Felipe Nunes Walmsley

Um Método de Geração de Pools de Classificadores Usando Instance Hardness

Recife

Julho, 2017

Felipe Nunes Walmsley

Um Método de Geração de Pools de Classificadores Usando Instance Hardness

Trabalho de Gradução

Universidade Federal de Pernambuco – UFPE

Centro de Informática

Graduação em Engenharia da Computação

Supervisor: George Darmiton da Cunha Cavalcanti

Recife

Julho, 2017

Felipe Nunes Walmsley

Um Método de Geração de Pools de Classificadores Usando Instance Hardness

Trabalho de Gradução

Trabalho aprovado. Recife, 12 de Julho de 2017:

Prof. Dr. George Darmiton da Cunha Cavalcanti
Orientador

Prof. Dr. Paulo Salgado Gomes de Mattos Neto
Avaliador

Recife
Julho, 2017

To my mother, who guided me through all of my eduaction, and all of life so far.
To the memory of my father, who I am sure would have loved to see his son graduate from college.

Acknowledgements

I'd like to thank my advisor, Prof. George Darmiton for his guidance and instruction, and for his thoroughness and attention to detail, which have made this work far better than it would have been otherwise.

I'd also like to thank my coworkers for their support. In particular, I'd like to thank Hector Pinheiro, José Ivson Silva and Vítor Antero, for helping me wrap my head around some pesky details of the statistical methods used in this work, and also for their help with data normalization procedures.

Finally, I would like to thank my parents, for putting my education first, regardless of circumstances.

I met a traveller from an antique land
Who said: Two vast and trunkless legs of stone
Stand in the desert... near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them and the heart that fed:
And on the pedestal these words appear:
'My name is Ozymandias, king of kings:
Look on my works, ye Mighty, and despair!'
Nothing beside remains. Round the decay
Of that colossal wreck, boundless and bare
The lone and level sands stretch far away.
(Ozymandias - Percy Byshe Shelly)

Abstract

In Machine Learning, ensemble methods have been receiving a great deal of attention. Techniques such as Bagging and Boosting have been successfully applied to a variety of problems. Nevertheless, such techniques are still susceptible to the effects of noise and outliers, which might be present in the training data. Both outliers and noisy instances are intrinsically likely to be misclassified, regardless of the choice of classifier. We propose a new method for the generation of pools of classifiers, based on Bagging, in which the probability of an instance being selected during the resampling process is inversely proportional to its instance hardness. The goal of the proposed method is to remove noisy data without sacrificing the hard instances which are likely to be found on class boundaries. We evaluate the performance of the method in fifteen public data sets, and compare it to the performance of the Bagging and Random Subspace algorithms. Our experiments show that the accuracy of the proposed method is at least as good as that of the original Bagging algorithm, the second best performing algorithm in our test, and in some cases is significantly better than that of Bagging.

Key-words: ensemble methods. Bagging. noisy data. instance hardness.

List of Figures

List of Tables

List of abbreviations and acronyms

kDN	k-Disagreeing Neighbors
kNN	k-Nearest Neighbors

List of symbols

α	Lowercase greek letter Alpha - Significance level
$\in$	Set membership relation

Contents

1 Introduction

1.1 Motivation

In Machine Learning, ensemble methods [1] are techniques that combine multiple predictors trained independently, using a combination of the outputs of each predictor as the final output. This is in contrast to traditional Machine Learning methods, which train a single classifier on the whole of the training set. The rationale behind this shift in paradigm is that, by utilizing ensemble techniques one can expect to obtain a pool of predictors with complementary competences. The pools generated can thus obtain performances gains over strategies that employ a single classifier, given that finding the single optimal model for a problem may be exceedingly difficult.

One ensemble learning method that has enjoyed widespread adoption is the bootstrap aggregating algorithm [2], or simply, Bagging. The algorithm relies on creating an ensemble of N classifiers trained on N training sets, sets which are created from the original training set. These training sets are generated by sampling uniformly and with replacement from the original training set.

The use of Bagging is interesting when the data sets available are small, noisy, or both [3]. In general terms, it can be expected that the classifiers produced by the Bagging algorithm will have complementary competences, making the decisions of the system better than those of a single classifier trained on the whole training set [2].

The main motivation behind the work presented here is that, while ensemble learning may offer performance gains, they can't completely avoid two common problems in Machine Learning: noise and outliers. Simply described, outliers are those examples that are considerably different from most members of its class, while noisy instances are instances that have had its value changed from its true value. It is important to note that when noise is discussed in this work, we mostly mean sources of noise that cannot be removed by calibration.

We focus on noise and outliers due to their influence on classifiers. It is possible that in the presence of noise, outliers, or both, the training process of a classifier can become unstable or prone to overfitting [4], regardless of the use of ensemble techniques. This is particularly concerning in the case of algorithms that place greater weight on misclassified instances during the training process, such as AdaBoost [5], since it's possible that the model will be strongly adjusted so as to correctly classify instances that do not represent the underlying distribution of the data. In these scenarios, one would expect to observe a significant decrease in the generalization accuracy of the model.

Nevertheless, there are techniques that seek to remove noisy instances from a data set, as a means to alleviate the previously mentioned problems. One such technique is the Edited Nearest Neighbor Rule [6], which filters the data set, removing those instances that are not correctly classified using a k-Nearest Neighbors classifier. Still, noise removal techniques might cause undesired side effects on the training set, such as the removal of examples that are not noise or a dramatic removal of examples in the boundaries between classes.

Parallel to the concept of noise we have the concept of instance hardness, or the difficulty in classifying an instance. The hardness of an instance can be understood as the likelihood that it will be incorrectly classified [7]. The difficulty in classifying an instance may be used as proxy to the probability that such instance is noisy or an outlier. In this manner, instance hardness measures may be used as a foundation for techniques which aim to selectively remove instances from the training set.

1.2 Objective

Armed with the concepts of outliers, noise and instance hardness, it is natural to question whether it would be possible to use some measure of instance hardness to remove the troublesome instances mentioned previously from a training set. One would expect that once those instances are removed, the training process would become more stable, leading to better generalization accuracy.

In this vein, we propose in this work a method based on Bagging, which seeks to remove outliers and noisy instances from the training set, while still preserving instances which are not truly noise, or instances that are on the border of classes. We decided to base our work on the Bagging algorithm due to its superior performance in noisy scenarios, as explained previously.

The core idea of this method is to modify the process by which the bootstrapped training sets are created in the Bagging algorithm. Instead of picking examples with uniform probability, the probability of an instance being picked is now defined to be inversely proportional to its hardness.

In order to illustrate the concepts discussed so far, Figure 1.2.1 shows a binary classification problem in which the data are subject to noise. The examples in blue belong to the negative class, while the examples in red belong to the positive class. The examples in green are also part of the negative class, though they are distributed differently from the rest of the examples. This illustrates the concept of outliers.

The examples follow a uniform random distribution. The negative examples are distributed in the interval (0;1) in the x-axis and in the interval (0;1) in the y-axis. The

examples in the positive class are distributed in the interval (1.5;2.5) in the x-axis and in the interval (-0.5;0.5) in the y-axis. Any example labeled as positive which does not follow the distribution defined for the positive class is incorrectly labeled, and therefore is noise. The same is true for the negative class. Noise was added to 20% of the instances to demonstrate its effects.

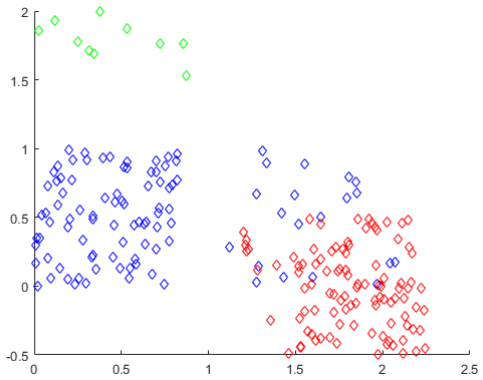

Figure 1.2.1 – Noisy distribution. The examples in blue belong to the negative the class, while the examples in red belong to the positive class. The examples in green are outliers belonging to the negative class.

Figure 1.2.2, on the other hand, illustrates the result of an ideal noise removal technique.

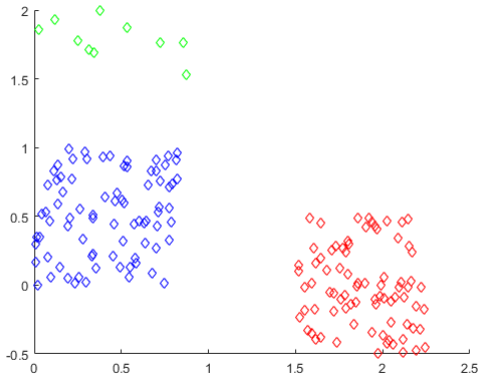

Figure 1.2.2 – Ideal noise removal.

Once the noise is removed, it becomes clear that the problem is linearly separable, and thus, easily learnable.

The proposed method was conceived with this ideal noise removal scheme in mind. It is necessary to note, though, that in our method, the instances are selected in a probabilistic manner. This means that even hard instances may be selected. It is indeed desirable

that hard instances may be selected, since they may be instances in the border between classes.

1.3 Methodology

In order to asses the effectiveness of the proposed method, we compared it with well-established ensemble learning methods. We added artificial noise to the class labels of a data set, and then trained classifiers on this noisy data. The accuracy of each method was measured in several public datasets, and a statistical analysis of the results was performed, in order to ascertain whether or not the proposed method achieved superior generalization accuracy.

Furthermore, we conducted an analysis of the distribution of instance hardness values for noisy and non-noisy instances, on the same data sets which were used to evaluate the accuracy of the classifiers. This analysis was aimed at determining precisely how the noise would affect the hardness of the instances, beyond the simple assumption that noisy instances are harder.

1.4 Structure of this document

This work is organized as follows. In Chapter 2, we present basic concepts pertaining to ensemble methods, noise and outliers, data complexity measures and instance hardness. We also present related works which will serve as a foundation upon which our work is built, and also as a benchmark for comparison with our method. In Chapter 3, we detail the proposed method, giving precise definitions for our method for determining instance selection probabilities, as well as presenting each step of our pool generation algorithm. In Chapter 4, we further explain our experimental methodology, including the data sets used and the statistical methods applied. We then present our results and discuss their implications. Finally, in Chapter 5 we present a summary of our findings and suggest possible future lines of investigation.

2 Background and Related Works

In this chapter, we present an overview of the concepts utilized throughout this document, as well as the theoretical foundation upon which our work is built. We also use this chapter to give a brief summary of other works related to our own.

2.1 Ensemble methods

Ensemble methods, classifier ensembles or multiple classifier systems are a family of Machine Learning methods that focus on creating systems composed of multiple classifiers, with the objective of achieving better performance than what would be possible with a single classifier.

In these systems, multiple classifiers are trained on the same training set, or on partitions of the same training set. The final output of the system is given by some combination of the outputs of the individual classifiers. The core idea that guides ensemble methods is that systems composed of diverse classifiers, with complementary strengths, can achieve better performance than individual classifiers.

In [8], the authors propose a taxonomy of multiple classifier systems. This taxonomy is based on the three phases of multiple classifier systems:

1. **Pool generation:** The process of generating the pool (the set) of classifiers that will be part of the system. The authors distinguish between homogeneous and heterogeneous systems, depending on whether all classifiers in the system are of the same type.

2. **Selection:** The selection of which classifiers will be used to calculate the output of the pool. In dynamic classifier selection schemes, a subset of the pool is chosen to predict the class of each test instance. Under static selection schemes, once the final pool is chosen, it remains unchanged. It can be said that some methods eschew selection altogether, since all trained classifiers are included in the final pool.

3. **Integration:** The combination of the outputs of each classifier in the pool to give the final classification of an instance. This can be done by combining either the predicted class labels, or by combining the predicted class likelihoods for each class, when available.

Our work focuses on the pool generation phase. In the taxonomy of [8], our work would be classified under homogeneous, data set based pool generation methods.

2.1.1 Diversity in ensembles

As mentioned previously, one of the central assumptions of ensemble methods is that the classifiers being generated are diverse amongst each other. A detailed exposition on the importance of diversity and the different methods which can be used to ensure diversity is presented in [9].

In [9], the authors argue that in data sets with small or very large amounts of data, multiple classifiers systems would be able to perform better than single classifiers - in the former case by taking advantage of bootstrapping methods, and in the latter case by partitioning the training set, and training different classifiers in each subset.

Furthermore, the authors posit that training different classifiers with different initializations might help the system in overcoming local optima in the training process.

While these first two arguments might appeal to intuition, the authors in [9] point out that there are proofs in the regression literature relating the diversity of ensembles and their generalization error, such as in [10] and [11]. There are also analytical results that show that classifier ensembles outperform any individual member of the pool [12], under certain assumptions. Nevertheless, the authors of [9] also concede that there's no universally agreed upon measure of diversity for classifier ensembles, and no analytical results that directly relate ensemble diversity and classification accuracy.

2.1.2 Seminal ensemble algorithms

In [9], the authors list two major topologies for multiple classifier systems - parallel and serial topologies. The authors also point out that parallel topologies are predominant in the literature.

Parallel topologies are defined as those where every classifier is fed the same input data, and the output of the system is given by weighing the outputs of each classifier. In contrast, serial topologies are those where the inputs are fed to the classifiers in sequence. It might be case in serial topologies that the next classifier in the sequence is only used if some condition is met. Therefore, there is no guarantee that all classifiers in the pool will be used.

The most frequently applied algorithms based on sequential topologies are Boosting algorithms [13]. In particular, the AdaBoost algorithm [5] enjoys widespread adoption. The central idea of the AdaBoost algorithm is to train in multiple steps a set of predictors, in which each predictor is specialized on the examples for which the previously trained predictor has the highest error.

Turning our attention to parallel topologies, we can cite the Random Subspace algorithm, proposed by Ho in [14]. The Random Subpace algorithm partitions the feature

set of the examples, training individual classifiers for different subspaces of the feature set.

Finally, we discuss the method of bootstrap aggregating (Bagging), first developed by Breiman [2] as a method to develop robust ensembles based on weak learners. Our work is based on the Bagging algorithm.

The original Bagging algorithm creates m new training data sets from an original training data set T. Each new training set is created by sampling examples uniformly and with replacement from T. The bootstrapped training sets are then used to train m instances of a base predictor (e.g. a Perceptron or a Decision Tree), one for each of the generated sets. These predictors are pooled together in the testing phase as one single predictor, subject to some voting scheme chosen by the user. The Bagging algorithm aims to create a pool of predictors which are competent in different regions of the feature space.

2.1.2.1 Adaptations to Bagging

The success of Bagging has lead to many modified versions of the original method have been proposed, mostly to deal with shortcomings of the original algorithm under specific scenarios.

Breiman himself proposed in [15] a modification to Bagging aimed at reducing the error of regressors. The modified algorithm relies on repeatedly applying Bagging to the same training set, but each time training the pools on the residual errors of the instances not chosen by the bootstrapping process in the previous step.

Several works based on Bagging are focused on dealing with the problem of imbalanced data sets. Li proposes in [16] partitioning the data points not belonging to the minority class into disjunct, equal sized sets, with as many elements as the minority class. These sets are then each used to train a different classifier, which are combined into a pool. The majority vote rule is used to decide the output of the pool.

In the same vein, Wang and Yao introduced in [17] SMOTEBagging. SMOTEBagging uses SMOTE [18] to generate several training sets on which to train the pool of classifiers. Each new training set is first balanced, both by repeated sampling from the minority class and generation of new instances through SMOTE.

Another technique that makes use of SMOTE is SMOTE-ICS-Bagging, proposed by Oliveira et al. in [19]. The technique is based on ICS-Bagging, a method proposed in the same paper. ICS-Bagging generates at each training step a pool of K classifiers trained using Bagging. However, it only adds to the final pool one classifier per step, the one classifier amongst all K that when added to the pool results in the highest overall fitness of the pool, where the fitness of the pool is defined as a weighted average between the accuracy and the diversity of the pool.

2.2 Ensemble learning, noise and outliers

The work presented here deals extensively with noise. Another recurring theme in this document is the concept of outliers. Therefore, it is important that we clarify our definitions of both noise and outliers.

The first definition we need to present is that of an outlier. An outlier is considered to be a instance that is very different from the other members of its class. Since outliers are not similar to most members of their class, they can be difficult to correctly classify.

Noise, on the other hand, is understood to be a data point that has had its value changed from its original value. The most common processes by which the value of a data point might be modified are mistakes in a measurement or an inherent uncertainty in the measurement. In either case, we are referring to a change in the value of the data point which is not due to systematic error, and therefore cannot be removed via calibration.

While the concepts of noise and outlier may at first sight appear somewhat similar, in this work we call an outlier those examples that, though they are different from other examples in their class, are still part of the underlying distribution. Noise, however, is that which does not belong to the actual underlying distribution.

In this work, we focus on label noise, as opposed to noise that acts on the features of instances. By label noise we mean any process that changes the label of an instance (as presented to a learning algorithm) from its true value. A rather comprehensive treatment of label noise can be found in [20]. We adopt their taxonomy of label noise types, which can be summarized as follows:

1. **Noisy completely at random:** The process which changes the label of an instance has no correlation to either the label or the features of the instance.
2. **Noisy at random:** The process which changes the label of an instance is dependent on the true label of the instance.
3. **Noisy not at random:** The process which changes the label of an instance is dependent on both the true label of the instance and the features of the instance.

Our work mainly deals with the first type of noise, as instances are randomly selected to have their labels changed, and the new label is also chosen at random from the set of possible labels in the data set (the original label being excluded).

2.2.1 The effects of noise

In [20], the authors summarize published results which suggest that label noise adversely affects both the performance of models and their complexity. The most relevant

results are briefly discussed in this section.

Okamoto and Nobuhiro establish in [21] theoretical results which relate the level of noise present in a training set and the performance of a k-Nearest Neighbors classifier. Bi and Jeske show in [22] that normal discriminant analysis and logistic regression are also affected by noise, though to a smaller degree.

There are also empirical results regarding the effects of noise on different learning algorithms. In [23], the authors investigate the effect of label noise on regularized linear regression methods (logistic regression, ridge regression and Support Vector Machines), and conclude that all the methods investigated are strongly affected by the presence of label noise.

Of particular interest to our work are the results that relate the presence of label noise with the performance of Boosting algorithms [24], [25], [4]. Not all Boosting algorithms are affected by the presence of noise in the same way, but a common theme in these works is the clear negative effect of noise on the performance of the AdaBoost algorithm.

Finally, it is interesting to point out that in [24], the authors conclude that in situations in which a substantial level of noise is present, the Bagging algorithm tends to outperform Boosting algorithms.

2.3 Data complexity measures and Instance Hardness

The evidence so far presented shows that some situations might make the process of training a classifier harder, and may result in a diminished generalization performance. In particular, we point out the situations in which there is class imbalance, noise, outliers or some combination of these factors. In [7], Smith et al. investigate which instances tend to be classified incorrectly, and how these instances affect the data set as a whole. These instances would be considered “hard” instances.

Once again in [7], the authors discuss works which deal with data complexity measures, such as the work in [26], which uses complexity measures on the data set to evaluate situations in which one learning algorithm might outperform another. Nevertheless, Smith et al. concede that there is no universally accepted data complexity measure.

In [7], the authors note that most data complexity measures are calculated over the whole data set, ignoring the fact that some instances may very well be harder than others. For example, in [27], the authors note that instances on the border between classes are inherently harder to classify correctly. These instances are nevertheless fundamental, since they are necessary for the classifiers to be able to correctly learn the separation between classes.

In this context, Smith et al. propose measures which evaluate the difficulty of

a instance on an individual level, called its instance hardness. The authors adopt an empirical definition of instance hardness, which corresponds to the probability that an instance will not be classified correctly. Several measures are proposed to quantify the hardness of an instance, among them measures based on the class likelihoods predicted by a k-Nearest Neighbors classifier, model complexity measures, and the relations between the number of examples in each class of the problem.

3 Proposed Method

As detailed in the previous chapter, the method proposed in this work is based on Bagging [2].

Our motivation in proposing changes to Bagging is that, while it may offer accuracy gains, its sampling process is still subject to the effects of noisy data or outliers. Since examples are drawn uniformly to create the new training sets, noisy examples and outliers are as likely to be picked as examples which are neither noise nor outliers. Therefore, it is possible that some of the bootstrapped training sets might have a high proportion of noise, outliers or both. When presented with such training data, some classifiers may either overfit the noisy data or to fail to learn at all, thereby reducing the generalization accuracy of the final system.

To alleviate this issue, we propose a modification to the sampling process used in the Bagging algorithm. Our motivations in proposing this new method are twofold:

1. We would like to avoid adding noisy examples and outliers to the bootstrapped sets too frequently.

2. We also would like to avoid completely removing "hard" instances, as they might be instances on the boundary of classes, which are naturally hard. Removing these instances might mean losing information about such boundaries.

Thus, we adopt a probabilistic approach. We use the concept of Instance Hardness to define the probability of including an instance from the training set in one of the bootstrapped sets. More specifically, we use the k-Disagreeing Neighbors measure, introduced in [7] as a measure of the hardness of the instance. Our reasons for choosing the k-Disagreeing Neighbors measure will become clear shortly, once we have defined and explained it.

The k-Disagreeing Neighbors (kDN) measure is defined as the fraction of the k nearest neighbors of a sample that do not share its class label. Formally, the kDN hardness $kDN(x)$ of an instance x , whose k nearest neighbors are denoted by $kNN(x)$, is defined as:

$$kDN(x) = \frac{|x' \mid x' \in kNN(x) \wedge label(x') \neq label(x)|}{k} \tag{3.1}$$

Where $label(x)$ is the class label of example x.

From the definition, $kDN(x)$ takes on values in the interval $[0, 1]$, evenly spaced by $\frac{1}{k}$.

The k-Disagreeing Neighbors measure was chosen as it is easily interpretable, since its value can be understood by simply inspecting the neighborhood of the instance, and is consistent with the intuitive notion of how much an instance "fits in" with its neighbors. Furthermore, the kNN classifier used in measuring the instance hardness does not require training, a step which could possibly increase the the cost of measuring the hardness.

Armed with the definition of the kDN measure, we can now present our method, shown below in pseudocode.

Algorithm 1: The pool generation algorithm

Input : The training set T
The pool size m
The bootstrapped set size n_b
The base predictor C
The value of k for the kDN measure
A boolean p_{type}

Output: The trained pool P

```
1  begin
2      Initialize the pool P as the empty set
3      foreach x ∈ T do
4          Calculate kDN(x)
5      end
6      foreach x ∈ T do
7          p(x) = normalize(x, T, p_type) ;    /* The selection probability of the
             instance */
8      end
9      for i from 1 to m do
10         Initialize the training set T_i as the empty set
11         for j from 1 to n_b do
12             Add an instance x_j ∈ T to T_i, sampled with replacement according to p
13         end
14         Train the classifier C_i (an instance of C ) using T_i
15         Add C_i to the pool P
16     end
17 end
```

Lines 3 to 5 of Algorithm 1 show the process of calculating the hardness of the instances, according to (3.1).

Our primary aim in measuring the hardness of an instance is to pick "hard" examples with a smaller probability than that of "easy" examples during the bootstrapping process, as shown in lines 6 to 8 of Algorithm 1. The normalization procedure will be explained separately, once we have presented the two methods for calculating selection probabilities.

The first method attributes probabilities which are inversely proportional to the instance hardness. Let n be the number of examples in the training set T. We define the function $f(x_i)$ of an instance $x_i \in T$:

$$f(x_i) = \frac{1}{n} + (1 - kDN(x_i)) \tag{3.2}$$

The first term in (3.2) attributes a sort of "uniform probability" to all instances of the training set. This ensures that even instances with a kDN value of 1 are given a chance to be selected, as an attempt to avoid the issue of instances that are altogether discarded. The rationale behind this decision is to preserve possible hard instances in the boundary of classes.

We then normalize the value of $f(x_i)$ by the sum over all $x_i \in T$ to obtain the probability of the instance being selected, $p_{linear}(x_i)$:

$$p_{linear}(x_i) = \frac{f(x_i)}{\sum_{i=1}^{n} f(x_i)} \tag{3.3}$$

The normalization is used to ensure that p_{linear} is a proper probability distribution. This is necessary for the proper functioning of computational implementations of the procedure of chosing with replacement.

Our second method for attributing probabilities explores the use of the Softmax function [28] as an alternative means to normalize the probability.

The Softmax function was adopted since it is widely used as a means to turn the output of a learning machine into class probabilities, most often in the context of Neural Networks. We define the Softmax probability $p_{softmax}(x_i)$ of an instance being selected as:

$$p_{softmax}(x_i) = \frac{e^{1-kDN(x_i)}}{\sum_{i=1}^{n} e^{1-kDN(x_i)}} \tag{3.4}$$

In this formulation, the issue of instances with null probability is naturally resolved, since the Softmax function only takes on a value of zero in the limit where k goes to infinity.

Armed with these definitions, we can now present the normalization procedure.

Algorithm 2: The instance selection probability normalization procedure

Input : An instance x The training set T

Output: The selection probability p(x)

```
1 begin
2     if p_type then
3         Calculate p_linear(x) ;   /* The selection probability of the instance
            */
4     else
5         Calculate p_softmax(x)
6     end
7 end
```

Depending on the value of p_{type}, we normalize the selection probabilities according to either (3.3) or (3.4).

Once these probabilities have been calculated, we proceed to generate the bootstrapped training sets as in the original Bagging algorithm. Drawing with replacement from a probability distribution given by either (3.3) or (3.4) , m bootstrapped training sets are generated (lines 11-13 of Algorithm 1), and m instances of a base predictor are trained (lines 14-15 of Algorithm 1).

It is important here to note that while the original Bagging algorithm was concerned with predictors in general, i.e. both classifiers and regressors, the work here presented is focused on classifiers. This means that we are dealing with pools of classifiers, and the output of the ensemble at test time will be a class label.

It should be noted that this restriction is due to our choice of hardness measure. It should be possible to adapt the method for regression, by using a measure of instance hardness that does not depend on the class of the instance as the ones in [7] do.

At test time, the predicted class label for an instance is calculated as follows:

Algorithm 3: The pool generation algorithm

Input : The trained pool P
The size of the pool m
A test instance x
A voting scheme V for the predictions of the pool

Output: The predicted class label of x, $y_{pred}(x)$

```
1 begin
2     Initialize the set of pool predictions Y to the empty set
3     for i from 1 to m do
4         Calculate the class label y_i(x) of x predicted by classifier C_i in the pool.
5         Add y_i(x) to Y
6     end
7     Calculate y_pred(x) as the result of V(Y)
8 end
```

Each classifier in the trained pool outputs a class prediction for the test instance x. The predictions are then weighed under some voting scheme (e.g. Majority Vote) to give the final output of the pool, $y_{pred}(x)$. The value $y_{pred}(x)$ is the one considered when evaluating the performance of the pool.

4 Experiments and Discussion

In this chapter, we compare the performance of the proposed method against that of commonly used pool generation methods. We also evaluate the effect of noise on the hardness of an instance.

4.1 Experimental design

In order to assess the effectiveness of pool generation methods, we compare their accuracy on several public data sets, which will be described shortly. The proposed method was compared with the Bagging and Random Subspace algorithms. In order to obtain a baseline, the performance of a single instance of the base classifier chosen to compose the pools was also evaluated. Furthermore, both variations of the procedure for determining selection probabilities were considered; these were treated as separate methods and also compared with each other.

We adopted a 5-fold cross-validation approach to partition the data sets, with 4 of the folds in each partition being used for training and the last for testing. The mean accuracy over the folds for each pool generation algorithm was measured. The cross-validation procedure was repeated 10 times, and the average and standard deviation of the mean accuracy were measured.

4.1.1 Experimental parameters

For all the pool generation methods, one must specify the following parameters:

- The pool size m.
- The base classifier C.
- The bootstrapped set size n_b.
- The voting scheme V.

In order to minimize variability between classifiers and try to ensure a fair experimental procedure, all pool generation methods were evaluated using the same parameters. The chosen parameters were:

- $m = 50$.
- C is the Perceptron.

- $n_b = |T|$.
- V is the Majority Vote Rule.

Where T is the training set, and $|T|$ is its cardinality.

We chose the Perceptron as the base classifier since Breiman pointed out [**?**] that Bagging usually achieves better results when weak learners are used.

The Majority Vote rule was chosen due to its widespread usage in the ensemble literature.

There is one extra parameter for the Random Subspace algorithm, which is the maximum size of the reduced feature set. In our work, we set the maximum size to be 50% of the original feature space dimension.

4.1.2 Data sets

The following public data sets were used in our experiments. All data sets were obtained from the UC Irvine Machine Learning Repository [29], except for the Glass and Satimage data sets, obtained from the KEEL-data set repository [30], and the make_moons data set, which is a synthetic data set available in the scikit-learn Python package [31].

Table 4.1.1 – The data sets used in the experiments

Data set	**# of examples**	**Dimensions**	**# of classes**
Pima	768	8	2
WDBC	569	30	2
CTG	2126	21	3
Ionosphere	351	34	2
Liver	345	6	2
Satimage	6435	36	7
Yeast	1484	8	10
Glass	214	9	6
Vowel	528	9	11
Haberman's Survival	306	3	2
Vertebral Column	310	6	2
Blood Transfusion	747	4	2
E. Coli	336	7	8
Indian Liver Patient Database (ILPD)	579	10	2
make_moons	1000	2	2

Since both the kNN algorithm and the Perceptron algorithm can be affected by the presence of features which have very different scales, we perform feature-wise scaling on all data sets, in order to have all features lie on the $[0, 1]$ interval. This is done by subtracting each feature of each data point from the minimum value of the feature, and dividing by the range in which the feature lies.

4.1.3 Assessing the effect of noise

One of the main motivations of this work is to investigate the effects of noise on the accuracy of pools of classifiers. The method we have proposed is heavily focused on dealing with the effects of noisy instances in the training data. Therefore, it is paramount that we conduct our experiments in a manner that allows us to systematically evaluate the effects of noise on the data.

In order to control the noise on the data sets and to be able to observe the behavior of the classifiers and pools under different noise conditions, we adopted the following procedure for adding noise to the data sets: For each instance $x \in T$, where T is the training set, its class label has a probability pr_{change} of being changed to one of the other classes present in the data set. Another way to put this is that the expected fraction of elements that will have its label changed is pr_{change}.

In this work, we evaluated all algorithms for values of pr_{change} in the set $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ The case $pr_{change} = 0$ is the noise-free scenario.

4.1.4 Methods for statistical analysis

In order to analyze the statistical significance of our results, we follow the recommendations of [32] for evaluating multiple classifiers over several data sets. We use the Friedman test [33], which ranks classifiers by accuracy, the classifier with the highest accuracy having the highest rank. Our null hypothesis is that all classifiers are equivalent, or, more precisely, that their mean ranks over the data sets are not different. We chose a p-value of 0.05, therefore rejecting the null hypothesis if $p < 0.05$.

To evaluate whether there is a significant difference between a pair of classifiers, we perform the Nemenyi post-hoc test [34]. The test calculates a value called the critical difference, denoted here by CD, where for a comparison between k algorithms over N, CD is defined as:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{(N)}}$$

where q_α is the critical value for significance level α. If the difference between the ranks of two algorithms is greater than the value of CD, then they are said to be different with a significance level of α.

We present, for each noise scenario, the result of the Friedman test, and the Critical Differences Diagram, proposed in [32]. The critical difference diagrams make it easy to visually determine which classifiers are significantly different.

4.2 Results

In this section, we present the results of our experiments. The analyses are grouped by noise scenario, as this makes it easier to understand how the proposed algorithm behaves under different circumstances. Following each set of results, we present our comments on them. We first present the comparison of the accuracy achieved by the pool generation algorithms under different noise scenarios. We then present an analysis of the distribution of Instance Hardness values for several data sets, contrasting the values observed in the noisy and noise-free examples.

4.2.1 Accuracy

In order to obtain a baseline of results, our first evaluation is concerned with the performance of the classifiers under a noise-free scenario.

Table 4.2.1 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 0%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	68.60 ± 2.49	65.34 ± 3.18	76.93 ± 0.70	**76.94 ± 0.49**	76.68 ± 0.64
wdbc	93.47 ± 1.85	94.14 ± 1.32	**97.29 ± 0.36**	97.12 ± 0.25	97.08 ± 0.40
ctg	85.15 ± 1.37	85.71 ± 1.32	**88.59 ± 0.28**	88.10 ± 0.23	88.05 ± 0.41
ionosphere	74.90 ± 8.83	76.38 ± 7.09	**86.61 ± 1.24**	84.62 ± 0.81	86.01 ± 1.34
liver	59.80 ± 2.91	57.68 ± 1.02	65.83 ± 1.60	**66.90 ± 0.81**	66.26 ± 1.84
satimage	75.76 ± 2.96	75.90 ± 2.66	82.84 ± 0.16	**82.90 ± 0.17**	82.82 ± 0.16
yeast	41.47 ± 2.95	32.11 ± 4.30	56.62 ± 0.85	**57.57 ± 0.42**	56.92 ± 0.76
glass	47.19 ± 4.76	46.24 ± 3.14	57.44 ± 2.34	**58.56 ± 1.02**	57.61 ± 1.79
vowel	32.49 ± 1.98	26.54 ± 2.63	**51.82 ± 1.17**	48.01 ± 0.90	48.98 ± 1.44
haberman	70.91 ± 3.79	69.35 ± 4.71	**74.68 ± 0.69**	74.05 ± 0.94	73.92 ± 0.62
vertebral	76.45 ± 2.08	71.03 ± 1.77	**81.94 ± 1.15**	80.68 ± 0.78	80.87 ± 1.17
transfusion	67.57 ± 4.04	68.13 ± 5.90	78.55 ± 0.55	78.42 ± 0.42	**78.57 ± 0.30**
ecoli	63.80 ± 2.51	61.50 ± 4.55	76.97 ± 1.32	77.22 ± 0.54	**77.54 ± 1.00**
ilpd	61.66 ± 5.80	63.18 ± 6.74	**71.22 ± 0.69**	70.81 ± 0.72	71.16 ± 0.50
make moons	85.36 ± 1.00	75.33 ± 3.04	88.39 ± 0.56	**88.65 ± 0.30**	88.32 ± 0.45
Mean	66.97	64.57	**75.71**	75.37	75.39

Figure 4.2.1 shows the critical differences diagram:

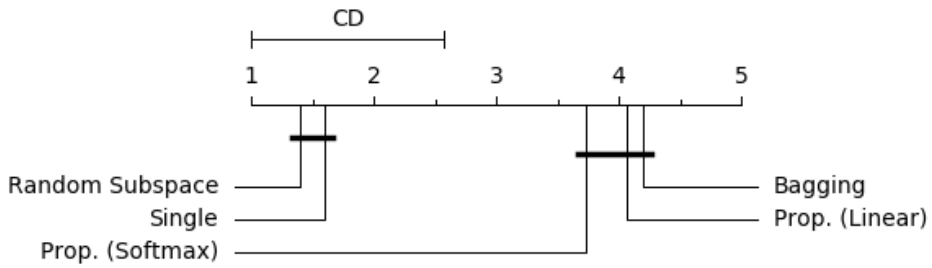

Figure 4.2.1 – Critical Differences Diagram for the noise-free scenario

For this scenario, the Friedman test rejected the null hypothesis with a p-value of 2.6933×10^{-9}.

When there is no noise added to the training instances, the proposed method does not perform any better than the original Bagging algorithm. Nevertheless, it does not fare any worse, either. These results fall in line with our previous expectations. The proposed method was specifically designed with noisy data sets in mind, or data sets with a large fraction of outliers. As such, it is expected that it will show its best results under these conditions, conditions which the first experiment does not satisfy.

However, this result also shows that the proposed method is not outperformed by the original Bagging algorithm. This suggests that the the classifiers in the pools generated by our algorithms are able to learn at least as well as the ones in the pools generated by the original Bagging algorithm. Given that the crucial difference in the pool generation procedure between the two sets lies in how the bootstrapped sets are chosen, this in turn indicates that the bootstrapped sets created by our algorithm as "good" as the ones generated by the Bagging algorithm. This is an indirect but important piece of evidence that points towards the correctness of one of our initial assumptions: that the kDN hardness measure is effective in distinguishing noisy data from noise-free data. This is because, should the kDN measure consider noise-free data points as hard, we would have bootstrapped sets were these data points would be underrepresented, something that would not happen under the original Bagging algorithm. In that case, we would expect to see the proposed method fall short of the accuracy of the original Bagging algorithm.

Furthermore, in this noise-free context, we can also note that both Bagging and our proposed method offer significant performance gains over a single instance of the base classifier. In contrast, the Random Subspace algorithm shows a lower value for the average accuracy than the single classifier in all but four of the data sets, and also larger values for the standard deviation, which in turn indicates high variability of the performance. This may caused by too drastic a reduction of the feature set, considering our data sets have relatively low-dimensional feature sets.

Having established a baseline of performance, we now analyze the noisy scenarios.

Table 4.2.2 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 10%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	64.67 ± 5.12	57.90 ± 5.54	75.76 ± 0.52	**76.60±0.56**	76.15 ± 1.09
wdbc	80.41 ± 9.25	83.99 ± 10.21	95.55 ± 0.55	**96.94±0.35**	96.45 ± 0.51
ctg	76.06 ± 7.52	72.01 ± 9.85	87.55 ± 0.48	**87.76±0.38**	87.70 ± 0.35
ionosphere	69.42 ± 10.00	67.51 ± 9.96	**85.44±1.70**	84.13 ± 1.07	84.73 ± 1.30
liver	60.70 ± 0.99	58.32 ± 0.68	63.07 ± 2.36	63.57 ± 2.11	**64.49±2.22**
satimage	67.64 ± 6.26	66.44 ± 8.01	81.04 ± 0.34	**82.53±0.23**	81.79 ± 0.33
yeast	34.96 ± 5.36	27.58 ± 5.61	55.07 ± 0.71	**57.16±0.58**	56.47 ± 0.68
glass	41.74 ± 6.44	39.82 ± 6.19	54.95 ± 1.79	56.35 ± 1.58	**57.68±3.28**
vowel	28.59 ± 2.68	22.45 ± 3.07	**48.12±1.44**	46.57 ± 1.42	46.41 ± 1.51
haberman	69.08 ± 5.74	69.78 ± 5.16	**73.99±0.84**	73.86 ± 0.53	73.40 ± 0.96
vertebral	71.71 ± 3.19	66.97 ± 3.43	80.39 ± 2.20	80.48 ± 1.26	**81.00±1.44**
transfusion	63.25 ± 9.30	60.26 ± 9.99	77.94 ± 0.78	**78.27±0.35**	77.98 ± 0.34
ecoli	56.19 ± 8.07	52.96 ± 7.43	76.28 ± 1.55	76.43 ± 0.69	**76.81±1.27**
ilpd	59.78 ± 5.89	59.36 ± 6.29	70.05 ± 1.31	70.29 ± 1.24	**70.52±1.35**
make moons	75.78 ± 6.50	67.82 ± 4.26	87.54 ± 0.64	88.28 ± 0.60	**88.38±0.56**
Mean	61.33	58.21	74.18	74.61	**74.66**

Table 4.2.3 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 20%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	59.39 ± 5.30	54.04 ± 4.95	74.21 ± 2.29	**75.38±1.57**	74.70 ± 0.98
wdbc	74.40 ± 11.68	76.93 ± 11.92	92.72 ± 1.24	**95.66±0.74**	95.20 ± 0.73
ctg	65.24 ± 9.62	60.05 ± 18.09	86.21 ± 0.45	**87.65±0.27**	86.75 ± 0.42

Continued on next page

ionosphere	60.11 ± 10.28	58.11 ± 6.93	82.45 ± 1.65	82.57 ± 1.63	**82.97±2.37**
liver	58.55 ± 4.40	56.29 ± 4.71	**62.75±2.39**	62.29 ± 2.83	62.64 ± 2.93
satimage	53.69 ± 5.57	56.19 ± 5.92	79.61 ± 0.51	**82.29±0.13**	80.75 ± 0.37
yeast	31.54 ± 4.07	24.06 ± 4.75	54.75 ± 1.11	**56.85±0.40**	55.42 ± 0.71
glass	38.69 ± 7.78	35.60 ± 7.00	53.10 ± 2.34	**55.57±1.91**	54.96 ± 2.02
vowel	24.51 ± 3.01	19.11 ± 2.76	43.83 ± 0.88	**46.07±1.22**	45.25 ± 2.06
haberman	66.64 ± 3.09	68.74 ± 4.58	73.44 ± 1.12	**73.89±0.43**	73.69 ± 0.49
vertebral	67.68 ± 10.57	64.71 ± 10.19	77.48 ± 2.69	78.32 ± 1.76	**79.00±1.70**
transfusion	58.75 ± 9.22	58.46 ± 9.68	77.09 ± 0.90	**77.58±1.36**	77.20 ± 0.72
ecoli	51.18 ± 7.82	46.45 ± 11.18	74.90 ± 0.82	**76.22±1.08**	75.75 ± 1.49
ilpd	57.26 ± 7.30	57.67 ± 9.31	69.86 ± 1.54	**70.10±1.21**	69.74 ± 1.75
make moons	67.19 ± 9.35	62.67 ± 7.40	86.20 ± 1.07	**88.09±0.54**	87.26 ± 0.96
Mean	55.65	53.27	72.57	**73.90**	73.42

Table 4.2.4 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 30%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	55.93 ± 6.66	50.22 ± 6.65	71.45 ± 2.04	**74.49±1.44**	73.84 ± 1.92
wdbc	66.36 ± 9.93	69.92 ± 11.85	88.05 ± 2.90	**92.70±1.58**	91.46 ± 2.07
ctg	52.91 ± 14.24	48.25 ± 16.96	85.48 ± 0.42	**86.64±0.59**	85.98 ± 0.50
ionosphere	56.82 ± 6.42	54.24 ± 6.64	77.69 ± 4.98	**80.15±2.41**	80.06 ± 2.24
liver	56.70 ± 3.67	56.64 ± 4.59	58.14 ± 2.45	**59.68±3.00**	58.84 ± 3.48
satimage	46.87 ± 5.17	48.07 ± 6.28	77.76 ± 0.62	**81.59±0.34**	79.42 ± 0.32
yeast	26.81 ± 7.21	19.91 ± 5.93	53.13 ± 1.23	**56.57±0.65**	54.68 ± 0.87
glass	31.55 ± 8.40	26.65 ± 6.26	52.13 ± 2.81	**54.90±2.33**	54.02 ± 3.48
vowel	22.87 ± 2.30	17.33 ± 2.47	40.55 ± 1.43	**44.77±1.85**	42.48 ± 1.16
haberman	57.38 ± 12.53	56.85 ± 16.95	71.02 ± 2.17	**72.52±1.54**	71.74 ± 2.21
vertebral	66.13 ± 10.48	64.58 ± 10.48	73.87 ± 3.77	**75.65±1.81**	74.35 ± 3.10
transfusion	57.46 ± 10.10	56.63 ± 9.54	75.11 ± 3.27	75.55 ± 2.96	**76.14±3.56**
ecoli	47.28 ± 9.61	41.89 ± 13.08	73.81 ± 1.63	**75.62±1.14**	74.85 ± 1.20
ilpd	55.89 ± 8.11	54.77 ± 10.06	65.50 ± 3.72	**67.84±1.65**	66.75 ± 3.00

Continued on next page

make	58.96 ± 6.78	59.18 ± 4.69	84.85 ± 1.94	**86.55±0.73**	85.87 ± 0.62
moons					
Mean	50.66	48.34	69.900	**72.35**	71.37

Table 4.2.5 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 40%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	51.10 ± 7.83	46.03 ± 5.74	66.28 ± 4.09	**69.38±1.90**	66.86 ± 4.06
wdbc	58.15 ± 10.69	56.32 ± 10.46	75.81 ± 4.14	**81.62±4.55**	79.12 ± 5.21
ctg	42.46 ± 17.70	36.43 ± 17.00	82.47 ± 0.85	**85.14±0.65**	83.85 ± 0.73
ionosphere	53.43 ± 4.94	51.78 ± 5.62	66.25 ± 5.98	**73.05±4.26**	71.03 ± 4.05
liver	**55.91±4.56**	55.04 ± 6.06	55.83 ± 4.11	55.65 ± 3.99	55.65 ± 1.88
satimage	39.63 ± 7.65	39.36 ± 8.14	75.55 ± 1.37	**80.65±0.34**	77.84 ± 0.61
yeast	25.38 ± 7.24	17.81 ± 6.50	50.15 ± 1.68	**55.64±0.61**	53.75 ± 0.87
glass	24.85 ± 7.18	21.64 ± 6.46	48.93 ± 3.25	**54.00±2.76**	53.19 ± 3.37
vowel	19.83 ± 2.57	15.16 ± 1.93	38.01 ± 1.22	**42.89±1.52**	39.37 ± 1.68
haberman	51.30 ± 15.74	47.14 ± 17.49	61.60 ± 9.16	**66.28±6.60**	64.61 ± 5.57
vertebral	62.16 ± 11.73	61.19 ± 13.21	64.03 ± 3.95	66.48 ± 4.48	**66.68±2.98**
transfusion	53.05 ± 12.23	54.34 ± 10.63	64.61 ± 7.85	**73.00±4.80**	69.38 ± 4.69
ecoli	41.93 ± 9.49	33.88 ± 10.35	71.55 ± 2.28	**73.63±1.59**	72.52 ± 1.44
ilpd	55.30 ± 8.69	53.69 ± 10.91	59.38 ± 2.76	**63.78±4.43**	63.48 ± 3.05
make	56.85 ± 9.27	55.85 ± 6.67	78.36 ± 2.04	**84.02±1.36**	81.70 ± 2.46
moons					
Mean	46.09	43.04	63.92	**68.35**	66.60

The Friedman test rejected the null hypothesis in all four noise levels, with p-values of 3.8499×10^{-10}, 9.6280×10^{-11}, 1.0532×10^{-11}, and 1.7848×10^{-11}.

When analyzing the results for noisy scenarios shown in tables 4.2.2 to 4.2.5 , the observed behavior changes, and the proposed method tends to have the highest mean accuracy, mostly in its linear variant. In most cases, when the linear variant of the proposed method is not the method with the highest accuracy, it takes second place, behind the Softmax variant of the proposed method.

Figures 4.2.2 to 4.2.5 show the critical differences diagram for the four noise levels:

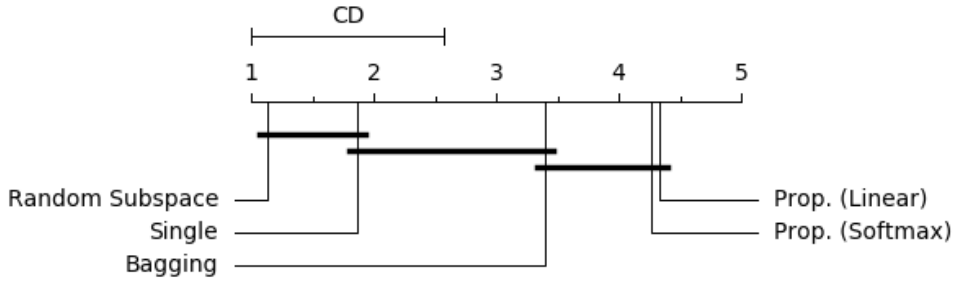

Figure 4.2.2 – Critical Differences Diagram for the 10% noise level

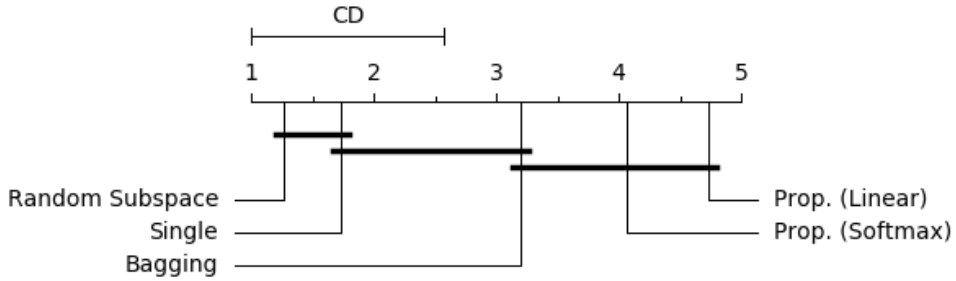

Figure 4.2.3 – Critical Differences Diagram for the 20% noise level

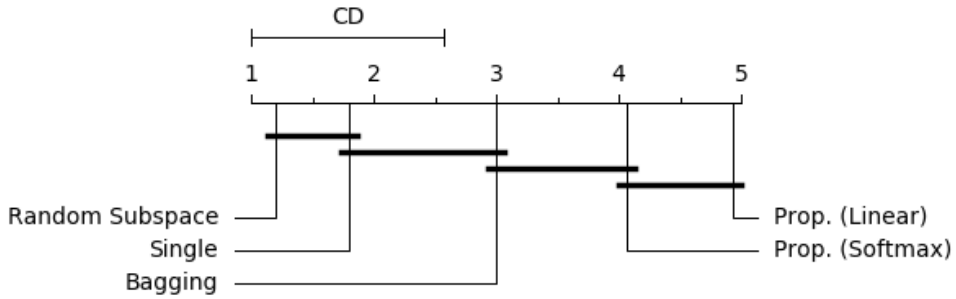

Figure 4.2.4 – Critical Differences Diagram for the 30% noise level

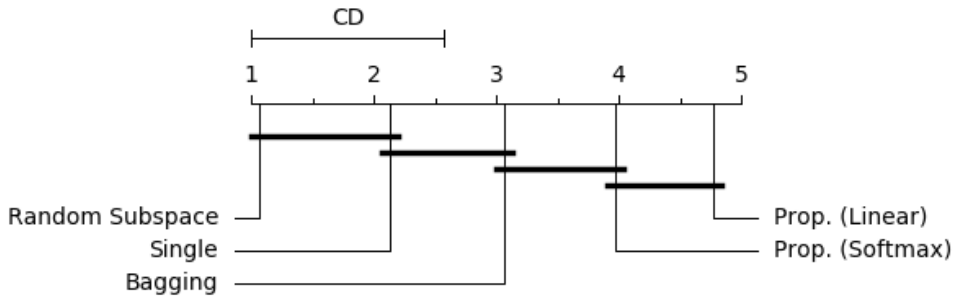

Figure 4.2.5 – Critical Differences Diagram for the 40% noise level

Figure 4.2.2 shows that at noise levels of 10% and 20%, there is no significant difference between the proposed method and the original Bagging algorithm. This again

shows that at lower noise levels, the proposed method does not offer an advantage over Bagging, but does not perform any worse, either.

However, for the noise levels of 30%, and 40%, we can observe there is a significant difference between the proposed method in its linear variant and the original Bagging algorithm. This indicates that our method is particularly suited to dealing with higher noise level. However, we can observe that there is not a significant difference between the proposed method in its Softmax variant and the Bagging algorithm.

Even though the results show that there is no significant difference between the two variants of our method, we still attempt to tease out the effects that lead to one variant having a higher accuracy in a data set. When analyzing the scenarios in which the Softmax variant has a higher mean accuracy than the linear variant, three data sets stand out. In three different noise levels the Softmax variant has higher mean accuracy in the Liver, Vertebral Column and Indian Liver Patient Database data sets, most notably in the 20% and 30% noise levels, where this higher accuracy is observed for all three data sets. These data sets share in common the fact that they are all binary classification problems, have a relatively small number of examples and low dimensionality of the feature space. This, however, is not conclusive evidence, since the Pima and Haberman's Survival data set also exhibit similar characteristics, but do not share the same behavior. Also, it must be noted that the differences are small.

One can also observe the accuracy of all the methods decreasing, as expected. Nevertheless, if we observe the original Bagging algorithm and our proposed method in its two variants, we see a gradual fall in the mean accuracy, where its value drops by no more than two percentage points which increase in the noise level. That is, until we move from noise level of 30% to 40%, when we see a drop of about 4 percentage points. This already points to a change in regime which will be more clear in our next analysis.

We now turn our attention to the last noise level analyzed.

Table 4.2.6 – Mean and standard deviation of the mean accuracy over the folds for each classifier, at a noise level of 50%. The best values (highest mean) are shown in bold.

Data Set	**Perceptron**	**Random Subspace**	**Bagging**	**Proposed (Linear)**	**Proposed (Softmax)**
pima	46.27 ± 2.51	42.85 ± 4.69	$\mathbf{54.74 \pm 6.05}$	51.49 ± 8.24	53.90 ± 7.54
wdbc	49.00 ± 7.53	48.04 ± 12.06	$\mathbf{51.71 \pm 5.71}$	50.41 ± 8.22	51.28 ± 6.78

Continued on next page

ctg	43.92 ± 16.03	30.87 ± 21.20	76.21 ± 3.66	**82.11±1.22**	81.04 ± 1.80
ionosphere	46.84 ± 7.21	**50.60±6.08**	47.26 ± 9.02	45.74 ± 8.52	44.25 ± 7.53
liver	52.00 ± 6.22	**53.22±7.22**	50.09 ± 3.79	49.25 ± 2.70	49.28 ± 3.81
satimage	34.01 ± 8.47	31.35 ± 8.34	72.91 ± 2.24	**79.40±0.35**	75.69 ± 0.55
yeast	19.65 ± 7.20	15.28 ± 6.71	47.17 ± 1.25	**54.25±0.96**	49.89 ± 1.60
glass	23.47 ± 5.99	19.15 ± 6.15	44.59 ± 3.59	**47.81±4.68**	46.58 ± 3.89
vowel	16.33 ± 2.20	12.40 ± 2.20	33.16 ± 2.36	**39.67±1.33**	35.97 ± 1.04
haberman	40.24 ± 7.71	39.99 ± 8.43	**51.70±6.61**	51.59 ± 8.37	51.34 ± 11.05
vertebral	55.71 ± 11.73	**58.65 ± 14.15**	47.29 ± 6.56	46.39 ± 8.32	46.48 ± 9.14
transfusion	50.97 ± 10.65	**51.86±7.34**	51.52 ± 9.73	48.71 ± 9.56	51.77 ± 7.93
ecoli	32.39 ± 10.12	26.29 ± 11.98	63.83 ± 3.23	**70.51±2.76**	68.41 ± 3.13
ilpd	**56.16±7.54**	53.35 ± 11.36	50.75 ± 3.45	50.00 ± 4.06	50.83 ± 6.77
make	47.26 ± 9.53	46.67 ± 5.21	**50.69 ± 11.32**	45.14 ± 13.54	49.34 ± 10.31
moons					
Mean	40.95	38.70	52.91	**54.16**	53.74

First of all, it should be noted that, at this noise level, there's basically no useful information left in the data. We nevertheless conducted these experiments to evaluate how the different methods would behave under such extreme conditions.

This was the only scenario in which the Friedman test did not reject the null hypothesis, with a p-value of 0.1712. This means that there is no significant difference between the classifiers, something that is further confirmed by observing the critical differences diagram.

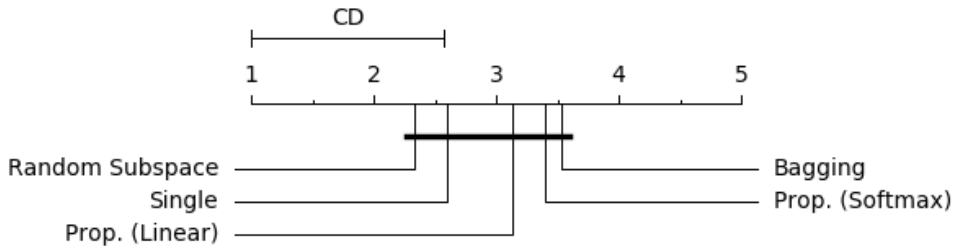

Figure 4.2.6 – Critical Differences Diagram for the 50% noise level

The second most important observation is that the performance of all methods has degraded to an extreme. In some binary classification data sets, for several of the proposed methods, the accuracy is worse than that of random guessing. In these cases, the accuracy indicates that the classifiers have failed to learn anything at all.

Turning our attention back to the analysis of the mean accuracy over the data sets, we see a drop of more than 10 percentage points for the original Bagging algorithm and our method, when compared to the mean accuracy at a noise level of 40%. This indicates that a change in regime has occurred, to one where the noise is so high that the classifiers fail to learn.

Furthermore, one can observe that the Random Subspace algorithm has the highest accuracy in five of the data sets, something which had not occurred in any occasion so far.

Even if we did not expect our method to perform well under these conditions, further analysis offered valuable insight into one of the mechanisms that cause this drop in performance. For this analysis, we defer the reader to the next subsection, where we evaluate the change in the distribution of the values of the kDN measure as the noise level increases.

4.2.2 The behavior of the kDN measure at different noise levels

Though the analysis of the accuracy of the different methods at varying noise levels already allows us to gain some level of insight, it does not show us the whole picture. In order to obtain a better understanding into the problem of working with noisy data, and also better understand how our own method behaves under different circumstances, we measured the values of the kDN instance hardness for varying levels of noise.

We used the same methodology as adopted in the experiments involving the accuracy of the classifiers, performing 10 repetitions of the 5-fold cross-validation. In fact, we applied the same seeds to the random number generator as used in the first experiment, which means the data are exactly the same for both experiments.

We recorded the kDN measure value for every data point in the training set, for each partition and each repetition, for all the data sets. We then calculated the average and the standard deviation of the values for each noise level in the set $\{10\%, 20\%, 30\%, 40\%, 50\%\}$. In order to keep this exposition brief, we only present a subset of the results here.

The results can be better understood by first noting whether the problem is one of binary classification or of multiclass classification. To illustrate this, let us first look at the results for two binary classification data sets.

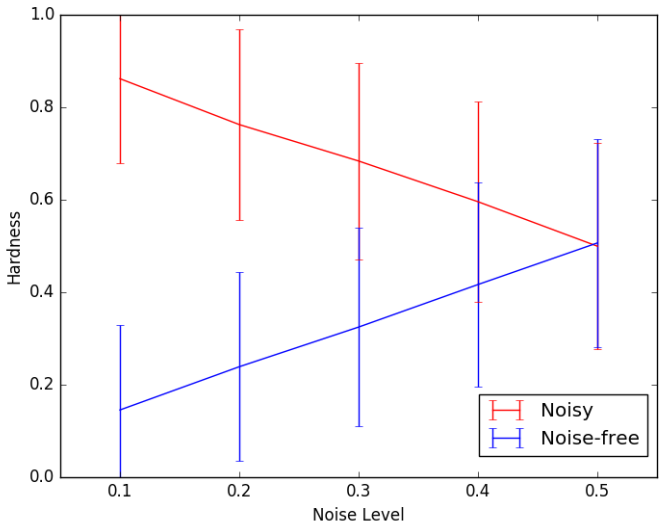

Figure 4.2.7 – Instance Hardness values - WDBC Data set

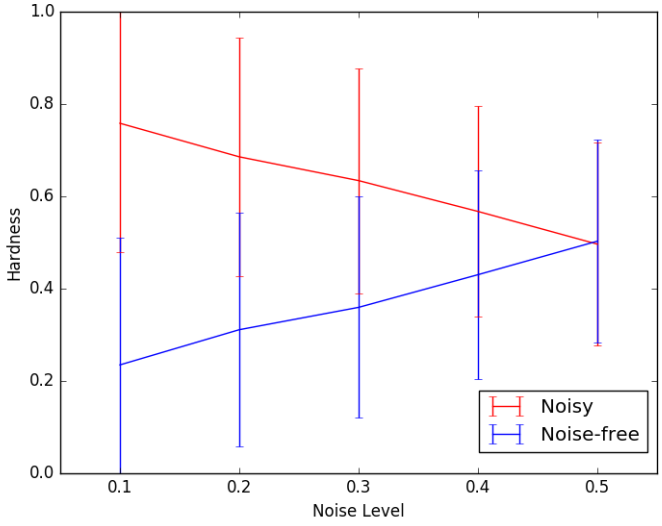

Figure 4.2.8 – Instance Hardness values - Ionosphere Data set

Figures 4.2.7 and 4.2.8 show the distribution of noisy and non-noisy instances at different levels of noise. At a noise level of 10%, it is expected that approximately 10% of instances will have had noise added to them. The distribution of the instance hardness of the noisy instances is calculated on these 10% of the instances. Likewise, it is expected that 90% of the instances will not have had noise added to them, and the distribution of the instance of the non-noisy instances will be calculated on these 90% of instances.

The first thing one can observe when analyzing Figure 4.2.7 and Figure 4.2.8 is that the hardness of noisy and non-noisy instances seems to truly be different, in

particular at lower noise levels. However, as the noise level increases, the distinction gradually disappears.

The first mechanism which causes the distinction to vanish is the increase in the hardness of noise-free instances. This can be understood by noting that, as the noise level increases, the more likely it is that even instances in the center of classes will end up with neighbors who had their class label changed to the opposite class. Therefore, the number of neighbors with a disagreeing class label increases, increasing the value of the instance's hardness.

The second, more counterintuitive mechanism is the decrease in the hardness of noisy instances. This can be explained by observing that, at lower noise levels, a noisy instance is likely to be surrounded by non-noisy instances of the opposite class. This means that it will have a high hardness value. However, as the noise level increases, it is possible that some of its neighbors will also have their labels changed, therefore decreasing the number of disagreeing neighbors, and the instance's hardness.

We now focus on multiclass problems, where we see a different kind of behavior.

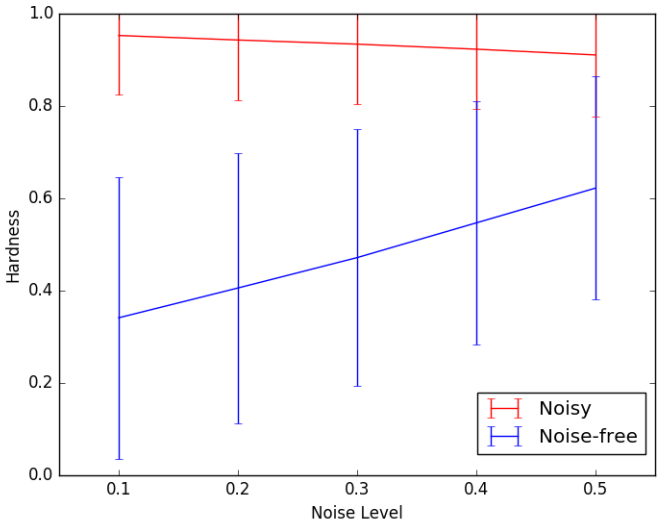

Figure 4.2.9 – Instance Hardness values - E. Coli Data set

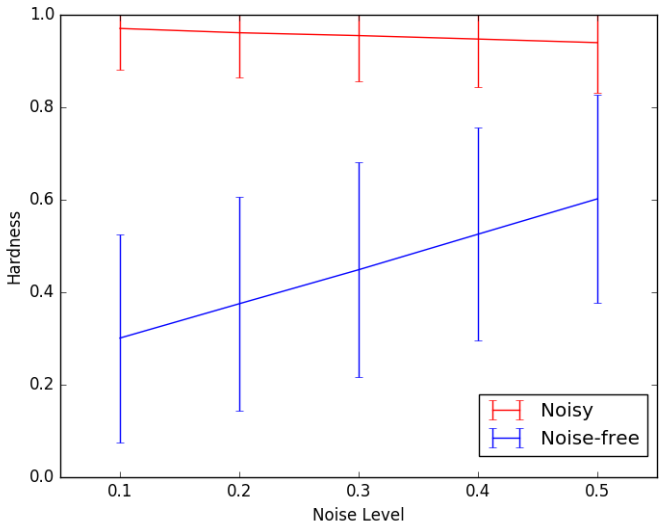

Figure 4.2.10 – Instance Hardness values - Vowel Data set

Here, we also observe a gradual blurring of the distinction between noisy and noise-free instances. However, we do not observe as pronounced a drop in the hardness of the noisy instances.

Let us analyze what happens when a neighbor $x^{'}$ of a noisy instance x has its class label changed. In a multiclass problem, the label of $x^{'}$ can change to any of the other class labels in the problem. Assuming there are c possible labels, there is only a $\frac{1}{c-1}$ probability that the class label of $x^{'}$ will be changed to the same class label as x. There is, however, a probability of $\frac{c-2}{c-1}$ that the label will be changed to one that is different from that of x. In the second case, the hardness of x will not change at all. This is in stark contrast to binary problems, where a change in label can only result in the decrease the hardness of x, as explained previously.

4.2.3 The distribution of the hardness in noisy vs. non-noisy instances

As explained in the previous subsection, we recorded the values of the instance hardness for all training examples in our experiment. While our previous analysis is sufficient to show the effects of the noise level on the values of the kDN measure, it does not show the complete picture. Since we only displayed the mean and standard deviation of the hardness, we may not have enough information to accurately compare the distribution of hardness values for noisy and non-noisy instances. Therefore, in this subsection we present box plots that show the contrast between the hardness values of noisy and non-noisy instances.

Again, for the sake of brevity, we limit ourselves in the number of figures included in this analysis. There are potentially 75 figures which could be presented, as there are

fifteen data sets, and five different noise levels (excluding the noise-free context). We include here only the box plots at a noise level of 10%, for the same data sets analyzed in the previous subsection. The results are shown in Figures 4.2.11 to 4.2.14.

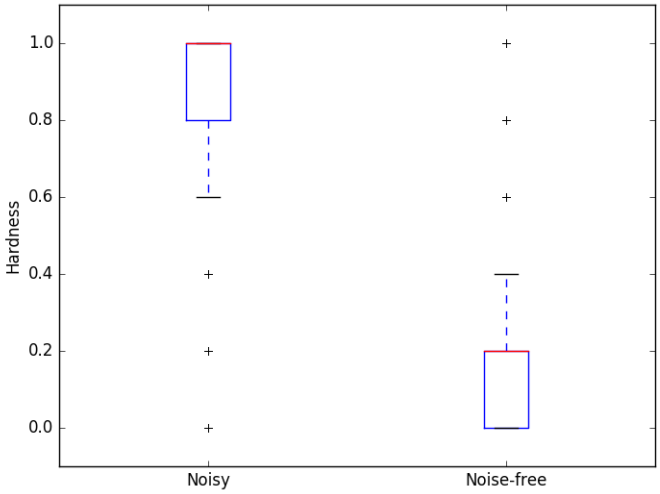

Figure 4.2.11 – Distribution of Instance Hardness values - WDBC Data set

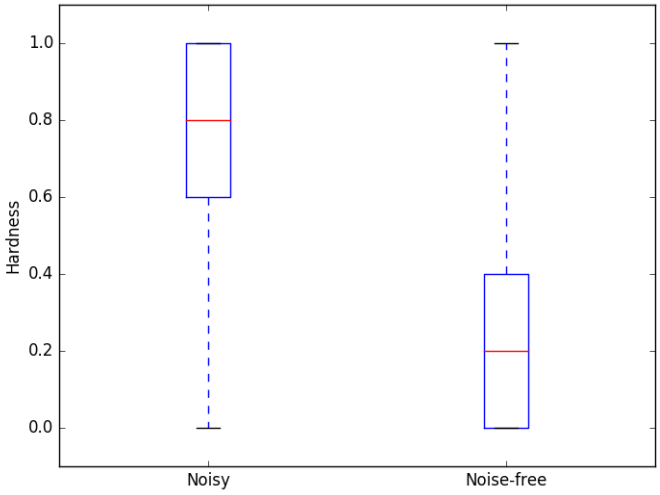

Figure 4.2.12 – Distribution of Instance Hardness values - Ionosphere Data set

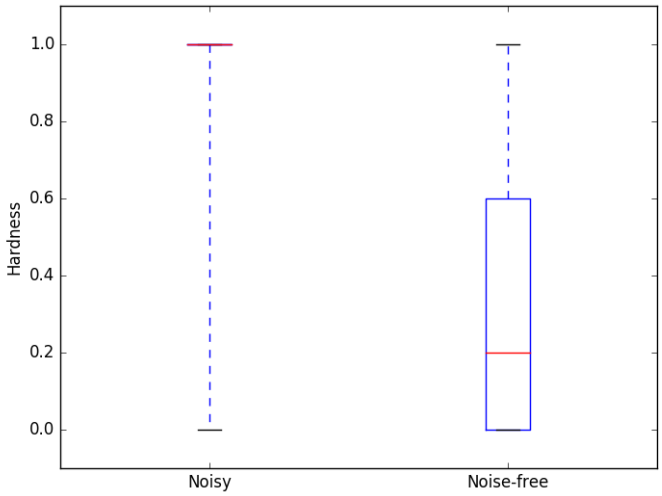

Figure 4.2.13 – Distribution of Instance Hardness values - E. Coli Data set

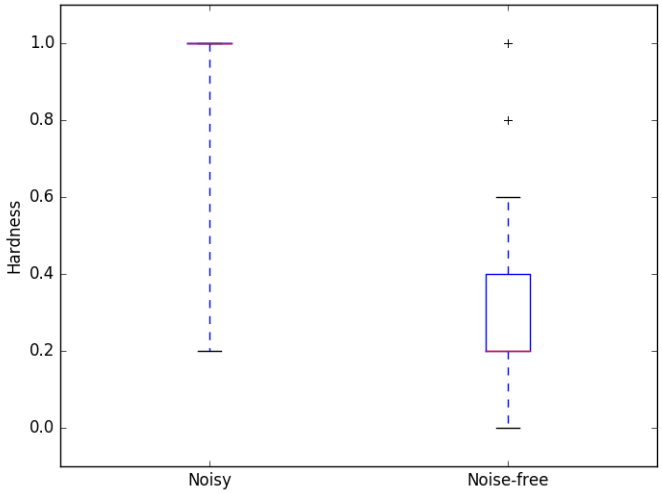

Figure 4.2.14 – Distribution of Instance Hardness values - Vowel Data set

The first noteworthy phenomenon is that, since the values of the kDN measure are discrete, the values of the median and the quartiles may coincide. The same is possible for the values of the quartiles and the upper or lower adjacent. It is even possible that one adjacent, one quartile and the median will have the same value. This is clearly visible on Figure 4.2.13 and Figure 4.2.14.

Looking at the box plots, we can see a much more nuanced picture than in our previous analysis. The medians of the distributions for noisy and noise-free instances tend to be far apart from one another. Furthermore, the quartiles show us that most instances of both subsets lie in specific ranges of hardness values, and these ranges tend

to not overlap for noisy and non-noisy instances. This is yet another piece of evidence that indicates that the kDN measure is highly effective in discriminating noisy instances from non-noisy instances.

5 Conclusions and Future Work

In this work, we proposed a new method which combines data complexity measures and ensemble methods to achieve better performance on scenarios which involve noisy data. More specifically, our method leveraged the k-Disagreeing Neighbors measure of instance hardness to modify the bootstrapping process of the Bagging algorithm.

The proposed modification to Bagging aimed to avoid adding noisy examples to the training datasets as much as possible, since several works in the classification literature pointed to the adverse effects of noise. Nevertheless, we adopted a filtering approach that was probabilistic in nature, in order to preserve examples that are inherently hard, such as examples on the border of classes. We proposed two different methods of calculating instance selection probabilities during the construction of the bootstrapped training sets. Both approaches were based on a measure of instance hardness, with one being linear in nature and the other making use of the Softmax function.

We performed experiments on fifteen publicly available datasets, comparing our method with the Random Subspace and Bagging algorithms, widely used in the literature. We also compared the ensemble algorithms with a single classifier, in order to obtain a baseline of performance.

Our experiments show that for noise-free scenarios and for noise levels of 10% and 20%, the performance of our algorithm was similar to that of the of the original Bagging algorithm. However, at noise levels of 30% and 40%, our algorithm is the best performing method, and there is a significant difference between the linear variant of our method and the Bagging algorithm, the second best performing method. This results suggests that our algorithm is better suited than Bagging to dealing with high label noise levels.

We were also able to observe that at a noise level of 50%, there is in effect no useful information left for the learning process. The performance of all algorithms was heavily degraded, and there was no significant difference between the algorithms.

An analysis of the distribution of the hardness of the instances was also performed. We were able to observe that it becomes progressively harder to distinguish noisy from non-noisy instances, as the noise level increases. Nevertheless, we still observed a difference in instance hardness values for noisy and non-noisy instances, which supports our initial hypothesis that instance hardness measures are useful in estimating which instances might be noise.

5.1 Future Work

There are quite a few possibilities which can be investigated based on the work here presented. We present here the ones we consider most immediately applicable.

The first possible line of inquiry is investigating the use of instance hardness measures other than the k-Disagreeing Neighbors measure. This opens up possibilities to prioritize removing different types of noise. More broadly, it would be interesting to investigate the use of instance hardness measures not based on class labels. This is because these measures are better suited to classification problems, precluding the development of techniques focused on regression, or even approaches focused on generative methods.

Another possibility for investigation is combining ensemble methods other than Bagging with data complexity measures. In particular, Boosting techniques seem particularly like to benefit from a proper treatment of noisy data, given our discussion of the effect of noisy on algorithms such as AdaBoost.

Finally, while our work was focused on static combination methods, one could also attempt to combine data complexity measures with dynamic selections schemes. This could be achieved by using the data complexity information to choose the most appropriate subset of classifiers for hard data, or by weighing the data points in such a manner that noisy data would have less of an effect in the dynamic choice of the classifiers.

Bibliography

1 ZHOU, Z.-H. *Ensemble methods: foundations and algorithms*. [S.l.]: CRC press, 2012. Referenced on page 21.

2 BREIMAN, L. Bagging predictors. *Machine Learning*, Springer, v. 24, n. 2, p. 123–140, 1996. Referenced 3 times, on pages 21, 27, and 31.

3 KHOSHGOFTAAR, T. M.; HULSE, J. V.; NAPOLITANO, A. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, IEEE, v. 41, n. 3, p. 552–568, 2011. Referenced on page 21.

4 LONG, P. M.; SERVEDIO, R. A. Random classification noise defeats all convex potential boosters. *Machine Learning*, Springer, v. 78, n. 3, p. 287–304, 2010. Referenced 2 times, on pages 21 and 29.

5 SCHAPIRE, R. E. A brief introduction to boosting. In: *IJCAI International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 1999. v. 2, p. 1401–1406. Referenced 2 times, on pages 21 and 26.

6 WILSON, D. L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE, v. 2, n. 3, p. 408–421, 1972. Referenced on page 22.

7 SMITH, M. R.; MARTINEZ, T.; GIRAUD-CARRIER, C. An instance level analysis of data complexity. *Machine Learning*, Springer, v. 95, n. 2, p. 225–256, 2014. Referenced 4 times, on pages 22, 29, 31, and 34.

8 BRITTO, A. S.; SABOURIN, R.; OLIVEIRA, L. E. S. Dynamic selection of classifiers - A comprehensive review. *Pattern Recognition*, v. 47, n. 11, p. 3665–3680, 2014. Referenced on page 25.

9 WOZNIAK, M.; GRANA, M.; CORCHADO, E. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 2014. Referenced on page 26.

10 UEDA, N.; NAKANO, R. Generalization error of ensemble estimators. *Proceedings of International Conference on Neural Networks (ICNN'96)*, v. 1, n. xi, p. 90–95, 1996. Referenced on page 26.

11 BROWN, G.; WYATT, J.; TINO, P. Managing Diversity in Regression Ensembles. *Journal of Machine Learning Research*, v. 6, p. 1621–1650, 2005. Referenced on page 26.

12 TUMER, K.; GHOSH, J. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 1996. Referenced on page 26.

13 FERREIRA, A. J.; FIGUEIREDO, M. A. T. Boosting Algorithms: A Review of Methods, Theory, and Applications. In: _____. *Ensemble Machine Learning: Methods and Applications*. Boston, MA: Springer US, 2012. p. 35–85. Referenced on page 26.

14 HO, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, n. 8, p. 832–844, 1998. Referenced on page 26.

15 BREIMAN, L. Using iterated bagging to debias regressions. *Machine Learning*, v. 45, n. 3, p. 261–277, 2001. Referenced on page 27.

16 LI, C. Classifying imbalanced data using a bagging ensemble variation (BEV). In: *Proceedings of the 45th annual southeast regional conference on - ACM-SE 45*. [S.l.: s.n.], 2007. p. 203. Referenced on page 27.

17 WANG, S.; YAO, X. Diversity analysis on imbalanced data sets by using ensemble models. *2009 IEEE Symposium on Computational Intelligence and Data Mining*, p. 324–331, 2009. Referenced on page 27.

18 CHAWLA, N. V. et al. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, v. 16, p. 321–357, 2002. Referenced on page 27.

19 OLIVEIRA, D. V. R. et al. A bootstrap-based iterative selection for ensemble generation. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–7. Referenced on page 27.

20 FRÉNAY, B.; VERLEYSEN, M. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2014. Referenced on page 28.

21 OKAMOTO, S.; NOBUHIRO, Y. An Average-Case Analysis of the k-nearest Neighbor Classifier for Noisy Domains. In: *IJCAI International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 1997. v. 1, p. 238–243. Referenced on page 29.

22 BI, Y.; JESKE, D. R. The efficiency of logistic regression compared to normal discriminant analysis under class-conditional classification noise. *Journal of Multivariate Analysis*, v. 101, n. 7, p. 1622–1637, 2010. Referenced on page 29.

23 ZHANG, J.; YANG, Y. Robustness of regularized linear classification methods in text categorization. In: *Proceedings of SIGIR-03, 26th ACM International Conference on Research and Development in Information Retrieval*. [S.l.: s.n.], 2003. p. 190–197. Referenced on page 29.

24 DIETTERICH, T. G. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and ranomization. *Machine Learning*, v. 40, n. 2, p. 139–157, 2000. Referenced on page 29.

25 MCDONALD, R. A.; HAND, D. J.; ECKLEY, I. A. An empirical comparison of three boosting algorithms on real data sets with artificial class noise. In: SPRINGER. *International Workshop on Multiple Classifier Systems*. [S.l.], 2003. p. 35–44. Referenced on page 29.

26 BRIGHTON, H.; MELLISH, C. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, v. 6, n. 2, p. 153–172, 2002. Referenced on page 29.

27 MANSILLA, E. B.; HO, T. K. On classifier domains of competence. *Proceedings - International Conference on Pattern Recognition*, v. 1, p. 136–139, 2004. Referenced on page 29.

28 DUDA, R. *Pattern classification*. New York: Wiley, 2001. 27 p. Referenced on page 33.

29 LICHMAN, M. *UCI Machine Learning Repository*. 2013. Disponível em: <http://archive.ics.uci.edu/ml>. Referenced on page 38.

30 ALCALÁ-FDEZ, J. et al. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, v. 17, 2011. Referenced on page 38.

31 PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Referenced on page 38.

32 DEMŠAR, J. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, v. 7, p. 1–30, 2006. Referenced on page 39.

33 FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, Taylor & Francis, v. 32, n. 200, p. 675–701, 1937. Referenced on page 39.

34 NEMENYI, P. Distribution-free multiple comparisons. In: *Biometrics*. [S.l.: s.n.], 1962. v. 18, n. 2, p. 263. Referenced on page 39.