [\mathbf{f}_S^\top \ \mathbf{f}_{S^C}^\top]^\top$, where $\mathbf{f}_S$ is the graph signal corresponding to only the vertices in S. It is well known that, given $E\{f_i\} = 0$, the conditional distribution of $\mathbf{f}_{S^C}$ given $\mathbf{f}_S$ will be another multivariate Gaussian [29, 30] with mean $\boldsymbol{\mu}_{S^C|S}$ and covariance matrix $\mathbf{K}_{S^C|S}$ such that:

$$\boldsymbol{\mu}_{S^C|S} = \mathbf{K}_{S^C S}(\mathbf{K}_S)^+ \mathbf{f}_S \tag{2.3}$$

$$\mathbf{K}_{S^C|S} = \mathbf{K}_{S^C} - \mathbf{K}_{S^C S}(\mathbf{K}_S)^+ \mathbf{K}_{SS^C} \tag{2.4}$$

where $\mathbf{K}_{S^C S}$ denotes the sub-matrix of K with rows indexed by S^C and columns indexed by S and $(\mathbf{K}_S)^+$ is the Moore-Penrose pseudoinverse of $\mathbf{K}_S$. For simplicity in the notation, $\mathbf{K}_S$ is equivalent to $\mathbf{K}_{SS}$. For the general case $E\{f_i\} \neq 0$, we just have to add a bias term to the mean $\boldsymbol{\mu}_{S^C|S}$ and to the graph signal $\mathbf{f}_S$.

This model gives us a probabilistic interpretation of the interpolated labels. The influence the subset of nodes chosen has on the predicted classes and the uncertainty of each prediction, which is given by the covariance matrix, is well defined by this model. Each prediction is given as a Gaussian distribution, where $sign(\boldsymbol{\mu}_{S^C|S})$ gives us the most probable combination of positive or negative class labels for S^C. To calculate the exact probability, we have to represent the multivariate conditional Gaussian distribution in the $|S^C|$ dimensional space. The integral of the multivariate conditional density function in one of the $2^{|S^C|}$ quadrants gives us the probability that the corresponding particular combination of class labels is correct.

Chapter 3

Single-label problem algorithms

3.1 Covariance dispersion algorithm

In this section we present the CDA, that we will use to solve Problem 1, for binary classification. For multiclass classification we would have to account for biases and apply the algorithm to each of the classes, which complicates the formulation. For the sake of simplicity in the notation we will show the CDA only for binary classification.

We will see that the CDA sorts the nodes of a given set of nodes S by the impact that an error has in the prediction of S^C. We will use the relative entropy to measure the impact of an error. This will allow us to determine the optimal labeling probability error for each worker so that the prediction error is minimized, for a given mean labeling error of the workers.

From Section 2.4 we know that for any graph signal $\mathbf{f}$ that has been generated by a GRF, $\mathbf{f}_{S^C|S}$ will be a multivariate Gaussian random variable with mean and covariance matrix given by (2.3) and (2.4). We face two problems because of the nature of labeling. First, the graph signal obtained from labeling the samples of S is not $\mathbf{f}_S$, but $\boldsymbol{l}_S = \text{sign}(\mathbf{f}_S)$. Second, the predicted graph signal $\mathbf{f}_{S^C}$ has to be thresholded. That is, we have to make a hard decision of the class of each sample of S^C from the graph signal obtained $\mathbf{f}_{S^C}$. The prediction error will be assessed based on $\boldsymbol{l}_{S^C}$, rather than $\mathbf{f}_{S^C}$.

We are going to make some assumptions in order to use the GRF model.

- We will assume that in reality the data points do not have a real hard value. In a sense, when we say a data point is of one class, we just are more confident that the data point is of that class than the others. Although there are points that can be easily classified, there are some others that have some uncertainty and for which even non noisy workers could strongly disagree. For example, this kind of situation may arise when trying to diagnose some types of cancer.

- We will assume that replacing $\mathbf{f}_S$ by $\boldsymbol{l}_S$ will just add more variance to the MAP estimation, specially at the nodes of the graph where the graph signal has values close to zero. However, the location of the near-zero points or frontiers depends on the frequency of the graph signal. Because we have no prior knowledge of the graph signal frequency we cannot optimize this error, which is independent of the parameters we use to optimize. It is important to point out that although we add variance by

using $\boldsymbol{l}$ instead of $\mathbf{f}$, obtaining $\boldsymbol{l}_{S^C}$ from $\mathbf{f}_{S^C}$ does not add error. That is, if we know $\mathbf{f}$ we know $\boldsymbol{l}$ but not vice versa.

Now, we will use the GRF model to assess the impact on the prediction of the subset S^C of errors on samples of S. First, we will use the second assumption above and substitute $\mathbf{f}_S$ by $\boldsymbol{l}_S$ in (2.3). The resulting conditional multivariate normal distribution will be given by:

$$\boldsymbol{\mu}_{S^C|S} = \mathbf{Q}_{S^C S}\boldsymbol{l}_S \tag{3.1}$$

$$\mathbf{K}_{S^C|S} = \mathbf{K}_{S^C} - \mathbf{Q}_{S^C S}\mathbf{K}_{SS^C} \tag{3.2}$$

where we define $\mathbf{Q}_{S^C S} = \mathbf{K}_{S^C S}(\mathbf{K}_S)^+$.

As we can see, the covariance matrix of the prediction is invariant to any errors, because it does not depend on the values of the labeling graph signal. However, we see that an error on a sample $i \in S$ is equivalent to changing the sign of l_i. This results on a shift of the mean of the multivariate normal distribution of value:

$$\boldsymbol{\epsilon}_{S^C|i} = -2\mathbf{q}_{S^C i}l_i \tag{3.3}$$

where $\mathbf{q}_{S^C i}$ is the column $i \in S$ of the matrix $\mathbf{Q}_{S^C S}$ and l_i is the true label of the node i.

We know that, if there are no errors in the labeling, $\boldsymbol{\mu}_{S^C|S}$ is the MAP estimation of $\mathbf{f}_{S^C}$ given $\mathbf{f}_S$. That is, it is the best estimation possible. Our objective will be to chose in which samples we would want to allocate the noisier workers so that the resulting shifted distribution is as close as possible to the best estimation. We will measure the closeness of the shifted distribution by calculating the Kullback-Leibler divergence (KL or relative entropy) between the shifted and original multivariate normal distributions.

Let $\tilde{\boldsymbol{l}}_S = \boldsymbol{l}_S - 2\boldsymbol{\phi}_S \cdot \boldsymbol{l}_S$, be the noisy labels. $\boldsymbol{\phi}_S$ is a column vector in which each of its components is a random variable that has Bernoulli distribution valued 1 with probability ε_i and 0 with probability $(1 - \varepsilon_i)$ with $i \subset S$, where ε_i is the probability that the worker assigned to sample i makes a mistake when labeling. The mean of the shifted distribution will be then:

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_{S^C|S} &= \mathbf{Q}_{S^C S}\tilde{\boldsymbol{l}}_S \\ &= \mathbf{Q}_{S^C S}\boldsymbol{l}_S \cdot (\mathbf{1}_S - 2\boldsymbol{\phi}_S)\end{aligned} \tag{3.4}$$

Now that we have defined the mean of the original and shifted distributions, we can measure the relative entropy between them, keeping in mind that their covariances are the same. The KL divergence of two multivariate normal distributions that share the same covariance matrix is:

$$\begin{aligned}D_{KL}(\boldsymbol{f}_{S^C|S}||\tilde{\boldsymbol{f}}_{S^C|S}) &= \frac{1}{2}(\tilde{\boldsymbol{\mu}}_{S^C|S} - \boldsymbol{\mu}_{S^C|S})^\top \mathbf{K}^{-1}_{S^C|S}(\tilde{\boldsymbol{\mu}}_{S^C|S} - \boldsymbol{\mu}_{S^C|S}) \\ &= (\mathbf{Q}_{S^C S}\boldsymbol{l}_S \cdot \boldsymbol{\phi}_S)^\top \mathbf{K}^{-1}_{S^C|S}\mathbf{Q}_{S^C S}\boldsymbol{l}_S \cdot \boldsymbol{\phi}_S\end{aligned} \tag{3.5}$$

We could further simplify this expression if we diagonalize $\mathbf{K}_{S^C|S}$. Let us go back to the equation (2.4). Because both terms of the right side are real and symmetric, we know that $\mathbf{K}_{S^C|S}$ will also be real and symmetric, and

by the spectral theorem it will be diagonalizable. Thus we can express the conditional covariance matrix as:

$$\mathbf{K}_{S^C|S} = \mathbf{U}_c \mathbf{\Sigma}_c \mathbf{U}_c^\top \tag{3.6}$$

where $\mathbf{U}_c$ and $\mathbf{\Sigma}_c$ are the corresponding eigenvector and eigenvalue matrices, respectively. Using terminology specific to the context of multivariate normal distributions, $\mathbf{U}_c$ contains the vectors of principal axes of the multivariate normal distribution and $\mathbf{\Sigma}_c$ contains the variance in the directions of each one of the principal axes. Because $\mathbf{K}_{S^C|S}$ is an Hermitian matrix, $\mathbf{U}_c$ is an orthogonal matrix.

Substituting the expression in (3.6) to the equation (3.5), we can further simplify the KL divergence as:

$$D_{KL}(\boldsymbol{f}_{S^C|S} || \tilde{\boldsymbol{f}}_{S^C|S}) = |\mathbf{\Sigma}_c^{-1/2} \mathbf{U}_c \mathbf{Q}_{S^C S} \boldsymbol{\phi}_S|^2 \tag{3.7}$$

where we used the fact that $(\boldsymbol{l}_S \cdot \boldsymbol{\phi}_S)^2 = \boldsymbol{\phi}_S^2$ and that $\mathbf{U}_c^\top = \mathbf{U}_c^{-1}$.

As mentioned earlier, trying to minimize the KL divergence is equivalent to minimizing the prediction error. To simplify the process, we minimize the square root of the KL divergence, in order to avoid having a squared vector in the expression. Furthermore, we will average the expression so that we can express it in terms of the labeling error probabilities and not the Bernoulli distributions. Then, instead of (3.7), we minimize:

$$E\left\{\sqrt{D_{KL}(\boldsymbol{f}_{S^C|S} || \tilde{\boldsymbol{f}}_{S^C|S})}\right\} = |\mathbf{\Sigma}_c^{-1/2} \mathbf{U}_c \mathbf{Q}_{S^C S} \boldsymbol{\varepsilon}_S| \tag{3.8}$$

where $\boldsymbol{\varepsilon}_S$ is the vector of error probabilities associated to each labeled sample of S, which depends on the worker assignment.

Trying to minimize the mean KL divergence or the mean square-rooted KL divergence over all possible worker assignments is unfeasible computationally, because we would have to check each of the $|S|!$ possible assignments and choose the one with least KL divergence. We could potentially use some heuristics in (3.8). For example, we could choose to apply the following inequality $|\mathbf{\Sigma}_c^{-1/2} \mathbf{U}_c \mathbf{Q}_{S^C S} \boldsymbol{\varepsilon}_S| \leq \sum_{j \subset S} |\mathbf{\Sigma}_c^{-1/2} \mathbf{U}_c \mathbf{Q}_{S^C j}| \varepsilon_j$ and minimize the right side, which is much easier than minimizing the original problem.

As an alternative, we relax the problem so that instead of having that each of the ε_i corresponds to a chosen worker from a known set of workers of size $|S|$, we allow any labeling error probability to be chosen. This means that we are not limited to evaluating the KL divergence on a discrete set of points given by all the different assignments of the workers.

It is trivial to see that without any constraint the optimal labeling error vector would be $\boldsymbol{\varepsilon}_S = \mathbf{0}_S$. Instead, we study what would be the optimal in terms of minimizing the KL divergence cost $\boldsymbol{\varepsilon}_S$ with the constraint $E\{\boldsymbol{\varepsilon}_S\} = \bar{\varepsilon}$, i.e., we constraint the average error probability to be given. This relaxation is useful empirically to see the validity of the assumptions we made at the start of this section, as we will see when we present the simulations and results in a later chapter. To solve it, we will use the following constrained linear

least-squares problem:

$$\begin{aligned} &\min_{\varepsilon_i} \mathbf{A}\boldsymbol{\varepsilon}_S \;\rightarrow\; \mathbf{A}\boldsymbol{\varepsilon}_S = \mathbf{0}_{S^C} \\ &\text{subject to } \mathbf{I}_S\boldsymbol{\varepsilon}_S = \bar{\varepsilon}|S|,\ (\varepsilon_i, \bar{\varepsilon}) \in [0,1] \end{aligned} \tag{3.9}$$

where $\mathbf{A} = \boldsymbol{\Sigma}_c^{-1/2}\mathbf{U}_c\mathbf{Q}_{S^C S}$ is a matrix of dimension $|S^C|$ x $|S|$, $\mathbf{0}_{S^C}$ is the all zero vector of dimension $|S^C|$, ε_i is the i-th element of $\boldsymbol{\varepsilon}_S$ and $\bar{\varepsilon}$ is the given constraint in the mean of $\boldsymbol{\varepsilon}_S$. This constrained overdetermined system of equations has been thoroughly studied and can be solved using standard tools [31].

To conclude this section, the CDA allows us to get the optimal distribution of error in the labeling. That is, if we had as only constraint the mean labeling error, it would give us the optimal error probabilities for each of the data points (3.9). We will show in Chapter 5 that using these optimal values improves greatly the prediction error. However, these optimal values cannot be obtained in the real world, because the workers have fixed error probabilities that do not match in general those from the optimal solution. Instead, we would have to minimize (3.8), which requires checking each possible configuration of workers. Our proposed solution (see Algorithm 3) is to compute the optimal values and then sort the nodes in S in increasing order of ε_i and then allocate workers from better to worse following that order.

Algorithm 3 Covariance dispersion algorithm

Input: Adjacency matrix $\mathbf{W}$, optimal subset of nodes S, worker probability errors $\boldsymbol{\varepsilon}$

Output: Optimal distribution of error $\boldsymbol{\varepsilon}_{opt}$, optimal assignation of workers $\boldsymbol{\varepsilon}$

1: $\mathbf{K} \leftarrow createCovarianceMatrix(\mathbf{W})$
2: $\mathbf{K}_c \leftarrow createCondCovMatrix(\mathbf{K}, S)$
3: $\mathbf{Q} \leftarrow createEstimatorMatrix(\mathbf{K}, S)$
4: $[\mathbf{U}, \boldsymbol{\Lambda}] \leftarrow getEigenvectors(\mathbf{K}_c)$
5: $\boldsymbol{\varepsilon}_{opt} \leftarrow solveConstrLinLSQ(\mathbf{U}, \boldsymbol{\Lambda}, \mathbf{Q}, \boldsymbol{\varepsilon})$
6: $\boldsymbol{\varepsilon} \leftarrow reorderWorkers(\boldsymbol{\varepsilon}_{opt}, S)$

Time Complexity: $O(|\mathcal{V}|^3)$

3.2 Node contribution algorithm

In this section we will present the NCA we have developed to optimize the assignment of unreliable workers (see Algorithm 4 at the end of this section). There are several reasons we have developed an alternative method to solve the same problem. First, although we will have to make the same initial assumptions as the other method to be able to use the GRF model, in the CDA we had to relax the problem rather than solving the integer problem. Second, we are forced to obtain all the eigenvectors and eigenvalues of the covariance matrix, which makes for most of the computational time, especially for the kind of machine learning problems we try to solve in this thesis where

$S^C \gg S$. Third, although the covariance matrix will always be invertible, from (2.2) we see that we could lose a lot of precision when λ_1 is too small due to numerical issues. Additionally, the covariance matrix cannot determine the real density function of $\mathbf{f}_{S^C}$ if it is not generated by a GRF. Although in the NCA we will start assuming generation by a GRF, we will not use in any way the covariance matrix, so that we are not as dependent on $\mathbf{f}$ being a Gaussian graph signal.

Following the same initial reasoning as for the previous method, we know that getting the labels for all data points by interpolating a subset of labeled points S is equivalent to making a MAP estimation [32]. The MAP estimate $\boldsymbol{\mu}_{S^C|S}$ and the conditional covariance matrix of the process $\mathbf{K}_{S^C|S}$ are given by (2.3) and (2.4) respectively. To predict the labels of S^C we evaluate $sign(\boldsymbol{\mu}_{S^C|S})$. The label will have the same class as the sign of this function.

This is where we find another intuition to solve the assignment problem. The sign of the MAP estimate is all that is required to classify the predicted labels. Because of that, we could just minimize the number of nodes whose MAP estimate changes sign due to error. That is, instead of minimizing the prediction error, we try to minimize the chance that any of the components of $\tilde{\boldsymbol{\mu}}_{S^C|S}$ has a different sign than the corresponding one in $\boldsymbol{\mu}_{S^C|S}$.

The MAP estimate is dependent on the values of the labels of S. However, for the prediction of each node of S^C, not all the labels obtained in S have the same weight. In fact, the weight that each node of S has on each node of S^C is determined by the values of the matrix $\mathbf{Q}$. $\mathbf{Q}$, which we will call the *estimator matrix*, is obtained from the subset of nodes S chosen and the values of the covariance matrix $\mathbf{K}$. Let us define the MAP estimation from (2.3) as:

$$\boldsymbol{\mu}_{S^C|S} = \sum_{j \subset S} l_j \mathbf{q}_j \tag{3.10}$$

where $\mathbf{q}_j$ is the j-th column of $\mathbf{Q} = \mathbf{K}_{S^C S}(\mathbf{K}_S)^+$. Let $\mu_i = \sum_{j \subset S} l_j q_{ij}$ be the MAP estimator of the i-th node of S^C. We know that l_j is a random variable that takes values $+1$ or -1, and the coefficients q_{ij} are known. Hence, estimating the variable μ_i is analogous to the following random walk problem: determining the position x of a particle that is initially located at zero and moves right or left with equal probability in steps of sizes q_{i1}, q_{i2}, etc.

3.2.1 Binary classification

Let us first explain the assignment procedure when the target only has two classes. We will examine first the special case $P(l_i = +1) = P(l_i = -1) = 0.5$. It is important to note that l_i is the true label and not the noisy label. Moving to right or left when making a particular step is equally probable. Thus, we get that $E\{\mu_i\} = 0$, which is trivial. However, what is more interesting is the distribution of μ_i, which gives an idea of where the random walk is likely to end. As a simple characterization for this interval we could use the interval given by $E\{\mu_i\} \pm std(\mu_i)$. If we were able to express these values in terms of the weights q_{ij}, we could get an idea of how much each node $j \in S$ contributes to the uncertainty of the value of a given node $i \in S^C$. We would

want to reduce this uncertainty by putting the workers with less labeling error probability in the nodes that produce most of the value variability. This would maximize the probability that the label predicted by the noisy set of labels in S has the same class as the label predicted when all the training set labels are true labels. In other words, think of each node $i \in S^C$ as a poll in which each node of S votes for a positive or negative outcome and the weight of each vote depends on the coefficient q_{ij}. This would be similar to weighted voting.

We can express the uncertainty of the result of each of the votes by measuring the standard deviation or the variance of each node $i \in S^C$. We will compute the variance for simplicity:

$$\begin{aligned} E\left\{\mu_i^2\right\} - E\left\{\mu_i\right\}^2 &= \sum_{j \subset S} E\left\{q_{ij}^2 l_j^2\right\} + \sum_{\substack{j,k \subset S \\ j \neq k}} E\left\{q_{ij} q_{ik} l_j l_k\right\} - 0 \\ &= \sum_{j \subset S} q_{ij}^2 + \sum_{\substack{j,k \subset S \\ j \neq k}} q_{ij} q_{ik} E\left\{l_j l_k\right\} \\ &\simeq \sum_{j \subset S} q_{ij}^2 \end{aligned} \tag{3.11}$$

where we make the approximation $E\left\{l_j l_k\right\} \simeq 0$ which is valid when $|S| \ll |S^C|$. This is equivalent to saying that because S is small, there is almost no similarity between nodes of S (they are located far to each other in the graph) and their values can be considered independent because they have almost no effect to the smoothness of the graph. Denote k_{ij} the contribution given by the node $j \in S$ to the variance of the node $i \in S^C$. We define k_j as the *importance weight* of the node j, which measures the total contribution of that node to the prediction.

$$k_{ij} = \frac{q_{ij}^2}{\sum_{j \subset S} q_{ij}^2} \tag{3.12}$$

$$k_j = \sum_{i \subset S^C} k_{ij} \tag{3.13}$$

$$\sum_{j \subset S} k_j = |S^C| \tag{3.14}$$

The reason k_{ij} is normalized is so that all the nodes of S^C are considered equally important. Because the value of each node $i \in S^C$ is only determined by the sign of μ_i, the absolute quantity of variance removed is not relevant and instead it is the relative variance that matters, further reinforcing the necessity of normalizing. The criterion of this method will be to assign the best worker to the node that most contributes on average to the variance of the nodes of S^C, which is given by k_j. Knowing the values of the nodes that make most of the variance let us put noisy workers in the remaining nodes, because they will have little to no influence in the result. Summing up, the workers must

be assigned so that we minimize $\boldsymbol{\kappa\varepsilon}$:

$$\boldsymbol{\kappa\varepsilon} = \sum_{j \in S} k_j \varepsilon_j \tag{3.15}$$

where $\boldsymbol{\kappa} = [k_1, k_2, ...]$ is the row vector of the importance weights and $\boldsymbol{\varepsilon}$ is the column vector that has for components the labeling probability error of the worker assigned to each sample of S. Trivially, we can see that to minimize this quantity we have to assign the most noisy worker to the sample with minimum importance weight.

For the general case $p_+ = P(l_i = +1) \neq P(l_i = -1)$, we have that the variance is $E\{\mu_i^2\} - E\{\mu_i\}^2 = 4p_+(1-p_+)\sum_{j \subset S} q_{ij}^2$, which is just the previous case scaled. Thus $\boldsymbol{\kappa}$ will be scaled as well and therefore the assignment chosen will be the same as the first case.

3.2.2 Multiclass classification

We now consider in more detail than for the CDA, the case of multiclass classification. This will be useful for the next chapter, where we will use a slight variation of this specific method for the unconstrained problem, in which we can use multiple labels in the same sample (which is the same as adding redundancy to reduce error when labeling). The changes are as follows. First, as mentioned in the previous method, instead of a positive or negative label, we will have a one hot vector for each class. This means that instead of having just one μ_i, we will have:

$$\mu_{in} = \sum_{j \subset S} l_{jn} q_{ij} \tag{3.16}$$

where $n = \{1 \dots C\}$ with C the number of classes, and $l_{jn} = 1$ if the label in the node j is of class n and $l_{jl} = 0$ otherwise. Second, we would want to reduce the uncertainty in each one of the classes, unlike in the binary case where we only reduce the uncertainty of the positive class. So, we will minimize the quantity $\boldsymbol{\kappa\varepsilon} = \sum_{n=1}^{C} \boldsymbol{\kappa}_n \boldsymbol{\varepsilon}$, where $\boldsymbol{\kappa}_n$ is the row vector of importance weights for the class n.

Let p_n be the prior probability that the node j has class n. If there are the same number of nodes for each of the classes we would have that $p_n = 1/C$. Also, without any prior information we will have to assume this. From (3.16) we can derive the following:

$$\begin{aligned} E\{\mu_{in}^2\} &= \sum_{j \subset S} E\{q_{ij}^2 l_{jn}^2\} + \sum_{\substack{j,k \subset S \\ j \neq k}} E\{q_{ij} q_{ik} l_{jn} l_{kn}\} \\ &= \sum_{j \subset S} q_{ij}^2 p_n + \sum_{\substack{j,k \subset S \\ j \neq k}} q_{ij} q_{ik} p_n^2 \end{aligned} \tag{3.17}$$

$$\begin{aligned} E\left\{\mu_{in}\right\}^2 &= \left(\sum_{j\subset S} E\left\{q_{ij}l_{jn}\right\}\right)^2 = \left(\sum_{j\subset S} q_{ij}p_{jn}\right)^2 \\ &= \sum_{j\subset S} q_{ij}^2 p_n^2 + \sum_{\substack{j,k\subset S \\ j\neq k}} q_{ij}q_{ik}p_n^2 \end{aligned} \tag{3.18}$$

$$E\left\{\mu_{in}^2\right\} - E\left\{\mu_{in}\right\}^2 = p_n(1-p_n)\sum_{j\subset S} q_{ij}^2 \tag{3.19}$$

We now normalize the variance so that each node $i \in S^C$ has the same importance, as in the binary case. For that we will just divide by $\sum_{j\subset S} q_{ij}^2$. Given a class n, we see that its vector of importance weights $\boldsymbol{\kappa}_n$ is scaled by the term $p_n(1-p_n)$. We know that if it was not scaled, the minimizer function would be the same as (3.15). Scaling, we will have that:

$$\boldsymbol{\kappa}_n\boldsymbol{\varepsilon} = p_n(1-p_n)\sum_{j\subset S} k_j\varepsilon_j \tag{3.20}$$

$$\boldsymbol{\kappa}\boldsymbol{\varepsilon} = \sum_{n=1}^{C} p_n(1-p_n)\sum_{j\subset S} k_j\varepsilon_j \tag{3.21}$$

So $\boldsymbol{\kappa}$ is just a scaled version respect to the binary case. This means that the assignment order is independent on the number of classes the target has.

Algorithm 4 Node contribution algorithm

Input: Adjacency matrix **W**, optimal subsets of n nodes S
Output: Optimal assignation of workers $\boldsymbol{\varepsilon}$

1: $\mathbf{K} \leftarrow createCovarianceMatrix(\mathbf{W})$
2: $\mathbf{Q} \leftarrow createEstimatorMatrix(\mathbf{K}, S)$
3: $\boldsymbol{\kappa} \leftarrow getImportanceWeights(\mathbf{Q}^2)$
4: $\boldsymbol{\varepsilon} \leftarrow reorderWorkers(\boldsymbol{\kappa}, S)$

Time Complexity: $O(|\mathcal{V}|^2 + |S|^3)$

Chapter 4

Multi-label problem algorithms

4.1 Extension of the node contribution algorithm

In the previous chapter we have used the NCA to assign workers when only one worker can be assigned to label a given sample. Because of this, we did not consider calculating the variance when there are nodes already labeled. Thus, we could assume $p_{jn} = p_n$ in equations (3.17), (3.18) and (3.19). That is, we had always started from the initial state, where we only have a *prior probability* of each class, $p_n^{(0)}$, representing the probability of any node being of a certain class before any labeling has been done. E.g., a face recognition dataset generated from images of random people will have approximately the same number of male and female faces, so in this case $p_{male}^{(0)} = p_{female}^{(0)} = 0.5$.

In this section, we will use $p_{jn} \neq p_n$, which is true for any general state. We want to know where to assign the next label, given an state in which we had already labeled a certain number of times. This differs from what we did in the previous section, where we started from a state where there were no labels and we wanted to assign all the workers at the same time. Because we could have already labels in any general state we cannot do the substitution $p_{jn} = p_n^{(0)}$, like we did in the previous section. Another difference is the function we minimize. In this case, we cannot just multiply a vector of importance weights with a vector of errors like we did in (3.21). The reason is that ε_i is not directly related with the labeling probability of any of the workers from the workforce. For example, if the sample has multiple labels, ε_i would be a combination of the error probabilities of all participant workers.

Searching for an optimal $\boldsymbol{\kappa\varepsilon}$ is simply unfeasible in this case. The solution is not as simple as sorting like in the NCA. We just cannot evaluate for each possible $\boldsymbol{\varepsilon}$. The reason for that is that for W workers and assuming the general case where each worker have an unique labeling error probability, there are a total of $W^{|S|}$ different error vectors $\boldsymbol{\kappa\varepsilon}$ (including permutations of its components). Our proposed solution is an iterative heuristic that tries to greedily reduce a cost function $\mathcal{H}$. We know that each component of $\boldsymbol{\kappa}$ is the normalized variance that one node of S contributes to all nodes of S^C in all target classes. Our cost function should then reduce the normalized variance contributed by all the nodes of S, which is similar of adding all components of $\boldsymbol{\kappa}$. That is:

$$\mathcal{H} = \sum_{j \subset S} \sum_{n=1}^{C} k_j p_{jn}(1 - p_{jn}) \tag{4.1}$$

where p_{jn} is the probability that the node j is of class n and k_j is the importance weight of node j as computed in (3.13).

Let us define $\mathcal{H}_i$ as the value the cost function has after applying iteratively i labels. For simplicity let us assume that we know initially that any node is equally likely to be of any class, i.e, $p_{jn} = 1/C$. Then the cost function before adding any labels will be:

$$\mathcal{H}_0 = \sum_{j \subset S} \sum_{n=1}^{C} k_j p_{jn}(1 - p_{jn}) = \frac{C-1}{C} \sum_{j \subset S} k_j \tag{4.2}$$

Now, for any state $\mathcal{H}_i$, we need to choose which node of S should be labeled next. The criterion when choosing a node is to maximally reduce the cost function. Let $p_{jn}^{(0)}$ and $p_{jn}^{(1)}$ be the probabilities that sample j belongs to class n, before and after adding a label. We can then express our criterion as:

$$\max_j \mathcal{H}_i - \mathcal{H}_{i+1} = \max_j k_j \sum_{n=1}^{C} p_{jn}^{(0)}(1 - p_{jn}^{(0)}) - p_{jn}^{(1)}(1 - p_{jn}^{(1)}) \tag{4.3}$$

Using this criterion we can determine iteratively which will be the node that gives the most uncertainty and then assign a worker to it. The fact that we have k_j in this criterion means that not only the error of each sample is relevant, but also its role in the graph.

However, before using this approach we need to solve two additional problems. First, suppose that we have a sample with a certain number of labels given by different workers whose labeling error probability is known. These labels need not be in agreement. For example, there could be two labels that state that the node is of class 1, three labels for class 2,... etc. The question is, what is the probability of the sample being of class n in such a scheme? Second, suppose that we know these probabilities and that we want to add another label from a new worker of labeling probability p_e. What is the expected probability of that node being of class n given that we do not know beforehand the new label?

We will answer these questions in the next section.

4.2 Estimating the class probabilities for any node

In this section we will derive step by step that the probability of a node being of class n is the softmax of a vector $\mathbf{x}$ with components $x_n = \sum_i \ln(p_{in})$, where p_{in} is the probability that the i-th label of that node is of class n.

Let us make the following initial assumptions for simplicity. They will be relaxed through this chapter. First, assume we only have two classes: positive

and negative. Second, the prior probabilities of the node being positive or negative are equal. Third, all workers have a labeling error probability of ε.

Let $\mathcal{L} = (n^+, n^-)$ be a two component vector with the number of positive and negative labels. Let p^+ be the probability that the true class of the node is positive.

First, for $n^+ = n^-$ we have that $p^+ = 0.5$. Because all the workers have the same probability of making a mistake, having the same number of positive and negative labels does not give any information about the class of the node, so we have to use the prior probability of the node being positive. This also implies that p^+ is not dependent on the number of labels, but on the difference between positive and negative labels.

$$p^+_{\mathcal{L}=(n^+,n^-)} = p^+_{\mathcal{L}=(n^++k,n^-+k)} \tag{4.4}$$

Second, given d, the difference between positive and negative labels for a given sample, we have that:

$$p^+_{(\mathcal{L}=(d,0))} = \frac{(1-\varepsilon)^d}{\varepsilon^d + (1-\varepsilon)^d} \tag{4.5}$$

Because the left side is a probability, this equation can be thought as proportion of desired cases divided by possible cases. In (4.5), all d labels given are positive, so we will make the decision that the node probably is positive. Thus, our desired case is that our decision is correct, that is, all labels are true: $(1-\varepsilon)^d$. But all the possible cases are that all labels are true or all labels are false (ε^d). Hence, the expression given by (4.5).

Third, given any $p^+_{\mathcal{L}^{(0)}}$, by the same reasoning as the previous statement, the probability of that sample being positive after adding one positive label of a worker with labeling probability ε' will be:

$$p^+_{\mathcal{L}^{(1)}} = \frac{p^+_{\mathcal{L}^{(0)}}(1-\varepsilon')}{(1-p^+_{\mathcal{L}^{(0)}})\varepsilon' + p^+_{\mathcal{L}^{(0)}}(1-\varepsilon')} \tag{4.6}$$

if the new label were negative, we would substitute ε' by $(1-\varepsilon')$ and vice versa in the previous equation.

We see that if we add a positive label of error probability ε and then add a negative label of error probability ε', the resulting probability of the node being positive is not the same as if it did not receive any labels at all. This means that equation (4.4) is not applicable when the workers have different probability. This implies that the labels have different importance depending on the labeling probability error of the workers that have labeled.

When there are different quality workers labeling the same sample, instead of using majority voting, where we just count the difference in the number of labels (in (4.5) this would be d), we will use weighted majority voting where each label has a corresponding *label weight* depending on its quality: $x = \sum \omega_i l_i$, where ω_i is the label weight, which depends on the labeling error probability of the worker i and l_i is the value of the label provided by this same worker. If all $w_i = 1$ we have that $x = d$.

Let us show that if we choose by design $\omega_i = \ln((1-\varepsilon_i)/\varepsilon_i)$ as in [33], where ε_i is the labeling error probability of the i-th worker, then p^+ is the sigmoid function of x:

$$p^+(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-\sum \omega_i l_i}} \tag{4.7}$$

To deduce this, we consider the following three situations: the initial state where we do not have any labels, adding multiple labels with the same error probability ε and adding a label that has error probability ε'. If we show that in these three situations the sigmoid gives the same result than the equations that describe these processes, that is, equations (4.4), (4.5) and (4.6), we would have that for any possible state of that sample the sigmoid gives us the positive class probability of that sample.

Verifying equation (4.4) is equivalent to checking that $p^+(0) = 0.5$, which is one of our assumptions. Furthermore, we will verify that adding d labels with probability error ε in (4.7) is equivalent to the expression of (4.5):

$$\begin{aligned} p^+\left(d\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)\right) &= \frac{(1-\varepsilon)^d}{\varepsilon^d+(1-\varepsilon)^d} = \frac{1}{1+\frac{\varepsilon^d}{(1-\varepsilon)^d}} = \frac{1}{1+e^{\ln\left(\frac{\varepsilon}{1-\varepsilon}\right)^d}} \\ &= \frac{1}{1+e^{-d\ln\frac{1-\varepsilon}{\varepsilon}}} \end{aligned} \tag{4.8}$$

Now we will verify that equation (4.7) also holds when adding one positive label of a worker with error labeling probability ε' by comparing it with equation (4.6):

$$\begin{aligned} p^+\left(x+\ln\left(\frac{1-\varepsilon'}{\varepsilon'}\right)\right) &= \frac{\frac{1}{1+e^{-x}}(1-\varepsilon')}{\frac{e^{-x}}{1+e^{-x}}\varepsilon' + \frac{1}{1+e^{-x}}(1-\varepsilon')} = \frac{1}{1+\frac{\varepsilon'}{1-\varepsilon'}e^{-x}} \\ &= \frac{1}{1+e^{-\left(x+\ln\left(\frac{1-\varepsilon'}{\varepsilon'}\right)\right)}} \end{aligned} \tag{4.9}$$

Because all possible situations are obtained from combining in some way these last three cases, we prove that the sigmoid function is indeed the way of estimating the probability of the class being positive, given a weighted sum of labels where each label weight is of the form $\omega_i = \ln((1-\varepsilon_i)/\varepsilon_i)$. In the more general case where the probability of any sample without labels being positive is $p_n \neq 0.5$, we just will add a bias term in the weighted sum whose value will be $\ln(p_n/1-p_n)$. This bias term has p_n instead of $1-p_n$ in the numerator, because p_n is the probability of the node being class n, and not the error probability of that node being class n.

For a multiclass classification problem, with total number of classes C, we have to make some modifications. First, each label quality is not defined by a single probability, but by a vector of probabilities $\mathbf{p}$ in which each component p_n is the probability of the sample being class n. The class of the label is k such that $\max_n p_n = p_k$. Second, instead of a having a scalar weighted sum, we have

a vector of weighted sums $\mathbf{x}$ where each component is defined as $x_n = \sum_i \omega_{in}$. For multiclass classification, $\omega_{in} = \ln(p_{in})$ where p_{in} is the probability that the label i is of class n. The final difference is that instead of using the sigmoid function we will use the softmax function. The probability of the sample being of class n is then defined as:

$$p_n(\mathbf{x}) = \frac{e^{x_n}}{\sum_{n'=1}^{C} e^{x_{n'}}} \tag{4.10}$$

One of the properties of the softmax function is that $p_n(\mathbf{x}) = p_n(\mathbf{x} + k\mathbf{1})$. Using this property it is easy to check that for $C = 2$ we can transform the softmax function into the sigmoid function that we have defined previously. It also can be used to simplify the number of operations we have to do when adding a label.

Let us make the assumption already stated in Section 2.1 that in case of error, all erroneous classes are equally probable. Then we will have that $\omega_{n=k} = \ln p$ and $\omega_{n\neq k} = \ln((1-p)/(C-1))$. We can use the property $p_n(\mathbf{x}) = p_n(\mathbf{x} + k\mathbf{1})$ and substitute $\mathbf{x}$ by $(\mathbf{x} - \ln((1-p)/(C-1))\mathbf{1})$. Then we would have that $\omega'_{n=k} = \ln((C-1)p/(1-p))$ and $\omega'_{n\neq k} = 0$. This greatly simplifies the computational time of our algorithm. We can use the equivalent sum of weights where each component is $x_n = \sum_i \omega'_{ik} l_{in}$ where $l_{in} = 1$ if the label is of class n and $l_{in} = 0$ otherwise.

4.3 Simplifying the cost function $\mathcal{H}$

Now that we can express the probability of any node being of a particular class using a softmax function, let us go back to the NCA. We have stated in Section 4.1 that after computing $\mathcal{H}_0$, the next labels are added so that $\mathcal{H}_i - \mathcal{H}_{i+1}$ is maximized at each iteration. Looking at equation (4.3), we would have to compute the expected $p_{jn}^{(1)}$ in each node to choose whichever j makes $\mathcal{H}_i - \mathcal{H}_{i+1}$ maximum. In this section, we will propose an approximation so that we do not have to compute these new probabilities for each node, reducing computational time.

First, suppose we have chosen a node j. Let us express $p_{jn}^{(0)}$ and $p_{jn}^{(1)}$ as softmax functions. Although we will not know this beforehand, suppose that the label added will be of class k. The probability of that label being correct will be $1 - \varepsilon$, with corresponding weight in the weighted sum of the softmax of $\omega = \ln(1 - \varepsilon)$.

$$p_{jn}^{(0)} = \frac{e^{x_n}}{\mathcal{K}} \qquad p_{jn}^{(1)} = \frac{e^{x_n}}{\mathcal{K} + \zeta} \qquad \text{for } n \neq k \tag{4.11}$$

$$p_{jk}^{(0)} = \frac{e^{x_k}}{\mathcal{K}} \qquad p_{jk}^{(1)} = \frac{e^{x_k + \omega}}{\mathcal{K} + \zeta} \tag{4.12}$$

where $\mathcal{K} = \sum_{n=1}^{C} e^{x_n}$, x_n are the components of the vector of weighted sums $\mathbf{x}$ of the node before adding a label and $\zeta = e^{x_k+\omega} - e^{x_k}$.

We will now try to express $p_{jn}^{(1)}(1-p_{jn}^{(1)})$ as a function of $p_{jn}^{(0)}(1-p_{jn}^{(0)})$ so that we will not have to calculate $p_{jn}^{(1)}$. To make it possible we will do some approximations. Because we do not know beforehand what will be the class of the label received, we will suppose two cases: that it is the same class as the most probable class (our prediction), which happens approximately with probability $(1-\varepsilon)$, or it is not the same class. Then we will average both cases to obtain a general expression.

Consider that the most probable class were k. First, we will make the approximation $e^{x_k+\omega} \gg e^{x_k}$, which is valid when the new label received is much more probable to belong to the node true class than a false one. Second, in the case that the label class were the true class of the node, we will do the approximation $e^{x_k} \gg e^{x_n}$, which is valid if before receiving the new label, we had that the probability of the node being the true class was much greater than any other class. This second approximation is better applied in labeled nodes or when the prior probabilities of unlabeled nodes favor one class over all the others.

$$\begin{aligned}\frac{p_{jn}^{(1)}(1-p_{jn}^{(1)})}{p_{jn}^{(0)}(1-p_{jn}^{(0)})} &= \left(\frac{e^{x_n}}{\mathcal{K}+\zeta}\frac{\mathcal{K}+\zeta-e^{x_n}}{\mathcal{K}+\zeta}\right)\Bigg/\left(\frac{e^{x_n}}{\mathcal{K}}\frac{\mathcal{K}-e^{x_n}}{\mathcal{K}}\right)\\ &= \left(\frac{\mathcal{K}}{\mathcal{K}+\zeta}\right)^2\frac{\mathcal{K}+\zeta-e^{x_n}}{\mathcal{K}-e^{x_n}} \simeq \left(\frac{\mathcal{K}}{\zeta}\right)^2\frac{\zeta}{\mathcal{K}} \\ &= e^{-\omega} = \frac{\varepsilon}{(C-1)(1-\varepsilon)} \qquad \text{for } n \neq k\end{aligned} \tag{4.13}$$

$$\begin{aligned}\frac{p_{jk}^{(1)}(1-p_{jk}^{(1)})}{p_{jk}^{(0)}(1-p_{jk}^{(0)})} &= \left(\frac{e^{x_k+\omega}}{\mathcal{K}+\zeta}\frac{\mathcal{K}+\zeta-e^{x_k+\omega}}{\mathcal{K}+\zeta}\right)\Bigg/\left(\frac{e^{x_k}}{\mathcal{K}}\frac{\mathcal{K}-e^{x_k}}{\mathcal{K}}\right)\\ &= \left(\frac{\mathcal{K}}{\mathcal{K}+\zeta}\right)^2\frac{e^{x_k+\omega}}{e^{x_k}}\frac{\mathcal{K}+\zeta-e^{x_k+\omega}}{\mathcal{K}-e^{x_k}} \simeq \left(\frac{\mathcal{K}}{\zeta}\right)^2 e^{\omega} \\ &= e^{-\omega} = \frac{\varepsilon}{(C-1)(1-\varepsilon)}\end{aligned} \tag{4.14}$$

Now, in the case that the most probable class were not k, instead of the approximation $e^{x_k} \gg e^{x_n}$ we will use the approximation $e^{x_k+\omega} \ll \mathcal{K}$. This approximation is valid if after adding the wrong label we did not change our estimate of the class of the node labeled. That is, it is less probable to estimate the class node wrongly before adding the label than the probability of this additional label being incorrect. This holds when the workers have similar error probabilities for nodes that have received multiple labels already.

$$\frac{p_{jn}^{(1)}(1-p_{jn}^{(1)})}{p_{jn}^{(0)}(1-p_{jn}^{(0)})} = \left(\frac{\mathcal{K}}{\mathcal{K}+\zeta}\right)^2\frac{\mathcal{K}+\zeta-e^{x_n}}{\mathcal{K}-e^{x_n}} \simeq 1 \tag{4.15}$$

$$\begin{aligned}\frac{p_{jk}^{(1)}(1-p_{jk}^{(1)})}{p_{jk}^{(0)}(1-p_{jk}^{(0)})} &= \left(\frac{\mathcal{K}}{\mathcal{K}+\zeta}\right)^2 \frac{e^{x_k+\omega}}{e^{x_k}} \frac{\mathcal{K}+\zeta-e^{x_k+\omega}}{\mathcal{K}-e^{x_k}} \simeq e^{\omega} \\ &= \frac{(C-1)(1-\varepsilon)}{\varepsilon}\end{aligned} \tag{4.16}$$

So, using the fact that the label will be correct $(1-\varepsilon)$ of the times and incorrect ε of the times, we can calculate the mean value:

$$\begin{aligned}E\left\{\sum_{n=1}^{C} \frac{p_{jk}^{(1)}(1-p_{jk}^{(1)})}{p_{jk}^{(0)}(1-p_{jk}^{(0)})}\right\} &= (1-\varepsilon)\frac{\varepsilon}{(C-1)(1-\varepsilon)} + \varepsilon\frac{(C-1)(1-\varepsilon)}{\varepsilon} \\ &+ \sum_{n\neq k} p\frac{\varepsilon}{(C-1)(1-\varepsilon)} + \varepsilon \\ &= \sum_{n=1}^{C} \frac{\varepsilon}{(C-1)} + \frac{(C-1)}{C}\end{aligned} \tag{4.17}$$

Although we may not fulfill in the first iterations the conditions needed for the approximations used in (4.13), (4.14), (4.15) and (4.16) to be valid, we accept that initial loss in accuracy to simplify our problem. As we add new labels in posterior iterations the approximations are more accurate, because we have more multi-labeled nodes (so in general one class will be much more probable than the rest), and in the case that we update the class probabilities of the unlabeled nodes, as we will explain in detail in the next section, we will have that one class will be more probable than the rest for unlabeled nodes too. In the end, substituting (4.17) in (4.3) we get:

$$\max_j \mathcal{H}_i - \mathcal{H}_{i+1} \simeq \max_j k_j \sum_{n=1}^{C} p_{jn}^{(0)}(1-p_{jn}^{(0)}) \left(1 - \frac{\varepsilon}{(C-1)} - \frac{(C-1)}{C}\right) \tag{4.18}$$

which makes it easier to maximize because this new term is independent of j, so we can ignore it. This also implies that adding labels to a node reduces exponentially the quantity that that node adds to the cost function. This also means that even if any node j has a very high k_j, there will be a point were the benefit of adding another label to it will be outweighed by the fact that the other nodes have much less labels.

The inaccuracy from using these approximations is greatly diminished when all the nodes have similar number of labels, when the number of labels is increased, when the workers have lower labeling probability error, when there is great difference between the probability of the most probable class and the other classes and when we increase the number of classes.

In conclusion, the extension of the NCA we have designed, which we will denote as *multi-label node contribution algorithm* (MNCA), minimizes the cost function $\mathcal{H}$ using equations (4.2) and (4.18), and updating the class probabilities of the labeled nodes at each step (see Algorithm 5).

Algorithm 5 Multi-label node contribution algorithm

Input: Adjacency matrix **W**, optimal subsets of n nodes S. When required, the error probability of the next worker ε and the class of its generated label l

Output: The index of the next node to be labeled j

1: $\mathbf{K} \leftarrow createCovarianceMatrix(\mathbf{W})$
2: $\mathbf{Q} \leftarrow createEstimatorMatrix(\mathbf{K}, S)$
3: $\boldsymbol{\kappa} \leftarrow getImportanceWeights(\mathbf{Q}^2)$
4: $j \leftarrow max(\boldsymbol{\kappa})$
5: **while** there are still labels to assign **do**
6: $\quad p_j \leftarrow updateNodeProbability(\varepsilon, p_j)$ (4.10)
7: $\quad j \leftarrow max(\boldsymbol{\kappa}\boldsymbol{p}(\mathbf{1} - \boldsymbol{p}))$ (4.18)
8: **end while**

Time Complexity: Before labeling, $O(|\mathcal{V}|^2 + |S|^3)$. Between labels, $O(|S|)$

4.4 Updating the unlabeled nodes

One additional matter that affects the estimation are the class probabilities of the unlabeled nodes. At first, when we do not have any labels yet, we have used the reasonable assumption that the class probabilities for all unlabeled nodes are $p_n = 1/C$. However, as the number of labels obtained increases, this assumption loses validity. The reason for that is that we could just label all the remaining unlabeled nodes from the set S by interpolating from the already labeled nodes. Because we expect the graph signal (with label information) to be low-pass, interpolating, even with a very limited set of nodes, is better than just choosing the class of an unlabeled node at random.

This effect is due to the cluster-like behavior of the graph signal, unlabeled nodes that belong to the same cluster than some labeled nodes have greater probability of sharing the same class as them. Although our heuristic has taken into account the fact that the samples of S have an impact on the values of the samples of S^C, we have not taken into account the inter influence that each sample of S has on the rest yet. This also implies that we could reduce the chance of error of the unlabeled, and even the labeled nodes, by taking into account the values of close labeled nodes of S.

Thus, we present a different approach (see Algorithm 6) that is better than not updating the prior probabilities. This approach is based on substituting the fixed probabilities of the unlabeled nodes $p_n = 1/C$ by $p_{n=k} = 1 - \varepsilon_{pred}$ and $p_{n \neq k} = \varepsilon_{pred}/(C-1)$ where ε_{pred} is an estimation of the total prediction error for the current number of labels. This estimation is obtained during the labeling process by comparing the class of the labels applied to unlabeled nodes with their previous predicted classes. This means that we cannot estimate the current prediction error but the prediction error of a state where we had fewer labeled nodes. This makes our estimation always worse than the real prediction error, as we will see in Chapter 5.

Algorithm 6 Improved MNCA

Input: Adjacency matrix **W**, optimal subsets of n nodes S. When required, the error probability of the next worker ε and the class of its generated label l

Output: The index of the next node to be labeled j

1: $\mathbf{K} \leftarrow createCovarianceMatrix(\mathbf{W})$
2: $\mathbf{Q} \leftarrow createEstimatorMatrix(\mathbf{K}, S)$
3: $\boldsymbol{\kappa} \leftarrow getImportanceWeights(\mathbf{Q}^2)$
4: $j \leftarrow max(\boldsymbol{\kappa})$
5: **while** there are still labels to assign **do**
6: $\quad p_j \leftarrow updateNodeProbability(\varepsilon, p_j)$
7: $\quad$ **if** $count == C$ **then** (*)
8: $\quad\quad numLabels \leftarrow countLabeledNodes()$
9: $\quad\quad \tilde{\boldsymbol{l}}^{(numLabels)} \leftarrow interpolateLabeledNodes(\boldsymbol{p})$
10: $\quad\quad \varepsilon_{pred} \leftarrow estimatePredError(\tilde{\boldsymbol{l}}^{(numLabels-k)}, \boldsymbol{p})$
11: $\quad\quad \boldsymbol{p} \leftarrow updateUnlabeledNodes(\boldsymbol{p}, \varepsilon_{pred})$
12: $\quad\quad count = 0$
13: $\quad$ **end if**
14: $\quad j \leftarrow max(\boldsymbol{\kappa p \bar{p}})$
15: $\quad count++$
16: **end while**

Time Complexity: Before labeling, $O(|\mathcal{V}|^2 + |S|^3)$. Between labels, $O(|S|)$ or $O(|\mathcal{V}|^2)$ (if we update the unlabeled nodes probabilities)

(*) = from steps (7) to (13) we are estimating the prediction error

Chapter 5

Experimental results

An important part of this study is to show if all the proposed methods given in the previous chapter hold empirically. To test this we have designed software that is capable of:

- Given a dataset, create its corresponding associated graph.
- Given the graph and a number of points, find the optimal subset S that best represents the graph.
- Given the graph, an optimal subset S and the probability error of each worker, solving the assignment problem using the proposed methods.
- Simulating the labeling and predict the remaining labels using a given allocation of workers in a subset S.

The software has been developed using Matlab R2014a. The code is available in GitHub [1]. The datasets used for the following experiments are the USPS handwritten digits dataset [2] and the Isolet dataset [3].

5.1 Generating a graph from each dataset

The USPS handwritten digits dataset consists of 1100 16 x 16 pixel images for each of the digits 0 to 9. The target is the value of each digit. Thus, it is a categorical variable with 10 classes. The features are each one of the pixels, which are valued between 0 and 255 (grayscale value). From this dataset, we randomly select 100 images for each digit class to create one instance of our dataset. So, each instance of our dataset consists of 1000 feature vectors of dimension 256. We create instances from the dataset until we have 30 of them. For each instance, we construct a graph using Gaussian kernel weights $w_{ij} = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2)$, where $\mathbf{x}_i$ is the 256-dimensional feature vector of one of the images. The parameter σ is chosen to be 1/3-rd of the average distance to the K-th nearest neighbor (KNN) for all data points as suggested in [34, 26]. We fix K = 10 and force the edges to be mutual connections after the KNN. That is to say, if $w_{ij} = 0 \neq w_{ji} \Rightarrow w_{ij} = w_{ji}$. This results in a symmetric adjacency matrix $\mathbf{W}$ for the graph, where $\mathbf{W}_{ij} = w_{ij}$.

[1] https://github.com/javier-maroto/GSP-Noisy-Crowdsourcing
[2] http://www.cs.nyu.edu/~roweis/data.html
[3] http://archive.ics.uci.edu/ml/datasets/ISOLET

The Isolet dataset consists of letters of the English alphabet spoken in isolation twice by 150 different subjects. The target is the value of each spoken letter. Thus, is a categorical variable with 26 classes. Each alphabet utterance has been preprocessed beforehand to create a 617-dimensional feature vector. The approach to generate the graphs is similar to the one used in the USPS dataset. Notable differences are the size and number of instances. Because the problem has more target categories than the USPS dataset, when randomly selecting 100 recordings of each letter, we generate instances of 2600 feature vectors instead. Bigger instances hugely increment the time needed for each simulation, so instead of 30 instances we generate 10. The graphs are also constructed using Gaussian kernel weights, and the σ and K are fixed similarly.

These datasets have been chosen to analyze the differences in behavior when varying the number of dimensions of the feature and target space or varying the context of the classification problem. All the adjacency matrices generated in this process are stored, for use as inputs in the next phase: selecting the optimal nodes.

5.2 Selecting the optimal nodes

We have implemented two different methods that select the subset that best represents the graph, for a given number of nodes.

The *'default'* method (see Algorithm 1) is similar to the selection scheme used in [6]. The idea is to maximize the cut-off graph frequency of the graph signals generated by the subset of nodes chosen. This is shown to be equivalent to maximizing the eigenvalue of the Laplacian matrix of the nodes of the complementary subset, $\lambda(\mathbf{L}_{\mathcal{S}^C})$. The algorithm uses the fact that the gradient of this eigenvalue is maximized when we add the node corresponding to the biggest absolute value in the eigenvector of $\mathbf{L}_{\mathcal{S}^C}$: $\mathbf{v}(\mathbf{L}_{\mathcal{S}^C})$. Thus, the algorithm chooses greedily one node at a time in order to obtain a sampling set.

The *'distance'* method (see Algorithm 2) is based on the intuition that every node predicted will obtain most of the information from labeled nodes that are close to it. This intuition has been used before for clustering. The k-medoids method [28] relies in maximizing the similarity between the features of any of the data points and the closest cluster representative (where there are k cluster representatives). As we can see, the *'distance'* method and the k-medoids method are similar. Closeness in graphs is equivalent to similarity between feature vectors. The samples selected are equivalent to the k cluster representatives of the k-medoids. The only difference is the context, it is normally used for clustering, while we use it for selecting representative nodes. There have been studies that show that k-medoids is competitive against other clustering algorithms like the k-means [35] and it has also been used for active learning [36].

As expected, when comparing both methods we obtain very different results. The graph sampling-based method greatly outperforms the k-medoids method. This is not surprising, because the k-medoids method does not take into account the complex interactions and the connectivity patterns that underline the graph data. It only uses as information the absolute distance

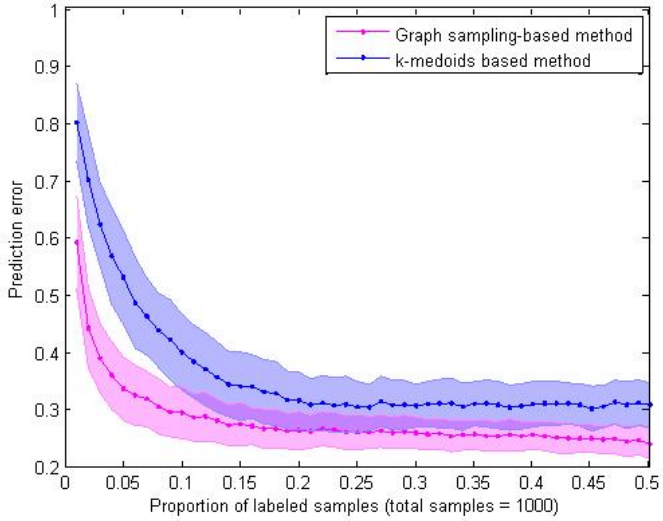


FIGURE 5.1: Comparison between the prediction error obtained by selecting the optimal sets of nodes using the graph sampling or the k-medoids method. The results were obtained combining 300 different simulations. We used the USPS handwritten digits dataset. The workers are assigned randomly to each of the samples.

between each node. However, one disadvantage of the graph sampling-based method is the time complexity of the algorithm. Although computing the distances between nodes in the k-medoids is time-consuming, it is required one time for execution only. However, the graph sampling method requires computing the smallest pair eigenvalue/eigenvector one time for each sample in the optimal subset, which is much more expensive.

5.3 Prediction and labeling simulation

We will skip for now the algorithms for assigning workers to samples. We will explain them in detail in the next section. Suppose that we have the graph, the subsets of nodes that best represents the dataset and the error probabilities expected in each sample. The latter is obtained after generating the error probabilities of each worker and sorting them in the subset of samples using an assignment optimizer method. We would like to evaluate the CDA and NCA algorithms (see Algorithms 3 and 4) based on prediction error for different sizes of subsets, different datasets and even different workers.

However, we cannot check the performance using a real crowdsourcing platform such as Amazon Turk. It is easier to analyze the effects of our assignment algorithms when there are no uncontrollable effects such as uncertainty in the quality of the workers or heterogeneous task difficulty. Hence, all the labeling process is simulated.

For each one of the subsets of nodes, we generate the worker labels. The worker labels are obtained using the equation $\tilde{\boldsymbol{l}}_S = \boldsymbol{l}_S - 2\boldsymbol{\phi}_S \cdot \boldsymbol{l}_S$. We then

interpolate the graph signal that has the values of the worker labels in S and zeros in the remaining nodes using an iterative algorithm developed in [37] based on projection onto convex sets (POCS). This is an iterative process where we force the predicted graph signal to fulfill two constraints: (1) for all nodes in S the predicted graph signal must be equal to the obtained worker labels and (2) the graph signal must be low-pass filtered so that $\mathbf{f} \in PW_{\omega}(\mathcal{G})$.

5.4 Single-label problem

In this section we will simulate and compare the prediction error of the algorithms we used to solve Problem 1, that is, CDA and NCA, with a baseline. We will make this comparison for different sized subsets S_n. Because there are no works that consider this problem, our baseline is just assigning the workers at random. The objective is to obtain insights about the labeling process as well as to evaluate the improvements that our algorithms could have respect to random assignation.

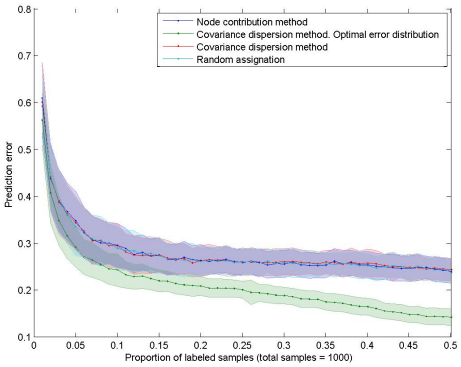
The CDA is thoroughly explained in Section 3.1. The general idea is to apply the Gaussian Random Field model to our graph. The process of interpolating all the labels of the dataset from a labeled subset of data points is quasi-equivalent in this model to making a MAP estimation [32]. The intuition is to try to allocate the error in the axes of the multivariate conditional covariance matrix that carry more dispersion. Thus, reducing the effect of any false label.

This algorithm has high time complexity. This is because for every subset of nodes, we have to compute all the eigenvalues and eigenvectors of the conditional covariance matrix. Not only that, but we also have to solve a constrained linear least-squares problem for each subset of nodes. For that, we use active-set methods, which, in some particular counterexamples, could take exponential time to converge [38]. Defining time complexity in this case is not easy, so we will give a lower bound that does not take into account the active-set methods. To address the time requirements of the algorithm we have reduced the number of subsets we check, only providing allocation solutions to every 10th subset of points.

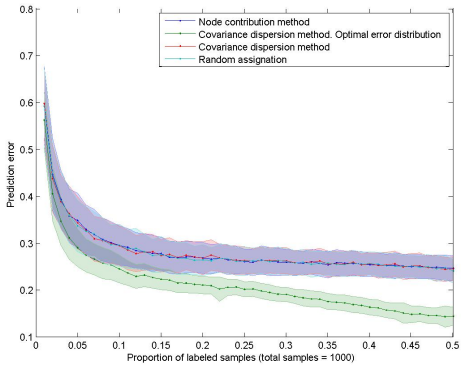
The NCA is explained in detail in Section 3.2. The general idea is to apply the Gaussian Random Field model to our graph. In the previous method we focused on the inherent Gaussian noise that the prediction process has. In this method, instead, we focus in ensuring that the maximum number of labels are predicted as if there was no error in the labeled data points. Modeling the MAP estimator as multiple univariate random walk processes we can determine which data points of the chosen subset have more power in the deciding the rest of the labels. Thus, reducing the number of predicted labels that differ from the non labeling error case.

This algorithm has the advantage that it requires much less computational power to solve the assignment problem. The main time complexity bottleneck is computing inverse matrices of size n x n, where n is the size of the subset chosen. So for graphs where $n_{max} \ll |\mathcal{V}|$ this method is especially faster.

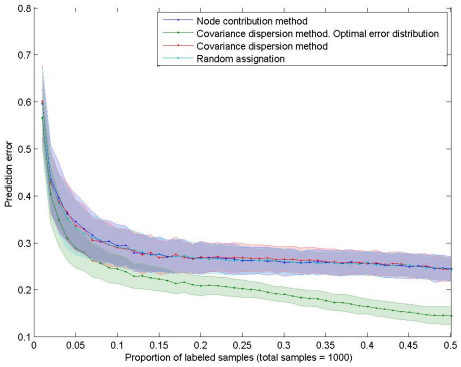
We have chosen a simple way of modeling the worker behavior. Any worker i has associated a probability labeling error ε_i. When that worker labels any given sample, the probability of labeling correctly the sample is $1 - \varepsilon_i$. Without loss of generalization we create as many workers as samples, and assign them in a one-to-one basis. The parameter ε_i for each worker is generated using a probability density function. Most simulations use the probability density functions plotted in Figure A.1.



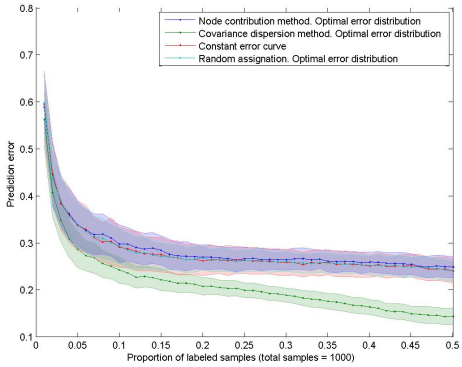
(A) *'Hammer-spammer'* with 40% of spammers



(B) *'Beta'* with $\alpha = 2,\ \beta = 8$



(C) *'Beta'* with $\alpha = 0.5,\ \beta = 2$



(D) *'Constant and optimal distribution'* with $\bar{p} = 0.2$.

FIGURE 5.2: Comparison between the prediction error obtained between the baseline (random assignation) and the selected assignations after using each method. The results were obtained combining 300 different simulations. We used the USPS handwritten digits dataset. All different distributions of error used in each of the Figures have mean = 0.2

We made multiple series of simulations for different distributions and different datasets (Figures 5.2 and A.3). From the results of this study we have obtained some insights about the labeling process:

- Labeling more samples yields little improvement when we already have between 8 to 10% of the samples labeled. That is to say, that we get most of the information of the dataset after labeling between 100 and

200 samples, for the applications and the range of complexity the feature and target space have in our analysis. This of course will depend on the problem we consider. When the workers have perfect quality this behavior also holds (see Figure A.2)

- Changing the distribution of error in the group of workers has almost no impact when we are assigning them randomly. There is no benefit in having an homogeneous group of workers or having a group of hammers and spammers as long as the mean labeling error is the same for both groups, as long as we are using just one label per sample.
- Using the NCA method does not yield any improvement versus just assigning the workers randomly, its modified algorithm, the MNCA, has good results in Problem 2 as we will see in next section. Although our heuristic of sorting the fixed worker labeling errors in the CDA seems to work almost only when the error probabilities of the workers coincide with the optimal distribution of error (figure 5.2d), we expect that an heuristic that better minimizes equation (3.8) could yield significant improvement.

5.5 Solving the multi-label problem

In this section we compare the prediction and labeling error of the algorithms we used to solve Problem 2, that is, MNCA and its improved version (see Chapter 4 and Algorithms 5 and 6), with some baselines: uniform labeling and adaptively minimizing the labeling error. The objective is to obtain insights about the efficiency of our methods respect to the baselines, the effect that the size of the subset S has in the prediction error and the labeling process in this scheme.

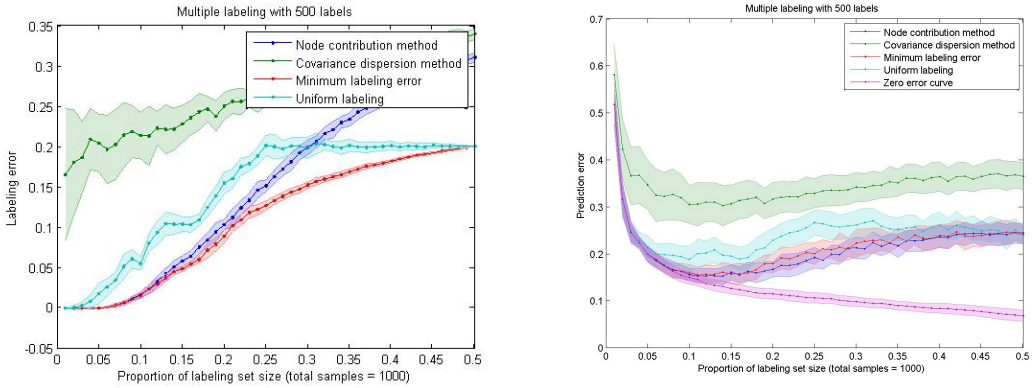
We will present three different experiments. In the first experiment, we will compare MNCA with the baselines and observe the effect of changing the size of S. In the second experiment we will show the improvements that updating the unlabeled nodes class probabilities have. That is, we will show the potential of the improved MNCA. In this second experiment, for reference only, we will update these errors with the real prediction error instead of an estimate. In the third experiment we show the prediction error obtained when using an estimate of the prediction error when updating the unlabeled nodes class probabilities. The difference with respect to the previous experiment is the use of estimates, which is required when applying the improved MNCA in the real world.

The first experiment presents our greedy algorithm, the MNCA (see Algorithm 5), that minimizes at each step the cost function $\mathcal{H}$. We use the approximation applied in equation (4.18), instead of the harder to compute equation (4.3). We do not apply any update in the probabilities of the unlabeled nodes.

This algorithm is compared against more popular methods such as uniform labeling or adaptively adding a label in the node where the probability of error

is maximum. We will show for a given total number of labels, the performance of all these methods as a function of the size of the set S. We will measure principally the mean error of the labeled nodes and the prediction error of the interpolation.

The results from this first experiment where the class signal is the original of the dataset used are plotted in figures 5.3, A.4 and A.5. In Figure 5.3 we have also tried an algorithm based on the CDA but, because its performance is significantly lower than any of the other algorithms, we do not plot it in the rest of simulations.



(A) Labeling error (B) Prediction error

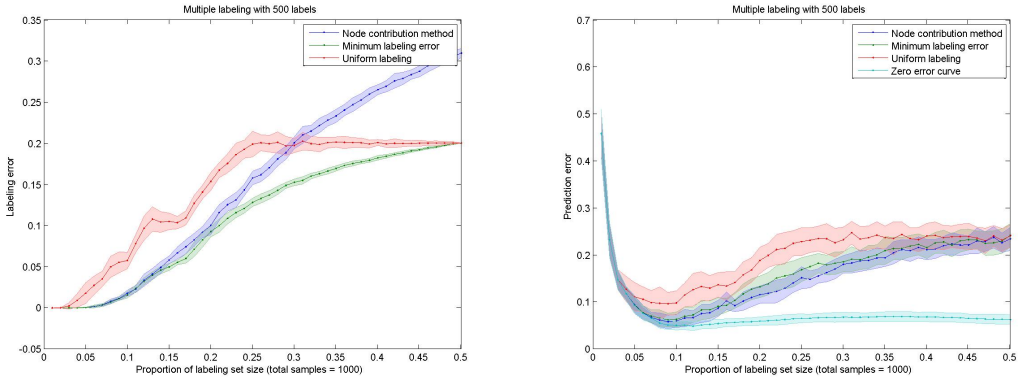
FIGURE 5.3: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The labeling probability error of each worker is 0.2. The number of labels used is 500.

In Figure 5.3a we can see that the better of the three curves is the minimum labeling error algorithm, which is expected. We see also that all three methods have less overall labeling error when S is smaller. This is also expected because the number of labels per sample is larger than when S is bigger, since we keep the number of labels constant. We also see that for S small, both iterative methods outperform the uniform labeling. That is because we spend more efficiently the labels by reducing the error primarily in the nodes that have contradicting labels first, as we cannot determine so easily their class. For S bigger we also see that our method is outperformed, but that is only because with our method we do not label all the nodes of S (only the important ones), so all the unlabeled nodes count as big sources of uncertainty.

In Figure 5.3b we can see that our method outperforms both the baseline methods for any given S. We have shown also what is the zero error curve, or the prediction error curve obtained from perfect labeling. The reason there is a minimum in the prediction error of each of the methods is the fact that we are not using the inter influences of the already labeled nodes, as explained in section 4.4. If we changed the prior probabilities of the unlabeled nodes iteratively as we are getting more labels in S, what we should see is that our algorithm will tend to choose to label less of the unlabeled nodes so that

ultimately the number of labeled nodes ends fixing for all sizes of $S > k$. This will be shown when instead of the MNCA, we use its improved version in the next two experiments.

Intuitively, there is a trade-off between the number of nodes labeled and the mean quality of the nodes labeled. If we have too few nodes labeled, even if they are of perfect quality, our interpolation will be very limited in frequency, which limits greatly the space of true class signals that we can predict. In Figure 5.3b, this would be the zone where the proportion of labeling set size is lower than 0.05, where the derivative of the zero error curve is extremely high (any new node added gives a lot of information). Alternatively, there is a point where having more labeled nodes adds little to no information. This is because the target has most of its energy in its low frequency components. The uncertainty given by the decrease of the mean quality of the nodes will outweigh the information that could be given by having an additional node. This corresponds to the zone where the proportion of the labeling set size is higher than 0.2. In Figure 5.4, we have limited beforehand the frequency of the true class signal so that this trade-off is clearer. Just looking at the zero curve error, we see that after labeling with no errors 10% of the dataset we do not improve the prediction error incrementing the labeling set size.



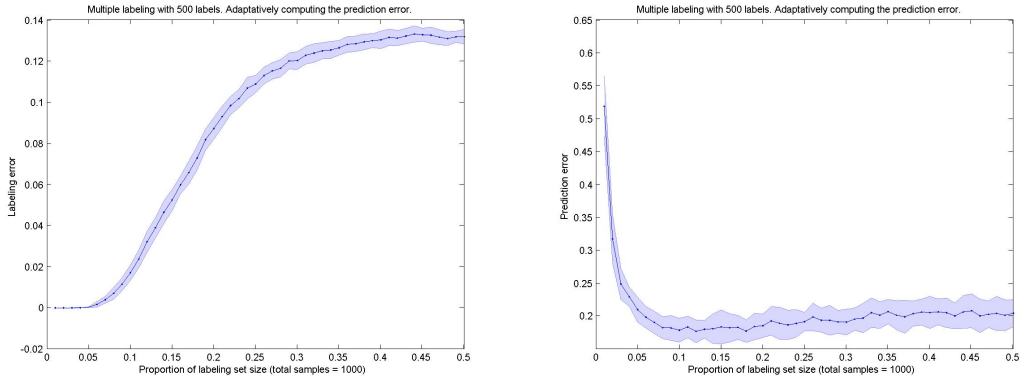
(A) Labeling error (B) Prediction error

FIGURE 5.4: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset limiting the target graph frequency. The labeling probability error of each worker is 0.2. The number of labels used is 500.

If we were to know beforehand the cut-off frequency of the true class signal, one way of improving drastically the prediction error obtained with a larger labeling set sizes would be to filter the labeling graph signal before interpolating. This filters all the labeling noise of high graph frequency, at the cost of reducing the bandwidth of the signals that can be interpolated. The result of filtering using the same cut-off frequency we have applied to the true class signal is shown in Figure A.6. We can see that the noise from the labeling errors it is indeed filtered out, just comparing its prediction error with the

prediction error of Figure 5.4. Alternatively we could set the size of S to be of about 10% of the dataset on this particular case, with the same trade-off we get when filtering.

Ultimately we would want to have a labeling set of size great enough so that we do not lose a lot of information, and also we would want the number of labeled nodes to be fixed for any chosen set S large enough. This means that although we have selected a large S, our algorithm effectively is using an smaller sized S because it is ignoring several of its nodes. To prove that updating the prior probabilities of the unlabeled nodes could achieve this, we have made another simulation in which we substitute the prior probabilities by the prediction error. For that we make a simulation every time we add 10 labels, get the true prediction error from the simulation, and set the probability error of the unlabeled nodes with the prediction error. So, if initially this probability error was set to $(N-1)/N$, after each simulation we set it to the prediction error only. We can see the result of this update in Figure 5.5. It is important to point that we have plotted the labeling error of just the labeled nodes instead of the labeling error of all the nodes. We see that for S sufficiently big, varying $|S|$ changes marginally the prediction error and does not change the labeling error at all. Which is the effect that we desired.

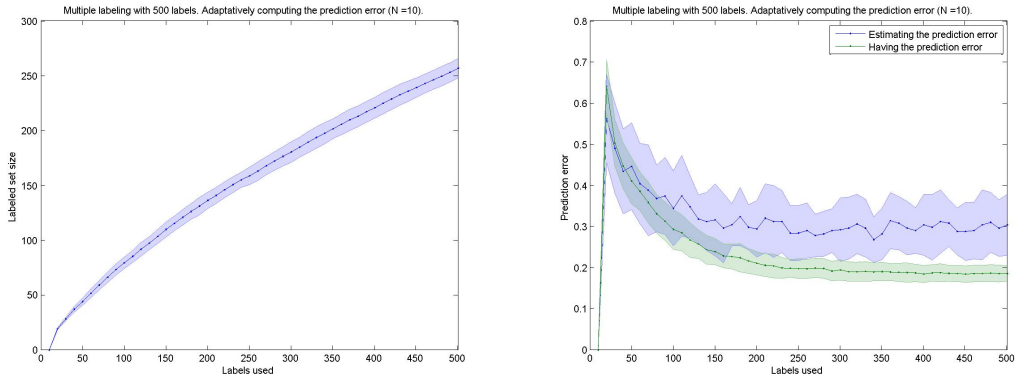


(A) Labeling error (B) Prediction error

FIGURE 5.5: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The labeling probability error of each worker is 0.2. The number of labels used is 500. We update the unlabeled nodes with the real prediction error every 10 labels.

However, this is just a theoretical simulation, because we cannot obtain the real prediction error without knowing beforehand the target of the dataset. In our third simulation we will try to estimate this prediction error adaptively, as we would have to do in practice. Because the size of the labeling set S does not affect the result if it is large enough in the theoretical simulation, we will impose its size to be half of the total size: $|S| = |S^C|$. We will monitor the evolution of the prediction error as we increment the number of labels and

we will compare it with our estimation of the prediction error. To estimate the prediction error, we will interpolate the rest of S from its labeled nodes each time 10 unlabeled nodes are labeled. We will compare these interpolations with the 10 new labeled nodes of the next iteration to estimate the prediction error. We take into account the probability of the new labeled nodes being wrong in our estimation. The results of this simulation are presented in figures 5.6, A.7 and A.8.



(A) Number of labeled nodes (B) Prediction error

FIGURE 5.6: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The labeling probability error of each worker is 0.2. The number of labels used is 500. We update the unlabeled nodes with the estimation of the prediction error every 10 labels. We compare the 10 newer labeled nodes with the interpolation generated without them

Figure 5.6a indicates how many labeled nodes we have as a function of the total number of labels. The interesting part of updating the probabilities of the unlabeled nodes is that its cost function is reduced, so use less labels on them. This reduces the total number of labeled nodes. As we can see in this subplot, we only label half of S although we have enough labels to label all S.

Figure 5.6b shows our estimation of the prediction error and the real prediction error. Our estimation is always worse than the real value (*'Having the prediction error'* line). We also cannot estimate the error if we do not have enough labeled nodes to compare, which explains the initial gap. If we compare the prediction error when using 500 labels with the prediction error of figure 5.3 for a labeling set size of $|S| = 500$, updating the probability of the unlabeled nodes improves this prediction error from $\tilde{2}4\%$ to $\tilde{1}8\%$. This improvement alone shows the significance of updating the unlabeled probabilities in the MNCA, giving us the improved MNCA (see Algorithm 6).

We see in figures A.7 and A.8 that changing the number of nodes checked to estimate the prediction error greatly reduces the variance of our estimation, at the cost of enlarging the initial gap. Checking more often does not make any significant improvement in the prediction error.

Chapter 6

Related and future work

The algorithms used for the selection of the subset of nodes and the model used for classification are similar to the ones used in [6]. Using the Gaussian Random Field as the first assumption for each of the assignation methods is closely inspired by [32]. However, both of these papers do not take into consideration the typical labeling uncertainty of the crowdsourcing schemes.

There are approaches to solve similar schemes to Problem 2 with a different approach than using GSP methods [39, 40, 41]. [39] uses support vector machines modified to be less affected by labeling noise, but does not consider that workers could have different quality and the number of nodes to be labeled using clustering is fixed before applying a majority voting based scheme. The fact that clustering is used instead of GSP-based sampling and that a prior in the sampling set is used, reduces greatly the efficiency of the algorithm respect to our work. This is shown in Figure 5.1, where we compare GSP with respect to using cluster representatives, and in Figure 5.3 respectively. In the latter we see that if we fix S and it is chosen too small or too large for a limited number of labels, we have a loss respect to the ideal S. However, we solve this problem with our improved MNCA algorithm (Algorithm 6).

There are other works that consider the crowdsourcing scheme, but do not use active learning or take into account the interrelation between data points [14, 42]. One of the works [42] answers how to be more efficient using multiple noisy labels in a single data point. Although they allow a way of estimating the quality of each worker, which our work does not, it does not give any improvement based in active learning that benefits from the dataset structure.

Finally, there are works that do not consider the crowdsourcing situation, where the workers could make mistakes in the labeling [6, 22, 25]. One of the works [22] uses harmonic functions and belief propagation on graphs in schemes of semi-supervised active learning. In other cases, the consider different crowdsourcing schemes, e.g. heterogeneus tasks where some worker have better expertise for particular tasks than the others [43].

To the best of our knowledge, not only or work solves efficiently Problem 2 as compared to other state of the art works but also there seems to be no literature about solving Problem 1. Generally, it is assumed that changing the order with we assign the workers to the samples in the single-label problem does not have any effect in the prediction error. That is, it is assumed that our

constrained problem has no optimal solution, but we have shown that this is not the case.

For future work, we can further improve the MNCA by, instead of imposing that the probability error of each node is the total prediction error, considering that each unlabeled node has different error probability (e.g., the nodes in the frontier between classes will be more noisy that those inside a cluster). Another interesting line of work would be to instead of assuming that the graph signal is obtained by a GRF (2.1), we could instead assume that its probability is of the form $p(f) \propto \exp(-\tau \sum v_{ij}|f_i - f_j|)$.. This is analogous to minimizing the L1-norm instead of the L2-norm, which could be useful in classification problems. Some of the expected advantages are robustness and sparsity. Other less significant improvements could be improvements to the transform from dataset to graph, e.g., using different σ_{ij} for each Gaussian kernel weight and considering them as hyperparameters [22].

Chapter 7

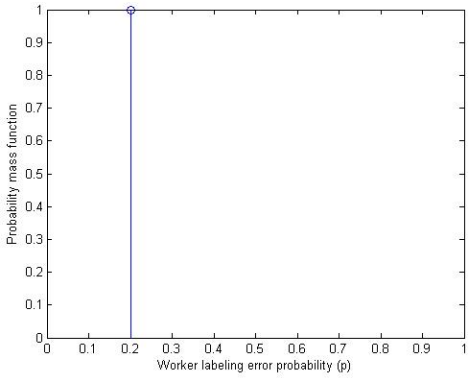
Conclusion

What we show in the first part of this work, that is, solving Problem 1 is that, not only there are cases where the assignment can affect the result, but also improves significantly the prediction error. We also expect that, for worse but faster algorithms that select the subset of nodes S than the one used in this study (see Algorithm 1), the assignment can play a bigger role in the overall prediction. This means that for a bigger number of worker error distributions than just the optimal one the improvement could be significant.

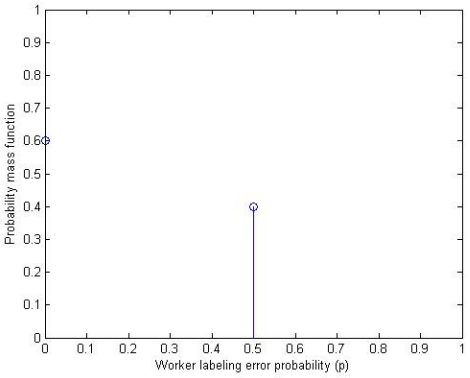
In the second part of the work, that is, solving Problem 2, we have showed that the IMNCA not only is always significantly better than the currently used methods in crowdsourcing but also can automatically take a decision between labeling new nodes or relabeling already labeled ones, proving to be useful in a real crowdsourcing system.

Appendix A

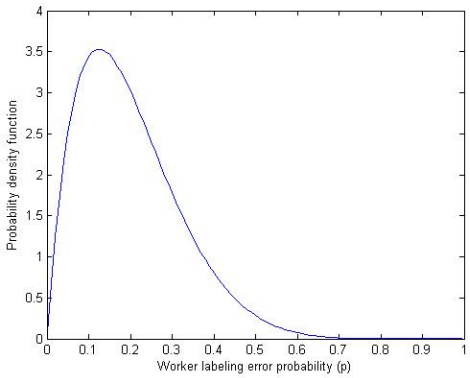
Additional simulation figures



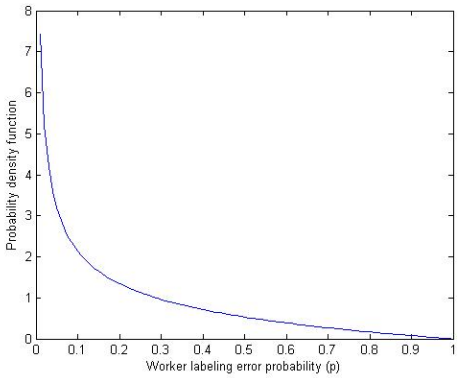
(A) *'Constant'* with $p = 0.2$.
Mean = 0.2, Std = 0



(B) *'Hammer-spammer'* with 40% of spammers.
Mean = 0.2, Std = 0.2449



(C) *'Beta'* with $\alpha = 2,\ \beta = 8$
Mean = 0.2, Std = 0.1206



(D) *'Beta'* with $\alpha = 0.5,\ \beta = 2$
Mean = 0.2, Std = 0.2138

FIGURE A.1: Plots of the probability mass/density functions of some of the used labeling error distributions.

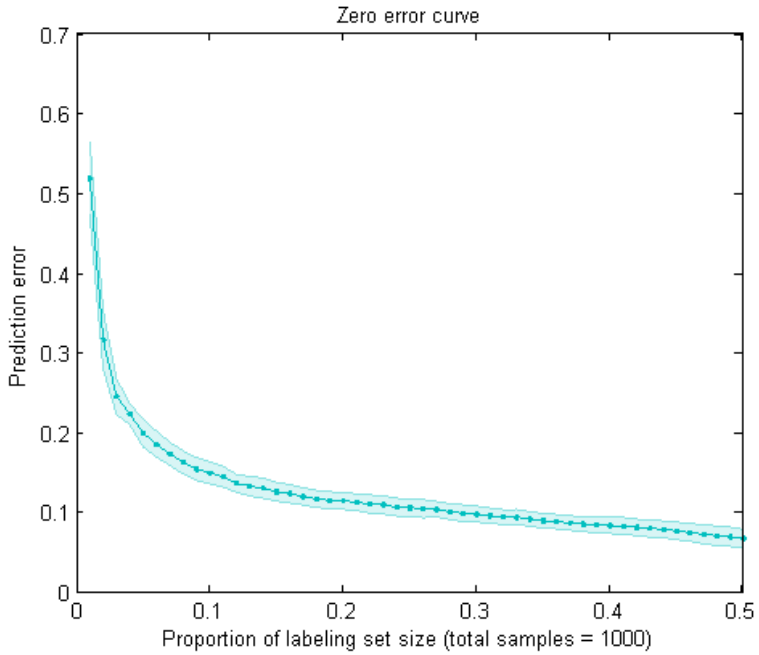
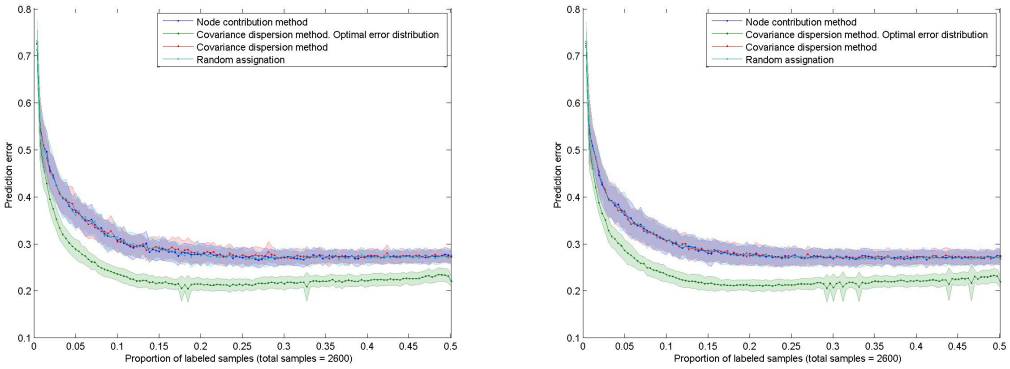


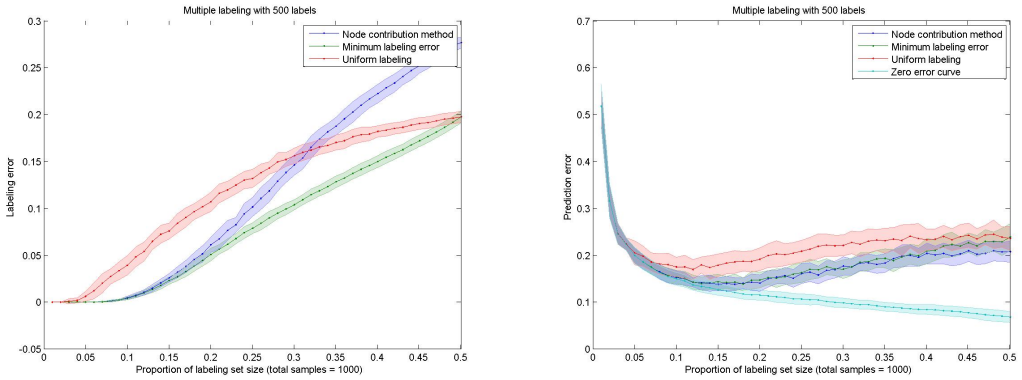
FIGURE A.2: Prediction error obtained when the workers have no labeling error. The results were obtained combining 100 different simulations. We used the USPS dataset.



(A) *'Hammer-spammer'* with 40% of spammers

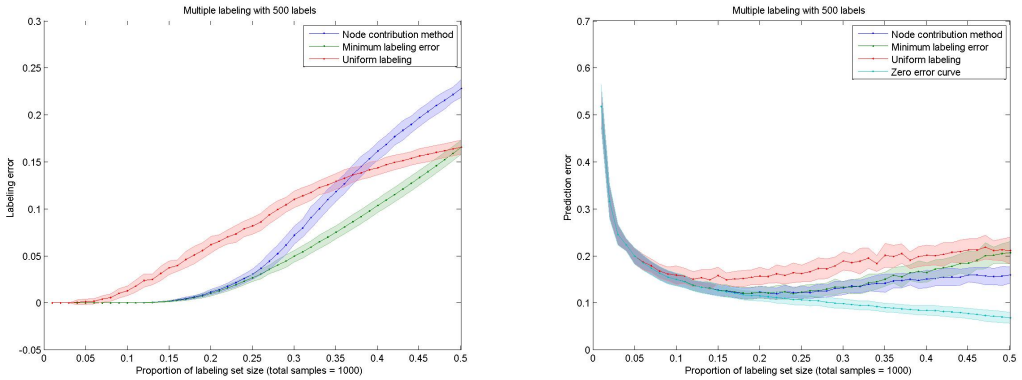
(B) *'Beta'* with $\alpha = 2,\ \beta = 8$

FIGURE A.3: Comparison between the prediction error obtained between the baseline (random assignation) and the assignations after using each proposed method. The results were obtained combining 100 different simulations. We used the Isolet dataset. All different distributions of error used in each of the Figures have mean = 0.2



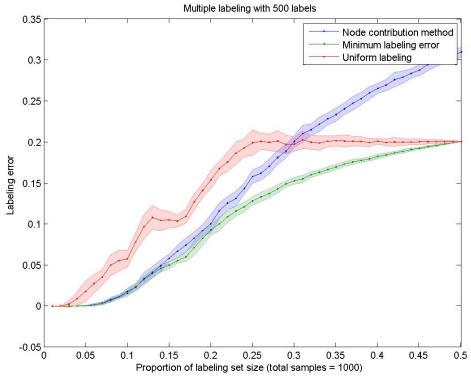
(A) Labeling error (B) Prediction error

FIGURE A.4: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The density function of the labeling probability error of each worker is a Beta(2,8). The number of labels used is 500.

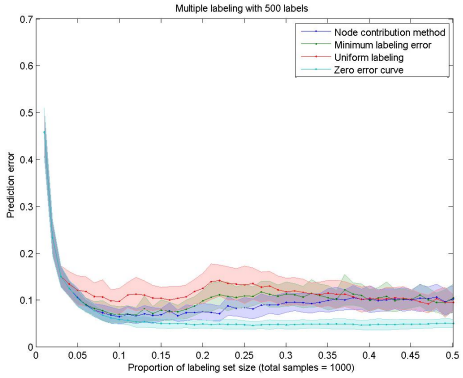


(A) Labeling error (B) Prediction error

FIGURE A.5: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The density function of the labeling probability error of each worker is a Beta(0.5,2). The number of labels used is 500.

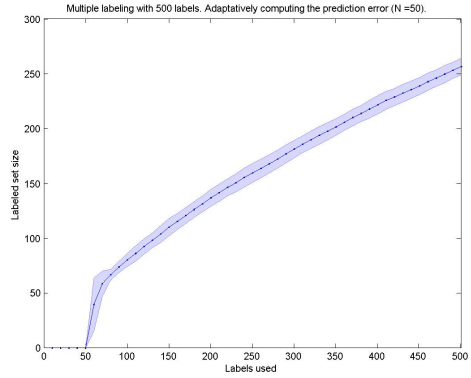


(A) Labeling error

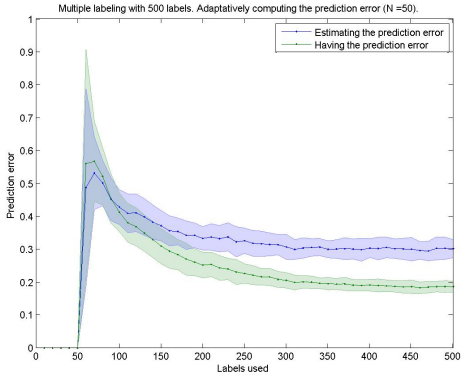


(B) Prediction error

FIGURE A.6: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset limiting the target graph frequency. The labeling probability error of each worker is 0.2. The number of labels used is 500. Previous to interpolating the labeling graph signal has been filtered to the cut-off frequency of the target graph signal.

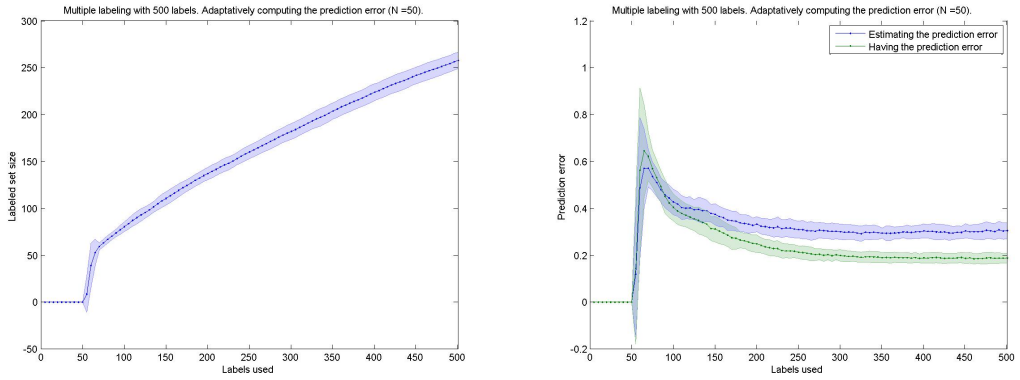


(A) Labeling error



(B) Prediction error

FIGURE A.7: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The labeling probability error of each worker is 0.2. The number of labels used is 500. We update the unlabeled nodes with the estimation of the prediction error every 10 labels. We compare the 50 newer labeled nodes with the interpolation generated without them



(A) Labeling error (B) Prediction error

FIGURE A.8: Comparison of the labeling and prediction error obtained between our proposed method and other conventional algorithms. The results were obtained combining 300 different simulations. We used the USPS dataset. The labeling probability error of each worker is 0.2. The number of labels used is 500. We update the unlabeled nodes with the estimation of the prediction error every 5 labels. We compare the 50 newer labeled nodes with the interpolation generated without them

Bibliography

[1] Nils J Nilsson. "Introduction to machine learning. An early draft of a proposed textbook". In: (1996).

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org.

[3] Bernd Heisele, Purdy Ho, and Tomaso Poggio. "Face recognition with support vector machines: Global versus component-based approach". In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*. Vol. 2. IEEE. 2001, pp. 688–694.

[4] Simon Tong and Daphne Koller. "Support vector machine active learning with applications to text classification". In: *Journal of machine learning research* 2.Nov (2001), pp. 45–66.

[5] Yi Yang et al. "Multi-class active learning by uncertainty sampling with diversity maximization". In: *International Journal of Computer Vision* 113.2 (2015), pp. 113–127.

[6] Akshay Gadde, Aamir Anis, and Antonio Ortega. "Active semi-supervised learning using sampling theory for graph signals". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 492–501.

[7] Faiza Khan Khattak and Ansaf Salleb-Aouissi. "Quality control of crowd labeling through expert evaluation". In: *Proceedings of the NIPS 2nd Workshop on Computational Social Science and the Wisdom of Crowds*. Vol. 2. 2011.

[8] Justin Cheng, Jaime Teevan, and Michael S Bernstein. "Measuring crowdsourcing effort with error-time curves". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. 2015, pp. 1365–1374.

[9] Gabriele Paolacci, Jesse Chandler, and Panagiotis G Ipeirotis. "Running experiments on amazon mechanical turk". In: (2010).

[10] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. "Amazon's Mechanical Turk a new source of inexpensive, yet high-quality, data?" In: *Perspectives on psychological science* 6.1 (2011), pp. 3–5.

[11] JCF De Winter et al. "Using CrowdFlower to study the relationship between self-reported violations and traffic accidents". In: *Procedia Manufacturing* 3 (2015), pp. 2518–2525.

[12] John Suler. "The online disinhibition effect". In: *Cyberpsychology & behavior* 7.3 (2004), pp. 321–326.

[13] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. "Quality management on amazon mechanical turk". In: *Proceedings of the ACM SIGKDD workshop on human computation*. ACM. 2010, pp. 64–67.

[14] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. "Get another label? improving data quality and data mining using multiple, noisy labelers". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 614–622.

[15] Wei Tang and Matthew Lease. "Semi-supervised consensus labeling for crowdsourcing". In: *Special Interest Group on Information Retrieval 2011 Workshop on Crowdsourcing for Information Retrieval, Beijing, China, July*. Vol. 28. 2011, pp. 1–6.

[16] Xiaojin Zhu. "Semi-supervised learning". In: *Encyclopedia of machine learning*. Springer, 2011, pp. 892–897.

[17] David D Lewis and William A Gale. "A sequential algorithm for training text classifiers". In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc. 1994, pp. 3–12.

[18] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. "Active hidden markov models for information extraction". In: *International Symposium on Intelligent Data Analysis*. Springer. 2001, pp. 309–318.

[19] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. "Query by committee". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 287–294.

[20] Andrew Kachites McCallumzy and Kamal Nigamy. "Employing EM and pool-based active learning for text classification". In: *Proc. International Conference on Machine Learning (ICML)*. Citeseer. 1998, pp. 359–367.

[21] Burr Settles, Mark Craven, and Soumya Ray. "Multiple-instance active learning". In: *Advances in neural information processing systems*. 2008, pp. 1289–1296.

[22] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions". In: *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*. Vol. 3. 2003.

[23] Ntorina Thanou. "Graph Signal Processing". In: (2016).

[24] David I Shuman et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98.

[25] Avrim Blum and Shuchi Chawla. "Learning from labeled and unlabeled data using graph mincuts". In: (2001).

[26] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. "Semi-supervised learning using gaussian fields and harmonic functions". In: *ICML*. Vol. 3. 2003, pp. 912–919.

[27] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples". In: *Journal of machine learning research* 7.Nov (2006), pp. 2399–2434.

[28] Leonard Kaufman and Peter J Rousseeuw. "Partitioning around medoids (program pam)". In: *Finding groups in data: an introduction to cluster analysis* (1990), pp. 68–125.

[29] Holger Rootzén. *Some probability and statistics*. http://www.math.chalmers.se/~rootzen/highdimensional/SSP4SE-appA.pdf. [Online; accessed 27-November-2016]. 2013.

[30] Georg Lindgren, Holger Rootzén, and Maria Sandsten. *Stationary stochastic processes for scientists and engineers*. CRC press, 2013.

[31] Philip E Gill, Walter Murray, and Margaret H Wright. "Practical optimization". In: (1981).

[32] Akshay Gadde and Antonio Ortega. "A probabilistic interpretation of sampling theory of graph signals". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 3257–3261.

[33] Daniel Berend and Aryeh Kontorovich. "Consistency of weighted majority votes". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3446–3454.

[34] O Chapelle, B Schölkopf, and A Zien. "Semi-Supervised Learning". In: (2006).

[35] Anja Struyf, Mia Hubert, Peter Rousseeuw, et al. "Clustering in an object-oriented environment". In: *Journal of Statistical Software* 1.4 (1997), pp. 1–30.

[36] Hieu T Nguyen and Arnold Smeulders. "Active learning using pre-clustering". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 79.

[37] Sunil K Narang et al. "Localized iterative methods for interpolation in graph structured data". In: *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE. 2013, pp. 491–494.

[38] Victor Klee and George J Minty. *How good is the simplex algorithm*. Tech. rep. DTIC Document, 1970.

[39] Liyue Zhao, Gita Sukthankar, and Rahul Sukthankar. "Incremental relabeling for active learning with noisy crowdsourced annotations". In: *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*. IEEE. 2011, pp. 728–733.

[40] Pinar Donmez and Jaime G Carbonell. "Proactive learning: cost-sensitive active learning with multiple imperfect oracles". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM. 2008, pp. 619–628.

[41] Pinar Donmez, Jaime G Carbonell, and Jeff Schneider. "Efficiently learning the accuracy of labeling sources for selective sampling". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 259–268.

[42] David R Karger, Sewoong Oh, and Devavrat Shah. "Iterative learning for reliable crowdsourcing systems". In: *Advances in neural information processing systems*. 2011, pp. 1953–1961.

[43] Jennifer Wortman Vaughan. "Adaptive task assignment for crowdsourced classification". In: (2013).