ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

COMPARISON AND ANALYSIS OF CLASSIFICATION METHODS ON MULTIVARIATE DATASETS WITH PYTHON

Διπλωματική Εργασία

του

Κουβαρά Σωκράτη

Θεσσαλονίκη, Μήνας 2019

COMPARISON AND ANALYSIS OF CLASSIFICATION METHODS ON MULTIVARIATE DATASETS WITH PYTHON

Κουβαράς Σωκράτης

Πτυχίο Μαθηματικών, ΑΠΘ,2014

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Σαμαράς Νικόλαος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ηη/μμ/εεεε

Ονοματεπώνυμο 1	Ονοματεπώνυμο 2	Ονοματεπώνυμο 3
.....	.....	.....

Κουβαράς Σωκράτης
.....

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι η μελέτη των διάφορων αλγορίθμων κατηγοριοποίησης, η κατασκευή ενός προβλεπτικού μοντέλου για κάθε αλγόριθμο με την βοήθεια της γλώσσας προγραμματισμού Python και βιβλιοθηκών μηχανικής μάθησης (Scikit , imblearn , XGBoost) και η σύγκριση της απόδοσης αυτών των μοντέλων.

Η αλγόριθμοι που θα μελετηθούν είναι :

1. Νευρωνικά δίκτυα (Multilayer Perceptron),
2. Κ πλησιέστεροι γείτονες (K-NN)
3. Naive Bayes
4. Boosting (Adaboost και XGBoost),
5. Δέντρα απόφασης (CART)
6. Support Vector Machines
7. Random Forest.

Για την κατασκευή ενός μοντέλου για κάθε αλγορίθμου, θα χρησιμοποιηθούν σύνολα δεδομένων από το UC Irvine Machine Learning Repository.

Λέξεις Κλειδιά:

XGBoost, Support Vector Machine, SVM, Adaboost, Random Forest, K-NN, Decision Trees, boosting, bagging , Neural Network, Multilayer Perceptron, Naïve Bayes

Abstract

The purpose of this research is the use and study of the variety of available Classification methods, building model with the use of Python and Machine Learning packaged (Scikit, Imblearn and XGBoost) and finally the comparison of those algorithms in terms of speed and accuracy, where the metric of accuracy will be determined based on the nature of the data for each dataset

The algorithms that are going to be studied are :

1. Neural Networks (Multilayer Perceptron),
2. K nearest neighbors (K-NN)
3. Naive Bayes
4. Boosting algorithms (Adaboost και XGBoost),
5. Decision Trees (CART)
6. Support Vector Machines
7. Random Forest (Bagging Algorithm)

The datasets that are going to be used where taken from UC Irvine Machine Learning Repository and are available online.

Keywords:

XGBoost, Support Vector Machine, SVM, Adaboost, Random Forest, K-NN, Decision Trees, boosting, bagging , Neural Network, Multilayer Perceptron, Naïve Bayes

Table of Contents

List of Figures

List of Tables

1 Introduction

Machine learning is a field in Computer Science that has been around since 1950 when for the first time the term appeared (Arthur, 1959). It has gained in popularity the last decade with an increasing number of companies using Machine Learning to interpret their data and gain valuable insights.

We can use it for a variety of subjects such as computer vision, email filtering but also for Data Mining, a subset of Machine Learning that discovers patterns in data sets. The process of Data Mining we are interested in this study is Classification, which is closely related to categorization. For Classification tasks, our goal is to predict a certain label of given instance based on the rest of its features.

1.1 Problem definition

There is a great variety of algorithms that can be used for Classification, such as Decision Trees, Support vector machines or Neural Networks. Due to the differences in the way each algorithm works there is not any guideline to choose which algorithm fits better the data we have in order to produce the best model that describes our data.

1.2 Objectives

The purpose of this research is to compare the most common classification algorithms in datasets of various complexity. To determine the effectiveness of each algorithm we are going to use the accuracy of the model that was produced from each one. For the Scania dataset a custom metric was used in order to compare each algorithm as it was requested at the Industrial Challenge 2016 at The 15th International Symposium on Intelligent Data Analysis (IDA). Finally, we are going to compare the time it takes for each algorithm to create a model.

1.3 Thesis Structure

In chapter two, a brief description of the classification algorithms is given in order to help the reader understand the basics of how each algorithm works. There is also a summary of the algorithms used as a part of the preprocessing for some datasets.

In chapters three to five we study the datasets that were donated to UCI. In each chapter a brief description of the dataset, the feature is contains and it's class attribute is presented in order to gain a better understanding of the dataset. Afterwards the processing methodology that was used is explained and finally the results of each algorithm (accuracy and running time until completion) is presented.

Finally at chapter 6 are the conclusions based on the overall results of all algorithms.

2 Background

In this section, all the algorithms that are going to be used are briefly explained, in order for every reader to gain basic understanding of them.

Ensemble Methods

Ensemble methods are techniques that create multiple "models", which are combined to create improved results. Models are usually predictive models such as decision trees, neural networks or support vector machines (Zhou, 2012). There are two major categories of ensemble methods, based on the way the predictive models (also called based learners) are created.

The algorithms where the models are created sequentially are called boosting algorithms, with Adaboost as a representative while on the other hand when they are created in parallel they with Bagging as a representative. The ensemble algorithms we are going to examine are Adaboost and XGBoost that belong to the Boosting framework and Random Forests from the Bagging framework.

2.1 AdaBoost

AdaBoost algorithm is a machine learning meta-algorithm that was first introduced in 1996 (Freund & Schapire, 1997). Suppose we are given a training set of N samples $\{(x_1, y_1), \dots, (x_N, y_N)\}$ and our initial distribution D_1 $that\ is$ $\{\frac{1}{N}, \dots, \frac{1}{N}\}$. At initialization, all training examples begin with the same weight and our weight vector w_1 at step 1 is equal to the distribution D_1.

At each step, some classifier creates a simple model (also referred to as hypothesis). The weak learners (or base learners) are usually decision trees with a depth of 1 (also called stamps). The error of this model is then calculated by adding the weights of all the misclassified instances with the mathematical formula:

$$\varepsilon_\tau = \Pr_i \sim D_t[h_t(x_i) \neq y_i] = \sum_{i=1} w_i \cdot I[y_i \neq h_t(x_i)]$$

A weight a_t is calculated for that hypothesis to minimize the error of the final model where $a_t = \frac{1}{2}\ln(\frac{1-\varepsilon_\tau}{\varepsilon_\tau})$ and new weights are calculated for the training set. Finally we update our distribution $D_{t+1} = \frac{D_t(i)e^{-a_t y_i h_i(x_i)}}{Z_t}$ where Z_t is a normalization factor. We set out weight vector to be equal to our new distribution and we repeat this process.

2.2 XGBoost (Gradient Boosting Framework)

Gradient Boosting is an ensemble method based in the boosting framework. Like all boosting algorithms, it builds a strong learner by sequentially adding weak learners. The initial model $f_0(x)$ should be a function that minimizes the loss function or the MSE:

$$f_0(x) = argmin_\gamma \left(\sum_{\iota=1}^{n} L(y_i, \gamma) \right) = argmin_\gamma \sum_{\iota=1}^{n} (y_i - \gamma)^2$$

By taking the first differential of the above equation we can see that $f_0(x)$ minimizes at the mean. This means that the first prediction for all out data is the average of the class we want to predict. After calculating the differences (residuals) r_i between the actual values of the class and our predictions, we fit a weak learner $h_1(x)$ using the training set$\{(x_i, r_i)\}_{i=1}^{n}$.

Then a weight γ_1 is calculated for that model using the formula:

$$\gamma_m = argmin_\gamma \sum_{\iota=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

Finally, we update out model $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

2.3 Random Forest

Random Forest is a supervised learning algorithm that creates a random collection (“forest”) of trees. The trees it creates are Decision Trees that are trained using the bagging method.

Bagging is selecting a random sample from the dataset with replacement. The extension (Ho, 1995) that Random Forest adds is that the Decision Tree is trained in a subset of the available features.

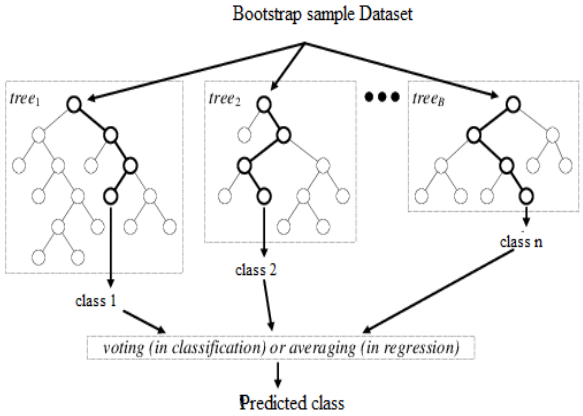

Figure 1: Random Forest

After all the trees are created, the prediction for the class of any given instance is done using voting (for classification).

2.4 Decision Trees (C. A. R. T implementation)

Decision Tree algorithms ID3 (Quinlan J. R., 1986) and C4.5 (Quinlan J. R., 2014) were invented by John Ross Quinlan and both build a tree from a set of training data using the concept of information gain. Around the same time with Quinlan, Breiman created his own implementation of decision trees named Classification and Regression Trees (C.A.R.T) (Breiman, Friedman, Olshen, & Stone, 1984) and is very similar to C4.5.

A key difference between C.A.R.T and C4.5 is that it supports numerical target variables (regression) and it does not compute rule sets. The Python package skicit-learn uses an optimized version of the C.A.R.T algorithm.

Let us suppose we are given a set of training vectors S = {s1… s_l} of classified samples $s_i \in R^n$ and a label vector y. At each node N, for each feature j and given a threshold t_N we partition the data into $Q_{right}(\{j, t_N\})$ and $Q_{left}(\{j, t_N\})$ where $Q_{right}(\{j, t_N\}) = (x, y)|x_j > t_N$ and $Q_{left}(\{j, t_N\}) = Q - Q_{right}$.

The quality of the split is called impurity and is measured using a function H(), but the quality measures used are usually entropy or Gini index. The choice of the measuring function depends on the task being solved, classification or regression.

Then we calculate the Gini index for node N and split at the j feature:

$$G(N, \{j, t_N\}) = \frac{n_{left}}{N_N} H\left(Q_{left}(\{j, t_N\})\right) + \frac{n_{right}}{N_N} H\left(Q_{right}(\{j, t_N\})\right)$$

We split our data on the feature that minimizes $G(N, \{j, t_N\})$ and continue the same procedure recursively for Q_{left} and Q_{right} until the stopping criteria are met.

2.5 Naïve Bayes

Naïve Bayes is a conditional probability model that makes use of Bayes' Theorem to calculate the probability for each class label C_k . Given a training set of N feature and an instance x represented by a vector $x = \{x_1, \dots, x_N\}$ using the formula:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

We are interested in calculating the numerator of our fraction that is equal to the joint probability model $p(C_k, x_1, \dots, x_N)$.Using the chain rule we can rewrite this probability as follows:

$p(C_k, x_1, \dots, x_N) = p(x_1|x_2, \dots, x_N, C_k)p(x_2|x_3, \dots, x_N, C_k) \dots p(x_{N-1}|x_N, C_k)p(x_N|C_k)$

This classifier is called "naïve" because we assume that all features are mutually independent, conditional on the category C_k.

Based on this assumption we can do the following approximation:

$$p(x_i|x_{i+1}, \dots, x_N, C_k) = p(x_i|C_k)$$

so now we can rewrite $p(C_k, x_1, \dots, x_N) = \frac{1}{K} p(C_k) \prod_{i=1}^{N} p(x_i|C_k)$ where $\frac{1}{K}$ is a normalization factor.

After the model is build given an instance we can calculate the probability to belong in each class label. Then we apply a decision rule to find the class of this instance. The most common decision rule is known as the maximum a posteriori rule, which means we assign the class label with the highest probability.

2.6 Support Vector Machines

Support Vector Machines, also known as Support Vector Networks (Boser, Guyon, & Vapnik, 1992) are supervised learning methods that can be used for both classification and regression.

This algorithm performs classification by calculating the hyperplane (a simple line in two dimensions) that best differentiates the classes of the dataset.

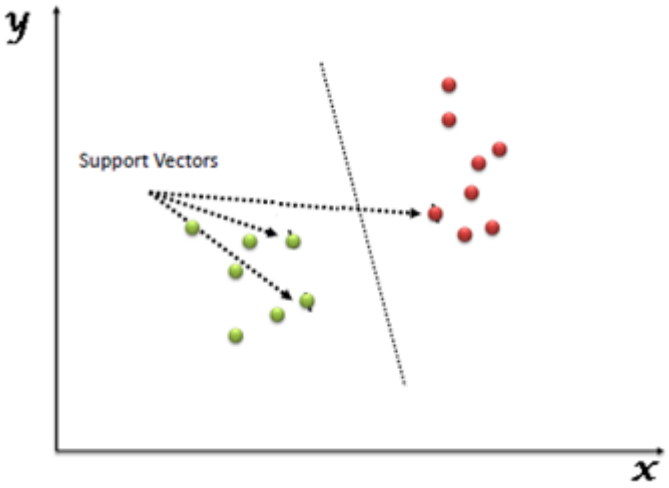

Figure 2 : Support Vectors

Out of an infinite number of hyperplanes SVM calculates the optimal using a subset of training samples, the support vectors. Support vectors are data points that lie closest to the decision surface and are the most difficult to classify.

Let's suppose that our separating line has a form of $w \cdot x - b = 0$ where the variable b let's us move the line without having to pass through the origin O and w is the weight vector we have to calculate in order to find our hyperplane. In order to maximize the margin between the classes we need the parallel hyperplanes to the optimal one that are closest to the support vectors of either class. Those hyperplanes can be described by equations $w \cdot x - b = 1$ and $w \cdot x - b = -1$.

The distance between these hyperplanes is $\frac{2}{|w|}$, thus to maximize the margin we have to minimize $||w||$. To ensure that no data points are going to be inside our margin then for every i either $w \cdot x_i - b \geq 1$ or $w \cdot x_i - b \leq -1$.

That is a quadratic constrained problem and can be solved by Lagrangian multiplier method. The Lagrangian formulation of the problem is:

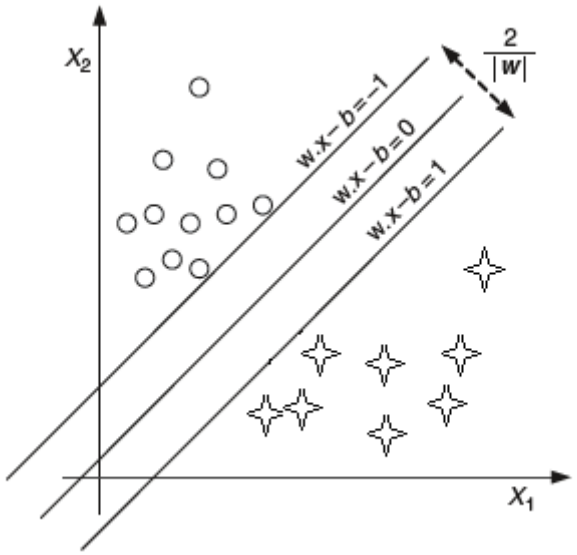

Figure 3 : Margins of SVM

$$\min L_p = \frac{1}{2}||w||^2 - \sum_{i=1}^{l} a_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} a_i$$

subject to :

$$\forall\, i\,, a_i \geq 0$$

where α are the Lagrange multipliers and l is the number of training instances.

Instead of solving this problem in the Scikit implementation, the dual problem is solved with the help of the Kuhn – Tucker theorem. The solution will be the same for the original problem.

The dual problem formulation is:

$$max L_D(a_j) = \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l} a_i a_j y_i y_j (x_i \cdot x_j)$$

subject to:

$$\sum_{i=1}^{l} a_i y_i = 0 \; and \; a_i \geq 0$$

Knowing the a_i we can calculate the weights $w = \sum_{i=1}^{l} a_i y_i x_i$ and given a known point of x_i features we can classify it by looking at the sign of

$$f(x) = w \cdot u + b = \left(\sum_{i=1}^{l} a_i y_i x_i \cdot u\right) + b$$

.

2.7 Multilayer Perceptron (Neural Networks Framework)

Artificial Neural Networks is a computational model inspired by nature and the way biological neural networks in the human brain works. It consists of a collection of nodes called artificial neurons. For every input given a weight is assigned for every feature of this input and a bias is included.

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. Then the sum of all the inputs multiplied by their weights goes through an activation function producing a single output producing the final output of our model.

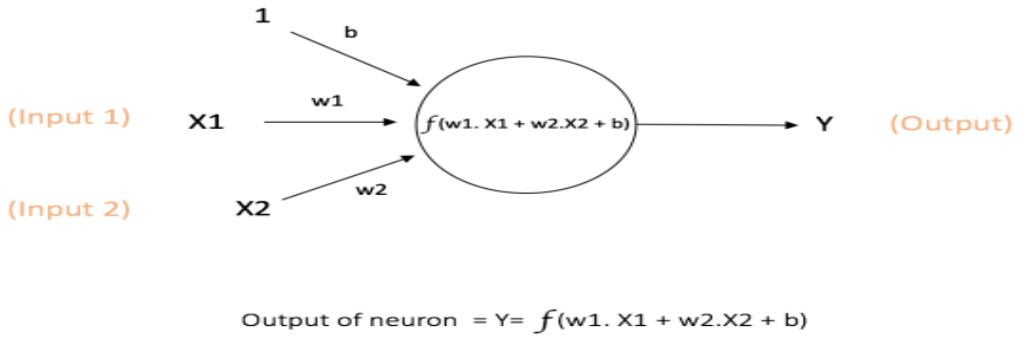

Figure 4 : An artificial neuron

It was proven that a model with a single perceptron can only solve linear separable problems (Minsky & Papert, 1969) so then need to expand this had arisen. That led to the use of Multi-Layer Perceptron models that consists of many neurons divided into layers.

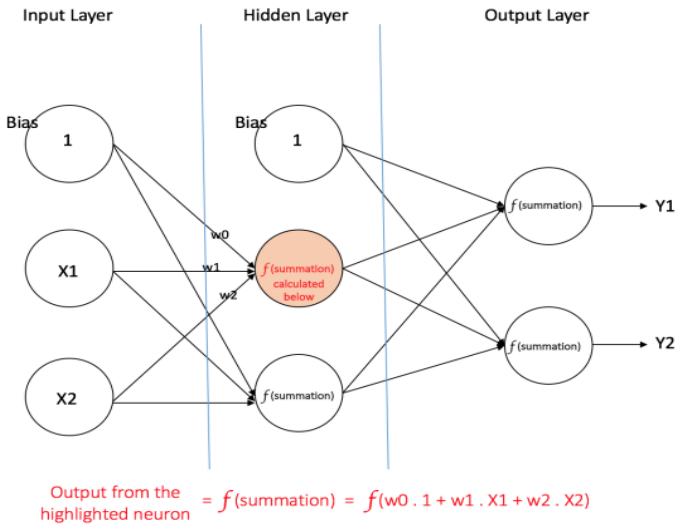

Figure 5 : A Multilayer Perceptron

The leftmost layer, known as the input layer, consists of a set of neurons {xi|x1,x2,...,xm} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w1x1+w2x2+...+wmxm, followed by a non-linear activation function g(·):R→R - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

Although the weights of the nodes for each layer are initially assigned at random, in order to maximize the capabilities of a neural network they should be tuned based on the training data. The backpropagation algorithm is applied on a feed-forward neural network to adjust the weights at each node based on the output of the outcome of the nodes of the next layer that it is connected to.

The advantages of Multi-layer Perceptron are:

- Capability to learn non-linear models.

The disadvantages of Multi-layer Perceptron (MLP) include:

- MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires tuning a number of hyper parameters such as the number of hidden neurons, layers, and iterations.
- MLP is sensitive to feature scaling

2.8 K-NN

K-NN algorithm is a non-parametric method used for classification and is amongst the simplest classification algorithms. Given a set of training examples and a custom distance metric (usually the Euclidian metric is used) we can predict the class of a new instance by a plurality vote of its neighbors.

Using the distance metric provided, we are looking for the k nearest neighbors of the new instance, and based on their label we decide for the label of the given instance.

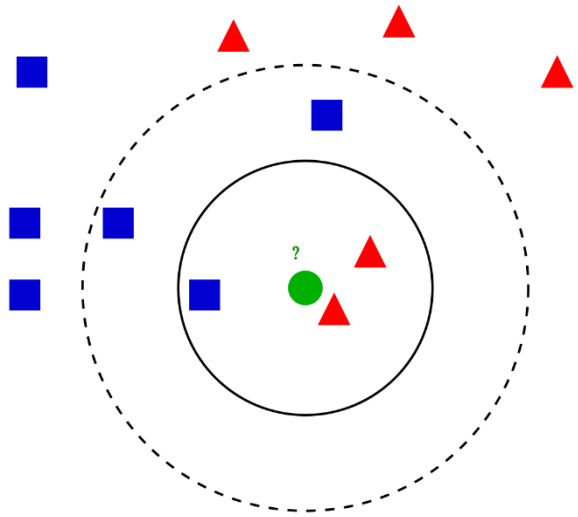

Figure 6 : 3-NN and 5-NN example for unlabel instance (green dot)

3 Case Study: Immunotherapy and Cryotherapy datasets

3.1 Introduction

Immunotherapy and Cryotherapy are two medical datasets donated by the dermatology clinic of Ghaem Hospital, Mashhad, Iran to the UCI Machine Learning Repository (Khozeimeh, et al., 2017). A study has been conducted on these datasets where a fuzzy rule-based method was applied to create a model with an accuracy of 83.33% in the Immunotherapy dataset and 80% in Cryotherapy dataset. Our purpose is to compare the performance of the classic classification algorithms and to build a more effective model in terms of accuracy.

3.2 Preprocessing

3.2.1 Data Merging

In order to be able to build a model with every algorithm both Cryotherapy and Immunotherapy datasets had to be combined resulting into a single dataset. The Immunotherapy dataset has an extra feature ('Induration_diameter') resulting in the addition of 90 missing values for the instances that come from the Cryotherapy dataset. A zero value was assigned to them.

3.2.2 Data Transformation – Data Cleaning

The class attributes for both Immunotherapy and Cryotherapy datasets where 0 if it was negative and 1 if the Response to treatment was positive. After merging the two datasets, the class values for the instances that come from the Immunotherapy dataset remained the same, while the ones coming from the Cryotherapy dataset were set to 2 and 3 respectively. In addition, the Cryotherapy dataset contains a duplicate value, two instance with the exact same attributes. It was considered noise and was removed from the dataset to avoid disturbances in the underlying distribution of the data.

Table 1
Features employed in the resulting dataset

Feature Name	Values	Mean +/- SD
Response to treatment	No with Immunotherapy (0)	
	Yes with Immunotherapy (1)	
	No with Cryotherapy (2)	
	Yes with Cryotherapy (3)	
Gender	88 Man (1) / 92 Woman (2)	
Age	15-67	29.82 ± 12.83
Time elapsed before treatment (month)	0-12	7.44 ± 3.25
Number of warts	1-19	5.82 ± 3.9
Type of warts	1-Common (101)	
	2-Plantar (31)	
	3-Both (48)	
Surface area of warts(mm)	4-900	90.7 ± 133.9
Induration diameter	0-70	7.16 ± 14.1

Figure 7 : Overview Table of resulting dataset

3.2.3 Class balancing

Cryotherapy dataset is balanced having almost equal number of positive and negative responses to treatment, but that is not the case for the Immunotherapy dataset where the positive responses were more.

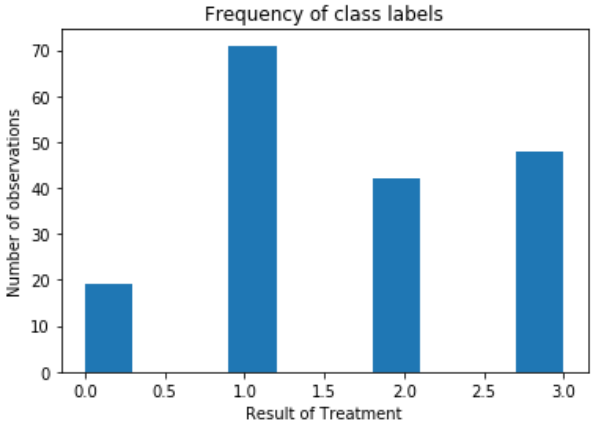

Figure 8 : Count plot of joined dataset classes

In order to balance the dataset we could use random oversampling to create a few dummy data of class 0, or we could use an even more advanced technique like Synthetic Minority Oversampling Technique (S.M.O.TE) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). After experimentation, it was found that the best choice would be to not use any oversampling technique and proceed with the dataset as it is.

3.2.4 Outlier Detection

In figure 9 we notice that the 'Surface Area of Warts' feature has a high variance and that can indicate that there are some outliers. Also in the pair-plot matrix (Fig. 11), we can see two possible outliers of type 3 warts at the Result-Area pair. Looking at the distribution of the data from both Cryotherapy and Immunotherapy datasets, we can see that the 'Surface Area' feature has a Poisson distribution with extreme values.

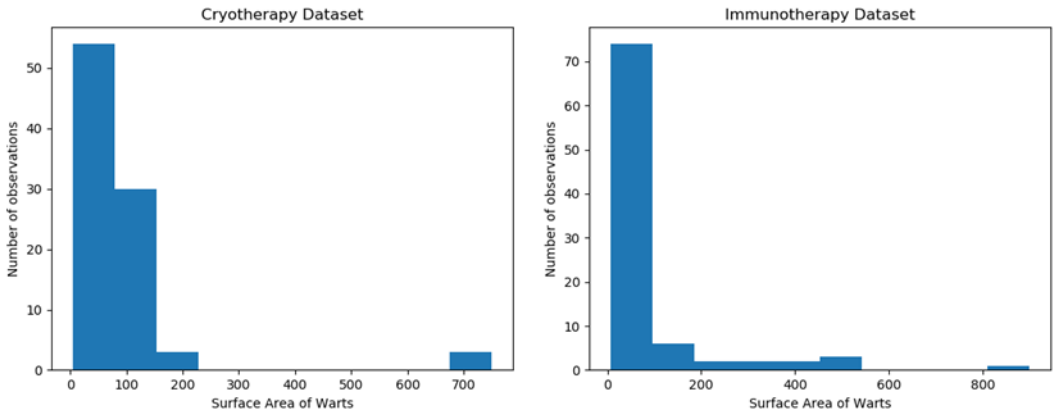

Figure 9 : Count plot of Surface Area for Cryotherapy and Immunotherapy Dataset

After merging the two datasets, we get the following distribution. The fact that the resulting distribution seems Poisson-like and we have a small dataset with 180 observations allows as to assume that those two values are not outliers.

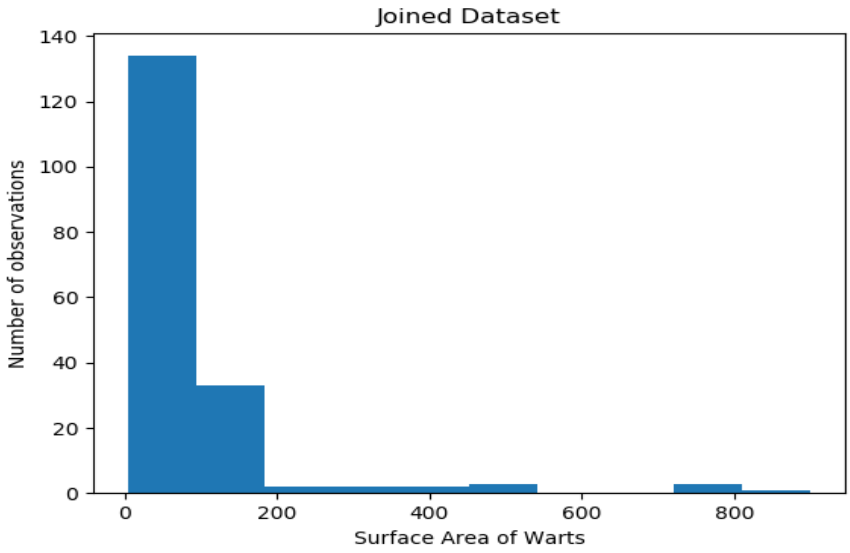

Figure 10 : Countplot of Area feature in joined dataset

3.3 Exploratory Data Analysis

3.3.1 Pairplot Visualization

To gain more insight about the relations between the features, and possibly find some pattern in our data, we are can use a set of visualizations that are going to help us explore and understand our data better. The Python package Seaborn provides us a plethora of different type of plots depending on the type of data you want to visualize (Relational, Categorical, and Distributions etc.). The result of the pairplot function (Fig 11) allows us to see a matrix of scatter plots for every combination of our dataset features. We also have the ability to color our data points based on some feature. Here they are colored based on the Type feature.

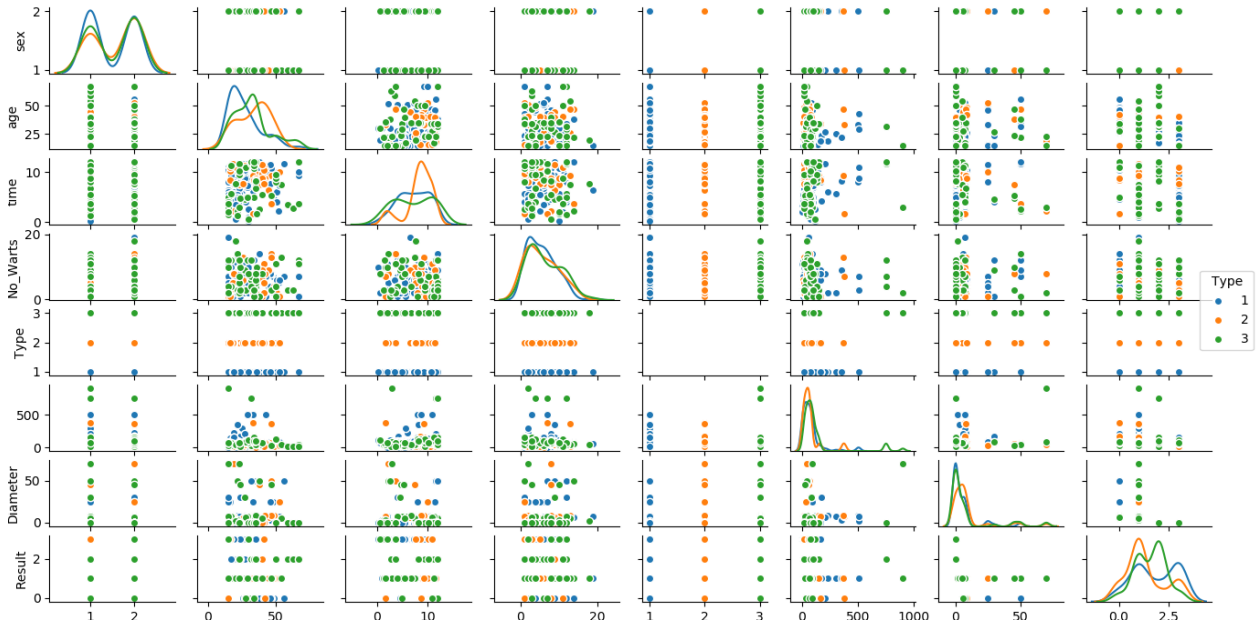

Figure 11 : Seaborn's pairplot

3.3.2 Univariate Analysis

After visualizing the relations between the features, we need to understand how each feature interacts with the class we want to predict. At Figure 12, we can see for every feature the distribution for every class attribute. A feature where those distributions are well separated is a good feature to be used in order to predictions. On the other hand, a feature with overlapping distributions, doesn't add much information. For example at

Figure 12 we can see that for the ‘Number of Warts’ attribute all class attributes have the same distribution, which makes it really hard to distinguish the class only by looking at this feature.

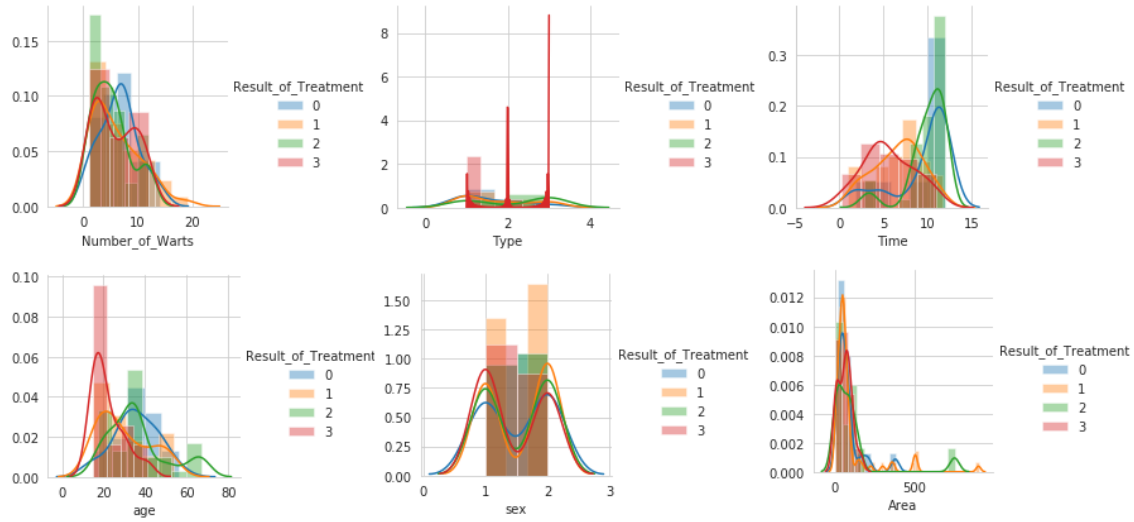

Figure 12 : Distribution plots for every feature colored by class attribute

That’s not the case for ‘Age’ and ‘Time’ attributes where the distributions may not be totally separated but there is a significant difference in the mean.

3.3.3 Boxplot Analysis - ‘Violinplot Analysis’

Another helpful visualization that is also often used in outlier detection is the boxplot. Instead of a regular boxplot we can see at Figure 13 Seaborn’s violinplot which is a combination of a regular boxplot in the center of every ‘violin’, and a distribution plot on the sides.

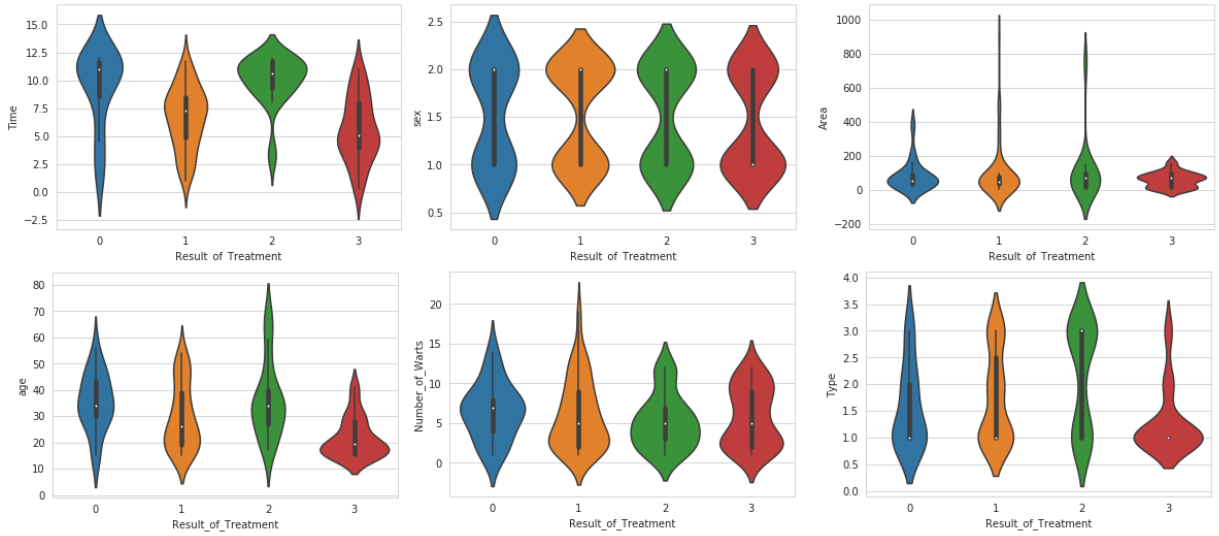

Figure 13 : Violinplot of every dataset feature

In figure 14 we can see that for the feature ‘Time’ classes 1 and 3 have a lower mean classes 0 and 2 have a higher mean but also smaller variance, which makes the date more concentrated around the mean. The use of this feature will help as distinguish the classes easier.

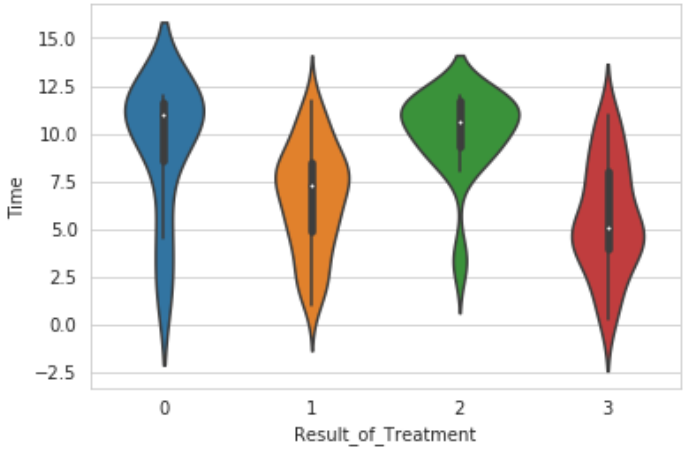

Figure 14 : Violin Plot for the feature Time

On the other hand, as we can see from the violinplot of feature ‘Number of Warts’ (Fig 15) the mean for every class is almost the same which makes this feature weaker compared to the predictive power of ‘Time’.

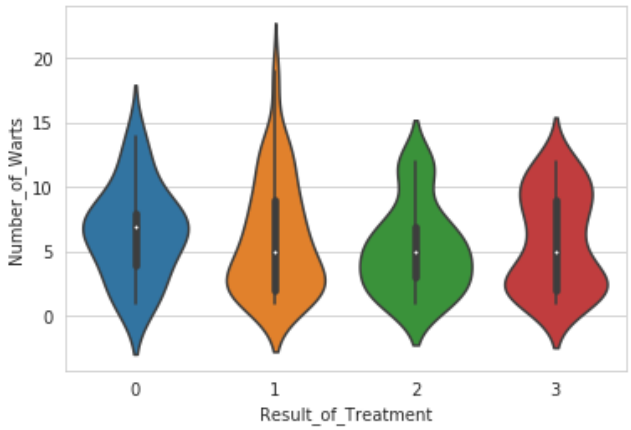

Figure 15 : Violinplot for Number of Warts feature

3.3.4 Correlation Matrix

To select the best features that describe our dataset we also calculated the correlation matrix (Figure 16) using the Spearman method. Because the Spearman correlation coefficient is based on the ranks of the data rather than the actual data, makes it a better choice for examining the relationships between variables with unequal variances, and of non-normal distribution.

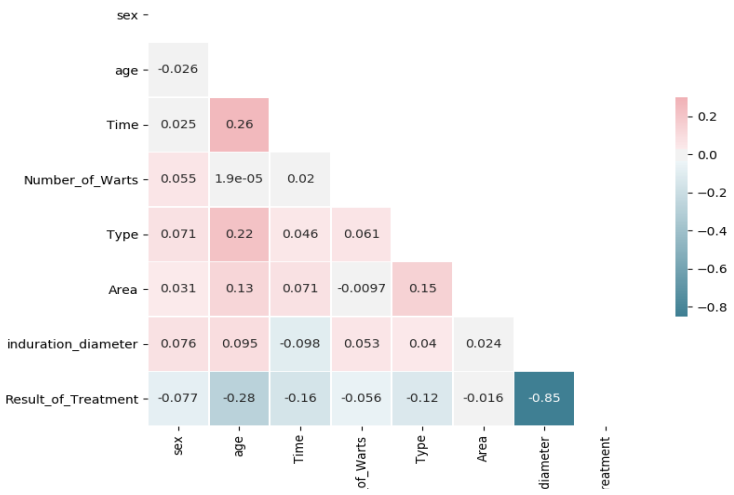

Figure 16 : Spearman Correlation Matrix

The features with the greatest relevance with the Result of Treatment were the induration diameter with a correlation coefficient of -0.85, age with -0.28 and Time with -0.16. The rest of the features had a less significant correlation coefficient and also based on our boxplot and univariate analysis, only the top three features were used to build the models.

3.4 Results

Each algorithm was tested using 10-fold cross validation, with a fixed seed to ensure that all algorithms are going to be trained and tested on the same training and testing sets. The class attribute was considered balanced so the metric that was used to compare the performance of the algorithms was the accuracy of the predictions. Except from the resulting dataset after combining Immunotherapy and Cryotherapy datasets, each algorithm was also tested on the original datasets in order to see if their combination would result in a better performance.

As we can see from tables, 1-9 the highest accuracy on the combined dataset was by XGBoost with a 92.22% accuracy and a standard deviation of 5.6%. For every algorithm except XGBoost and Support Vector Machine, the accuracy on Cryotherapy was higher than both Immunotherapy and combined datasets. Also the accuracy on Immunotherapy dataset was the lowest for every algorithm except Support Vector Machine where Cryotherapy had the lowest accuracy score.

Table 1 : XGB results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,777	0,666	1	0,777	0,555	0,777	1	0,777	1	1	0,833	0,15
Cryotherapy	0,888	1	1	0,888	0,888	0,888	0,666	0,888	0,888	0,888	0,888	0,086
Combined	0,833	0,944	0,944	0,833	0,888	0,944	1	0,944	0,888	1	0,922	0,056

Table 2 : Random Forest results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,888	1	1	0,777	0,777	0,777	1	0,777	0,777	1	0,878	0,1
Cryotherapy	0,888	1	1	0,888	1	0,888	0,777	1	0,888	0,666	0,9	0,1
Combined	0,833	1	0,833	0,833	0,888	0,944	1	0,888	0,888	0,777	0,888	0,07

Table 3 : C.A.R.T results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,777	0,777	1	0,777	0,666	0,777	1	0,777	1	1	0,855	0,12
Cryotherapy	0,888	1	1	0,888	1	0,888	0,888	1	1	0,666	0,922	0,1
Combined	0,777	0,833	0,777	0,833	0,777	0,888	1	0,888	0,833	0,888	0,85	0,06

Table 4 : Adaboost results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,888	0,888	0,888	0,666	0,555	0,777	1	0,777	0,777	1	0,822	0,13
Cryotherapy	0,888	1	1	0,888	0,888	0,888	0,777	1	0,888	0,666	0,888	0,09
Combined	0,777	0,888	0,888	0,833	0,888	0,888	0,944	0,833	0,888	0,777	0,861	0,05

Table 5 : Neural Network results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,888	1	0,777	0,777	0,777	0,777	0,777	0,777	0,666	0,666	0,788	0,092
Cryotherapy	0,888	1	0,777	0,888	0,888	0,888	0,666	0,888	1	0,777	0,866	0,096
Combined	0,722	1	0,888	0,777	0,888	0,777	0,944	0,833	0,833	0,722	0,838	0,087

Table 6 : SVM results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,888	0,666	0,666	0,222	0,555	0,222	0,555	0,666	0,555	0,666	0,566	0,195
Cryotherapy	0,222	0,666	0,444	0,444	0,555	0,777	0,333	0,222	0,444	0,777	0,488	0,193
Combined	0,722	0,944	0,888	0,777	0,888	0,888	0,888	0,722	0,833	0,833	0,838	0,072

Table 7 : Naive Bayes results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,777	0,888	0,888	0,666	0,777	0,666	0,888	0,777	0,777	0,888	0,799	0,083
Cryotherapy	0,888	1	1	0,888	0,888	0,888	0,555	0,888	0,666	0,888	0,855	0,13
Combined	0,611	0,944	0,888	0,722	0,833	0,833	0,833	0,777	0,833	0,888	0,816	0,089

Table 8 : KNN results on both standalone and combined datasets

Dataset\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd
Immunotherapy	0,777	0,555	1	0,888	0,666	0,777	0,888	0,888	0,777	0,555	0,777	0,14
Cryotherapy	1	0,888	1	0,888	1	0,888	0,888	1	0,777	0,888	0,922	0,07
Combined	0,888	0,888	0,777	0,666	0,888	0,888	1	0,777	0,777	0,888	0,844	0,08

Finally at Table 9 we can see the time it took for every algorithm to train, predict and measure the accuracy of the model, for every fold, in total.

Table 9 : Performance results on Immunotherapy & Cryotherapy combined dataset

Algorithm\Fold	1	2	3	4	5	6	7	8	9	10	Mean	Sd	Time
XGBoost	83.3%	94.4%	94.4%	83.3%	88.8%	94.4%	100%	94.4%	88.8%	100%	92.2%	5.6%	0.17
Random Forest	83.3%	100%	83.3%	83.3%	88.8%	94.4%	100%	88.8%	88.8%	77.7%	88.8%	7%	1.54
C.A.R.T	77.7%	83.3%	77.7%	83.3%	77.7%	88.8%	100%	88.8%	83.3%	88.8%	85%	6.5%	0.07
Adaboost	77.7%	88.8%	88.8%	83.3%	88.8%	88.8%	94.4%	83.3%	88.8%	77.7%	86.1%	5.1%	3.75
Neural Network	72.2%	100%	88.8%	77.7%	88.8%	77.7%	100%	83.3%	83.3%	72.2%	83.8%	8.7%	2
SVM	72.2%	94.4%	88.8%	77.7%	88.8%	88.8%	88.8%	72.2%	83.3%	83.3%	83.8%	7.2%	0.06
Naïve Bayes	61.1%	94.4%	88.8%	72.2%	83.3%	83.3%	83.3%	77.7%	83.3%	88.8%	81.6%	8.9%	0.07
KNN	88.8%	88.8%	77.7%	66.6%	88.8%	88.8%	100%	77.7%	77.7%	88.8%	84.4%	8.8%	0.07

4 Case Study: APS Failure at Scania Trucks Dataset

4.1 Introduction

Scania, a major Swedish manufacturer of commercial vehicles, donated the APS failure dataset to UCI Machine Learning Repository. It contains data collected from heavy Scania trucks in everyday usage. The dataset is already split in a testing and a training set. The datasets consists of two classes, where the positive class corresponds to trucks with failures associated to the Air Pressure System (APS) while the negative class corresponds to trucks with failures that are not connected to it.

It has 170 feature columns that have been anonymized, for example ‘aa_000’ and ‘ab_000’ are the first two features. There are two types of features, the usual standalone type where one column is one feature and the histogram type. The histogram features are groups of 10 feature columns that in fact are bins of a histogram. For example, the features ‘az_000’-‘az_009’ (Fig 18) belong to a single histogram. There are 70 features that belong to a histogram thus there are seven histograms in our data.

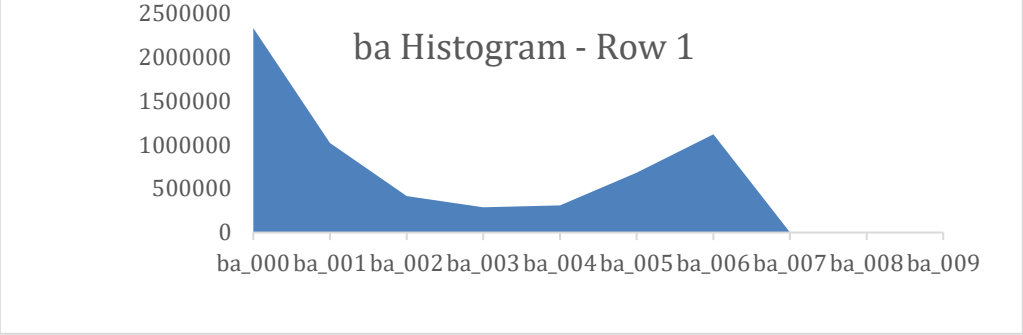

Figure 17 : Histogram of ba_000 - ba_009 features in Row 1

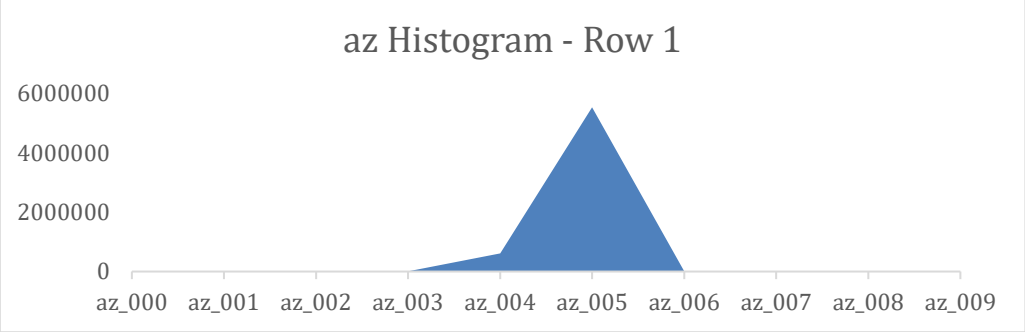

Figure 18 : Histogram of az_000 - az_009 features in Row 1

4.2 Preprocessing

One of the most common tasks of a data analyst, when preparing a dataset to be used for constructing a predictive model, is to handle the missing values of his dataset (if those exist). In this particular dataset, 850.015 missing values take up to 8.3% of the total data.

4.2.1 Removing insignificant rows

As we can see in figure 19 that is the distribution of the count of missing values per row, that there are rows with more than 150 missing values out of 170 features. These rows can be safely removed, as they do not add any information to the dataset. For the final models only rows with less than 60 missing values were kept.

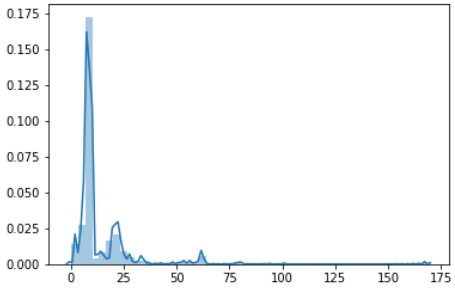

Figure 19 : Frequency plot of number of missing values

Removing rows should be done in consideration with the class that those rows belong to. Due to the fact that this is a highly imbalanced dataset, having only 1000 rows that belong to the positive class and 59.000 rows that belong to the negative class, removing can reduce even more the instances that belong to the positive class. For that reason only rows with more than 60 missing values where removed resulting in a dataset with 58.026 rows with 947 instances belonging in the positive class and 57079 instances belonging in the negative class.

4.2.2 Filling the missing values

One of the most common ways to replace missing values is to substitute them with either the average or the median of that feature. Using the median or a simply average to replace the missing values isn't a valid method for this dataset, cause there are features (columns) that have a really high percentage of missing values, (e.x columns

'ab_000','bn_000','bo_000','bp_000' have 46329,44009,46333 and 47740 missing values respectively) and doing so would disrupt the underlying distribution.

I decided to use a machine learning approach using the mean of the values of the 50 nearest neighbors to estimate each missing value (Ozan, Riabchenko, & Kiranyaz, 2016). The metric used to compute the distance between instances was the following:

$$D(x,y) = \sum D_i(x_i, y_i) \; the \; distance \; between \; rows \; x \; and \; y$$

$$where \; D_i(x_i, y_i) = \begin{cases} 1 \; if \; x_i \; or \; y_i \; is \; na \\ |x_i - y_i| \end{cases}$$

$$and \; x_{i,} \; y_i \; are \; the \; features \; of \; x \; and \; y \; rows \; at \; i \; column$$

4.2.3 Dimensionality Reduction

Another problem about this dataset is the number of columns it has making the computation times longer. To overcome this I tried various dimension reduction techniques such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD).

To determine the best dimension to use for our classification task, as well as the best technique for dimension reductions, I trained an XGB model and evaluated it using 10 fold cross validation and taking the average cost.

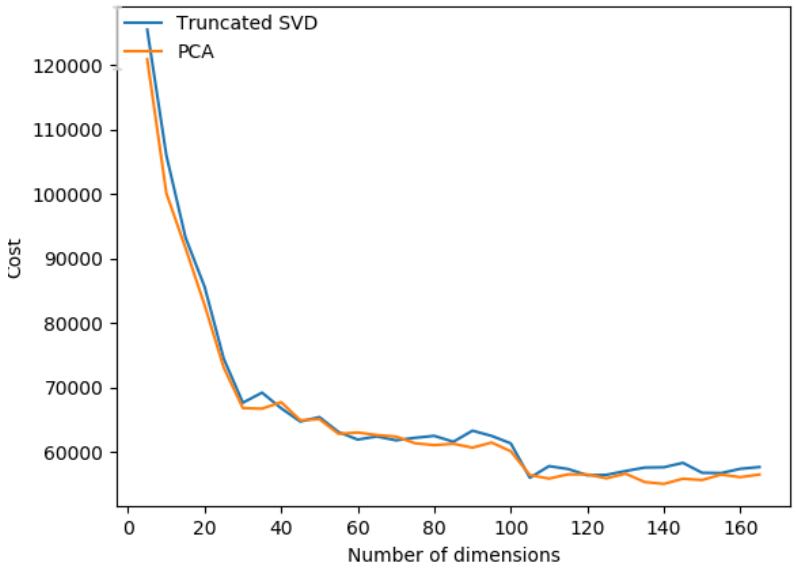

Figure 20 : Performance of PCA and SVD per Number of Neighbors

We can achieve minimum cost projecting our data into 140 dimensional space using PCA algorithm.

4.2.4 Class balancing

Finally the last thing we have to take in consideration is that this dataset is highly imbalanced. It has only 947 instances belonging to the positive class and 57079 samples belonging to the negative class.

To help build better model with all algorithms there are many techniques for oversampling. Oversampling is the creation of dummy instances that are going to be added to our dataset. Those instances are going to be of the desired class, in our case the positive class. There are many algorithms for oversampling and I chose to use Synthetic Minority Oversampling Technique (S.M.O.TE).

Figure 21 : Performance of S.M.O.TE per number of Neighbors

The best results where achieved when creating synthetic samples considering the 15 closest neighbors.

4.3 Results

To compare the performance of every algorithm the dataset was split using 10-fold cross validation. A fixed seed was used to ensure that all algorithms use the same training and testing subset. In Table 10 we can see the cost of each algorithm in each fold, with XGBoost algorithm having the minimum average cost of 12253 units and the second lowest standard deviation of 5.6 after Naïve Bayes classifier.

Table 10 : Performance results on Scania dataset

Algorithm\fold	1	2	3	4	5	6	7	8	9	10	Mean	StDev	Time
XGBoost	12010	12370	11950	12110	11640	12120	12590	12650	12460	12630	12253	321,654	2255.5
CART	20000	22480	29200	23240	23210	33130	21140	20640	33330	25410	25178	4738,55	336.9
Random Forest	12660	12330	15940	12410	12430	14960	12690	13360	13570	11310	13166	1298,22	113.1
Naïve Bayes	16660	16300	16700	16710	16690	16640	16640	16660	16660	16660	16632	112,942	5.7
Adaboost	15600	14520	16950	15910	14310	15220	15880	14240	14030	16970	15363	1029,64	1130.3
SVM	29830	26410	49750	29000	29190	62380	38670	36530	27130	49230	37812	11570,1	9064.4
Neural Network	13040	16580	19620	16720	13630	13370	13510	12940	13860	14830	14810	2064,12	794.7
KNN	18920	20150	18940	18930	18900	18910	19910	18950	18920	19910	19244	492,467	641.6

5 Case Study: S.C.A.D.I DATASET

5.1 Introduction

Self-Care Activities Dataset contains 206 features of 70 children with physical and motor disabilities based on International Classification of Functioning, Disability and Health for Children and Youth (ICF – CY) (Zarchi, 2018). Our first two features are gender (1 for male and 0 for female) and age while features 3-205 are self-care activities based on ICF-CY. An ICF-CY code consists of four levels where the first level classifies general activities (Body Functions, Body Structures, Activities and Participation and finally Environmental Factors). Self-Care problems is a subset of Activities and Participation and starts with d5.

The prediction class consists of the following seven labels:

1. class 1 : Caring for body parts problem
2. class 2 : Toileting problem
3. class 3 : Dressing problem
4. class 4 : Washing oneself and Caring for body parts and Dressing problem
5. class 5 : Washing oneself, Caring for body parts, Toileting and Dressing problem
6. class 6 : Eating, Drinking, Washing oneself, Caring for body parts, Toileting, Dressing, Looking after one's health and Looking after one's safety problem
7. class 7 : No problem

5.2 Exploratory Data Analysis

As we can see at figure 22 class 6 is the most common in our dataset while classes 1, 3 and 5 have only two, one and three cases respectively.

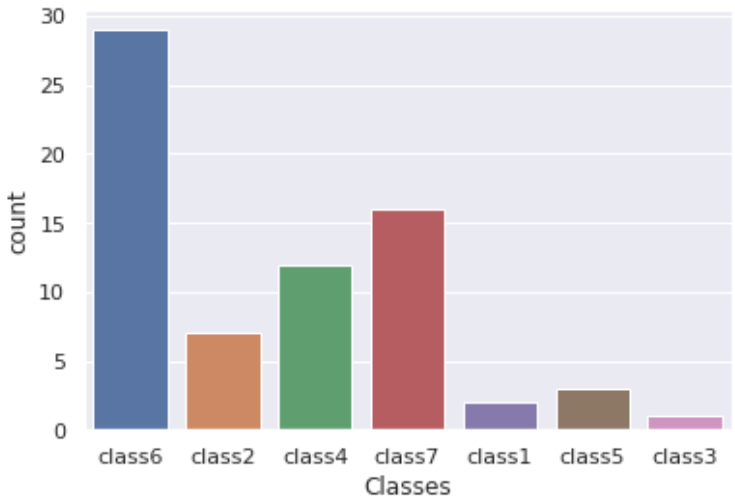

Figure 22 : Count plot of class labels

As we can see due to the small size of this dataset the classes are not equally distributed between genders. There were only female cases of class 5 and 3 and only male cases of class 1 and we can see a huge difference in proportion in class 6.

Figure 23 : Distribution of class labels per sex

5.3 Preprocessing

Using Seaborn's cluster map function we can see the various clusters of instances (using hierarchical algorithm) but the first thing we can notice is that a black column in the cluster map means that all instances have the same value (zero) in that feature. For example features d 5602-8 and d 5602-9 has the value 0 across all dataset. When a column has the same value for all the instances of a dataset means that all instances have

the same feature which adds zero information that would help us with our classification task. As a result we can drop 63 out of 202 columns.

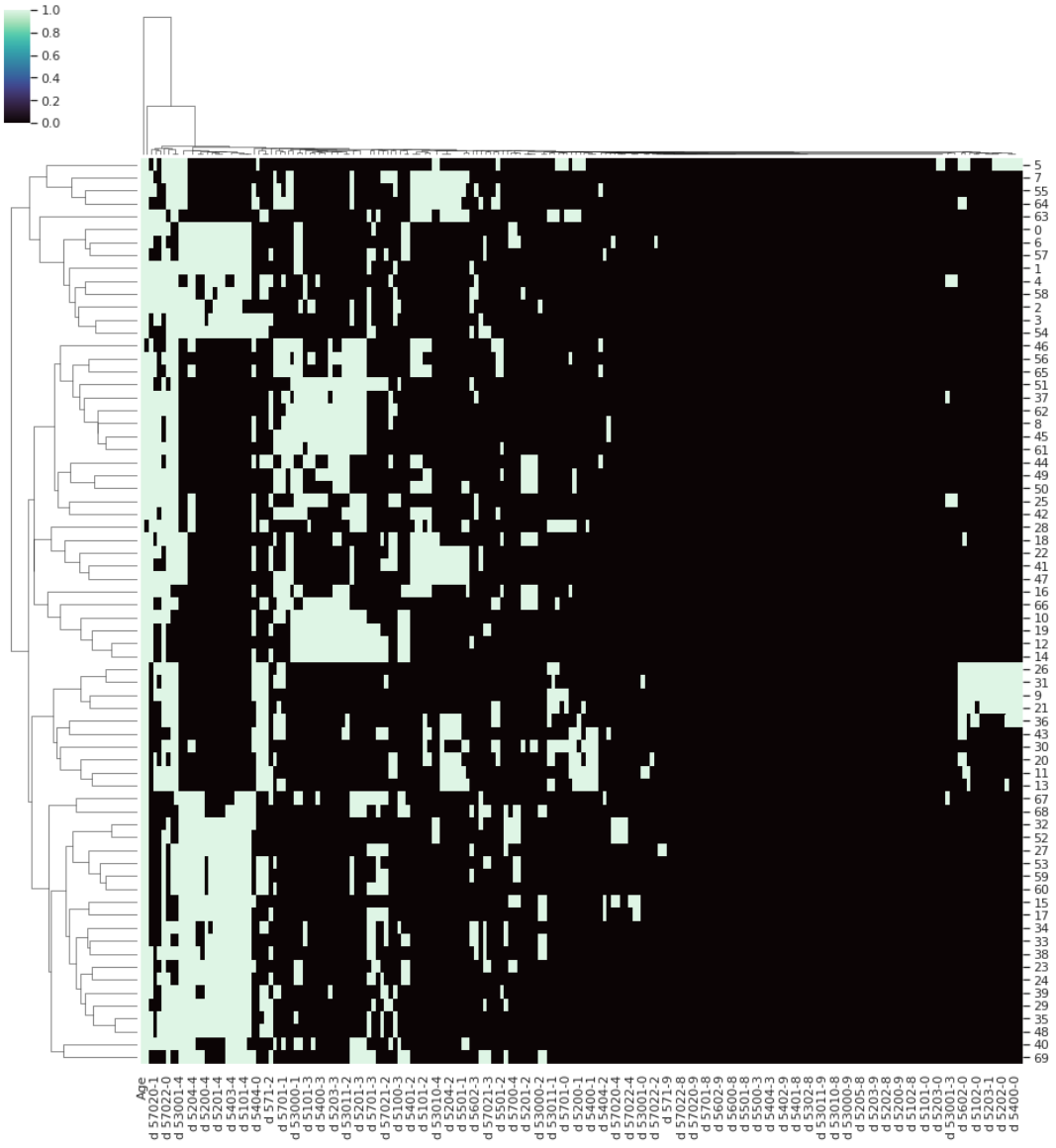

Figure 24 : Seaborn's clustermap

After using Scikit's LabelEncoder class we encode our class labels from strings to numbers in order for all algorithms to be able to process the data. For this dataset the problems we have to overcome is the huge number of features and the fact that some of the class labels are undersampled.

Three kinds of preprocessing where tested on this dataset:

- Use all the features without using any oversampling technique to balance the class feature.
- Use all the features using RandomOversampler as an oversampling technique.
- After training an XGB model using the first preprocessing we extract the feature importance and used only the best features without any oversampling.

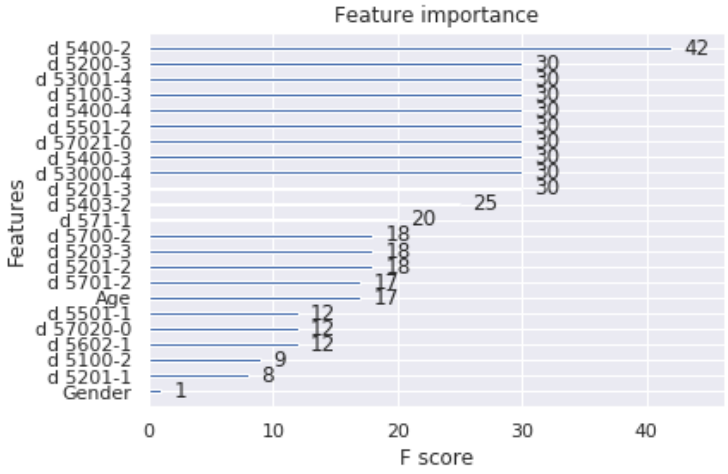

Figure 25 : Importance of Features from XGBoost model

At figure 26 we can see that using oversample had the best average accuracy after using 10-fold cross validation as an evaluation technique across all algorithms. That is not the case for XGB that had the best results when using only the top 12 features (those that had an f-score that was equal or greater than 30) without using any oversampling.

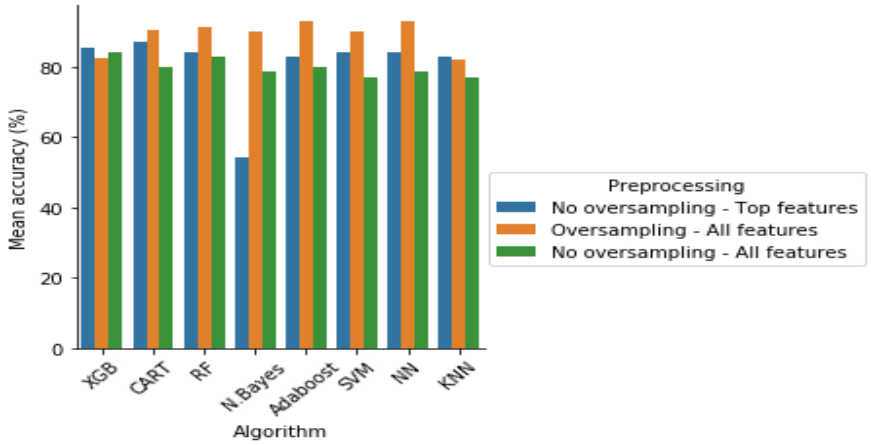

Figure 26 : Performance of preprocessing type for each algorithm

5.4 Results

To compare the performance of every algorithm the dataset was split using 10-fold cross validation. A fixed seed was used to ensure that all algorithms use the same training and testing subset. In Table 11 we can see the accuracy of each algorithm in each fold, with Neural Network algorithms having the highest average accuracy of 93.18% and the lowest standard deviation.

Table 11 : Performance results on S.C.A.D.I dataset

Algorithm\fold	1	2	3	4	5	6	7	8	9	10	Mean	StDev	Time
XGBoost	81,81	54,54	100	80	70	100	90	70	100	80	82,64	14,42932933	13,15
CART	90,9	81,81	90,9	80	70	100	100	90	100	100	90,36	9,835613301	0,05
Random Forest	100	72,72	100	80	80	100	100	90	90	100	91,27	9,911702982	1,59
Naïve Bayes	100	90,9	100	90	80	100	90	80	90	80	90,09	7,750670939	0,05
Adaboost	100	100	100	90	70	100	90	90	90	100	93	9	3,38
SVM	100	72,72	100	80	80	100	100	90	90	90	90,27	9,475539879	0,11
Neural Network	100	81,81	100	90	80	100	100	90	90	100	93,18	7,522556015	2,2
KNN	100	81,81	100	80	80	100	100	90	90	100	92,18	8,482384629	0,03

6 Conclusions

Every Data Analysts dream would be to be able to use an algorithm that had the best performance in terms of accuracy and time of execution, regardless the dataset he tests it. There are many researches on the performance of Data Mining algorithms either on a particular problem, for example image recognition (Martin Weis, 2017) or a more empirical comparison on different datasets (Rich Caruana, 2006) like the one you are reading now.

Even though XGBoost won the competition against the other algorithms in the first two datasets, could not outperform the Multilayer Perceptron in the third case study.

The outcome from every research, including this one, is the same, no single algorithm performs best across all datasets (Christopher Sibona, 2012). There are many conclusions that can be drawn and several statements to be formulated, such as the boosting and bagging algorithms have an increased efficiency compared to older algorithms (Kristína Machová, 2006). The comparison between boosting and bagging tends to be in favor of boosting with boosting based algorithms having better results.

In many cases, methods that clearly perform poorly on average can outperform every other algorithm. It is a Data Analysts duty to have in-depth knowledge of the mechanics of Data Mining methods, in order to be able to choose the most fitting method to the problem he is trying to solve.

The classification of facts, the recognition of their sequence and relative significance is the function of science, and the habit of forming a judgment upon these facts unbiased by personal feeling is characteristic of what may be termed the scientific frame of mind.

—Karl Pearson, The Grammar of Science

Bibliography

Arthur, S. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, pp 210-229.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *COLT '92 Proceedings of the fifth annual workshop on Computational learning theory* (pp. 144-152). Pittsburgh: ACM .

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* New York.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 321–357.

Christopher Sibona, J. B. (2012). A Statistical Comparison of Classification Algorithms on a Single Data Set. *AMCIS.*

Freund, Y., & Schapire, R. E. (1997). Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of computer and system sciences*, pp. 119-139.

Ho, T. K. (1995). Random Decision Forests. IEEE.

Khozeimeh, F., Alizadehsani, R., Roshanzamir, M., Khosravi, A., Layegh, P., & S. Nahavandi. (2017, 1 2). An expert system for selecting wart treatment method. *Computers in Biology and Medicine, vol. 81*, pp. pp 167-175.

Khozeimeh, F., Azad, F. J., Oskouei, Y. M., Jafari, M., Tehranian, S., & Alizadehsani, R. (2017). Intralesional immunotherapy compared to cryotherapy in the treatment of warts. *International Journal of Dermatology*. doi:DOI: 10.1111/ijd.13535

Kristína Machová, M. P. (2006). A Comparison of the Bagging and the Boosting Methods Using the Decision Trees Classifiers. *Computer Science and Information Systems, 3*.

Martin Weis, T. R. (2017). Comparison of different classification algorithms for weed detection from images based on shape parameters. *Image Analysis for Agricultural Products and Processes, BAB Volume 94.*

Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry, The MIT Press, 1969. The MIT Press.

Ozan, E. C., Riabchenko, E., & Kiranyaz, S. (2016). An Optimized k-NN Approach for Classification on Imbalanced Datasets with Missing Data. *Advances in Intelligent Data Analysis XV*. Springer.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, Volume 1*(Issue 1), pp. 81-106. doi:https://doi.org/10.1007/BF00116251

Quinlan, J. R. (2014). *C4.5: Programs for Machine Learning*. Elsevier.

Rich Caruana, A. N.-M. (2006). An Empirical Comparison of Supervised Learning Algorithms. *23 rd International Conference on Machine Learning*. Pittsburgh.

Zarchi, M. a. (2018). SCADI: A standard dataset for self-care problems classification of children with physical and motor disability. *International Journal of Medical Informatics*.

Zhou, Z.-H. (2012). *Ensemble Methods : Foundations and Algorithms*. Chapman & Hall / CRC Press.