ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Econometrie & Operationele Research

Improving CLASSY: Reviewing different frequent pattern mining approaches

Name student: Quinten Obbink

Student ID number: 474183

Supervisor: Dr. Hakan Akyuz

Second assessor: MSc. Hong Deng

Date final version: 4-7-2021

Abstract

Over the last decades, data mining has been one of the most researched topics for data analysis. A popular example of data mining is classification, in particular multiclass classification. A plethora of methods and approaches for multiclass classification exist, but most methods only focus on the accuracy of the classification, disregarding the interpretability of the obtained results. Opposite to this, methods that do provide interpretable results often achieve low accuracy results. This leads to an unavoidable trade-off between accuracy and interpretability which has no clear single solution.

The CLASSY algorithm, proposed by Proença and Leeuwen (2020), is one of the few algorithms capable of obtaining accuracy and interpretability results competitive with other state-of-the-art algorithms. CLASSY performs multiclass classification based on rule lists. These rule lists are constructed using patterns mined from a data set using a frequent pattern mining algorithm. This research tries to improve the CLASSY algorithm by using other frequent pattern mining approaches, namely Apriori and ECLAT, instead of the current one FP-growth.

Twenty data sets from the UCI Repository for Machine Learning are used to test this, comparing accuracy, Area Under the ROC Curve (AUC) and computation time results. These results showed that there was no difference between the accuracy and AUC results. The computation times were significantly slower for ECLAT compared to FP-growth, but between FP-growth and Apriori no significant difference was found. This leads to the conclusion that changing the frequent pattern mining algorithm does not result in an improvement in performance measures of the CLASSY algorithm.

Contents

1 Introduction

Ever since the introduction of computers, the concept 'data' has been used in many different areas and currently can be found virtually anywhere. With this massive amount of data comes an increasing desire to analyze the data and understand its features. One of the most researched areas for this is data mining, which can be best described as a process where raw data is transformed into useful information. This is a very broad and general area, but it can be divided into two categories: model building and pattern detection (Hand and Adams 2014). For this research, the focus is on pattern detection.

Over the last decades, most pattern detection methods have implemented some form of machine learning techniques. A well known example of this is classification, which will be addressed in this research. Classification is the process of giving an item a label that best corresponds to the characteristics of that item. In a data context this means categorizing a data set into different classes based on different rules. An easy example of this is classifying people based on gender. Someone who is male is classified in one group and someone who is female in the other. This example is also called a *binary classification*, as the sample is divided into two distinct groups.

However in practice, often there are more than two groups. The problem is then called *multiclass classification*. There are quite some methods to achieve this, but there is a recurring problem with many of these methods. Although the accuracy and efficiency of these methods are often quite high, understanding how the method got its results often is too difficult for a human brain. In other words, the interpretability of these methods is very low. However, methods that do provide interpretable results often cannot reach competitive accuracy results. This leads to a complex trade-off between interpretability and accuracy when choosing a multiclass classification method.

This trade-off has been focus of previous research, but as it is a recent research topic there have not been many solutions. For multiclass classification interpretability is difficult, because often there are many parameters resulting in a complex model which is incomprehensible to the human brain. Proença and Leeuwen (2020) introduce a novel approach to finding interpretable multiclass classifiers using probabilistic rule lists, called CLASSY. The CLASSY algorithm is one of the few interpretable multiclass classification methods, so it raises the question if its performance can be improved while still retaining its interpretability.

The CLASSY algorithm is based on three different theoretical subjects: constructing a rule list, the Minimum Description Length (MDL) principle and mining the patterns for the rule list. The main idea behind the algorithm is constructing rule lists using the MDL-principle. These

areas of the algorithm will stay untouched, because changing those will create an inherently different algorithm which is not the purpose of this research. Hence, only the pattern mining part is left. The pattern mining approach used for CLASSY is a frequent pattern mining approach and is called Frequent Pattern (FP)-growth. To test if CLASSY can be improved, other frequent pattern mining algorithms will be implemented in the algorithm. The following research question can be constructed from this:

Can the CLASSY algorithm be improved by using other frequent pattern mining approaches?

There exist many frequent pattern mining algorithms, but for this research two popular alternatives will be used, namely Apriori (Agrawal, Srikant, et al. 1994) and ECLAT (Zaki 2000). The three pattern mining approaches will be compared using three performance measures: accuracy, Area Under the ROC Curve (AUC) and computation time. To empirically test the effects of changing the pattern mining algorithm twenty data sets from the UCI Machine Learning Repository will be utilized. To be able to use the data sets in the CLASSY algorithm, they first need to be normalized and discretized. This is done using the LUCS-KDD (Liverpool University Computer Science - Knowledge Discovery in Data) DN (Discretisation/Normalisation) software. Some data sets contain missing values which can lead to biased results. The CLASSY algorithm has no built-in function to deal with missing values, so research is done to see if the algorithm can be adjusted to deal with missing values.

Missing values turned out to be a problem which could not be fixed, so the choice was made to not use data sets that contain missing values. After running the algorithm for the twenty data sets and each frequent pattern mining algorithm, the accuracy and AUC results turned out to be exactly the same for the three frequent pattern mining algorithms. The computation times are different between the three frequent pattern mining approaches, so Friedman's test is used in combination with Holm's post-hoc procedure to test if this difference is significant. The computation times were significantly higher for ECLAT compared to FP-growth, but between FP-growth and Apriori no significant difference was found.

So changing the frequent pattern mining algorithm has no effect on the accuracy and AUC results. For the ECLAT algorithm the computation times were significantly slower, but there was no significant evidence that the computation times of the Apriori algorithm were faster or slower. Concluding, changing the frequent pattern mining algorithm leads to no improvements in performance for the CLASSY algorithm.

The remainder of this paper is structured as follows: Section 2 gives an overview of the existing literature on the topic. This is followed by a discussion of the theoretical backgrounds of the algorithm in Section 3. Section 4 contains a review of the different frequent pattern

mining algorithms. Section 5 gives a quick overview of the preparation that is done to the data sets, followed by the computational experiments done with them in Section 6. Finally, the discussion and conclusion are given in Section 7 and 8 respectively.

2 Literature review

This section gives an overview of related work in interpretable classification. In addition, the literature is related back to the CLASSY algorithm to illustrate the differences. Classification is an important area of data mining, so there is a substantial amount of previous research into methods that perform classification well. Most research is done to improve the accuracy of classification methods, but improving the interpretability of these methods has recently gotten more popular.

Classification is possible in many ways using many different methods. One of the earliest, and easiest, classification methods is decision trees. Breiman et al. (1984) and Quinlan (2014) propose algorithms which perform classification accurately. The nature of decision trees makes them easy to interpret, and both algorithms, being CART and C4.5 respectively, are still considered amongst the top-performing decision tree algorithms. Dhebar and Deb (2020) discuss rule extraction from non-linear decision trees where interpretability is otherwise difficult. The downside of using decision trees to perform classification is that they often do not have global optimization criteria, leading to the need for so called *hyperparameters* to avoid overfitting.

Interpretable models are often associated with the use of rules that explain how the classification is being done. These classifications methods are categorized as rule-based classification models. A Bayesian framework is an early example of rule based classification, where rules are learned using Bayesian theory (Buntine 1989). More recently, T. Wang et al. (2017) and Letham et al. (2015) have discussed more accurate interpretable classification algorithms for the Bayesian framework. However, most Bayesian based models, including the previous mentioned, can only perform binary classification and sometimes need up-front statistical assumptions.

Rule lists are a widely used tool to combine the rules that explain the classification into a single interpretable list. Cohen (1995) proposed a novel approach for multiclass classification with the RIPPER algorithm using the Minimum Description Length (MDL) principle. RIPPER is a top-performing algorithm, but was recently improved by Asadi and Shahrabi (2016) to be able to perform global optimisation, as this was one of the shortcomings of the previous version. However, it still requires the use of hyperparameters to avoid overfitting. One of the algorithms which was made to be faster than the RIPPER algorithm is the MCAR algorithm proposed by Thabtah, Cowling, and Peng (2005). MCAR produces accurate results very fast,

but it constructs very complex rules, making interpretability often difficult. The algorithm is improved by Yusof and Refai (2012), leading to a higher accuracy and less rules being generated. However, the rules still remain complex, so interpretability was not improved.

Evolutionary programming is sometimes used to minimize the number of rules and conditions to maximize the interpretability. The ICRM algorithm (Cano, Zafra, and Ventura 2013) being such a method. The major disadvantage is that the accuracy of these models is often low due to the high possibility of important rules being omitted. On top of this, computation times are often slow compared to other classification algorithms. A recent research by Aoga et al. (2018) implements rule list based classification using the MDL principle in a similar way as the CLASSY algorithm. Their algorithm only works for binary classification though, and focuses more on describing the data set rather than the classification of it.

Sometimes rules can be extracted after an uninterpretable algorithm has been performed. Malioutov and Varshney (2013) propose a method to extract rules, but rule extraction is an inherently different type of rule based classification, so comparison is difficult.

Fuzzy rule based algorithms, like the FURIA model by Hühn and Hüllermeier (2009), are known for high accuracy and a distinct way of interpretability, but the fuzzy rules are put together in a rule set instead of a rule list. A rule set can be an accurate classifier, in some cases as accurate as the RIPPER algorithm (T. Wang 2018), but they often fall behind in terms of accuracy.

Another increasingly popular method aimed at increasing interpretability is making so called 'black-box' models interpretable in some way. Neural networks have provided high classification accuracy results due to the vast amount of calculations within each network, but left interpretability to be desired. Augasta and Kathirvalavakumar (2011) and Zhang, Wu, and Zhu (2018) propose different methods to make neural networks more interpretable, either through rule extraction or by making the neural networks themselves more interpretable.

Support Vector Machine (SVM) algorithms belong to the most used classification algorithms due to their high accuracy and low computation times. Originally SVM algorithms only worked for binary classification and were uninterpretable (Crammer and Singer 2001). Barakat and Bradley (2010) and Z. Wang and Xue (2014) provide solutions to both problems, adjusting SVM algorithms to be more interpretable and working for multiclass problems. However, no algorithm combining the two has been proposed yet, so using SVM algorithms for the multiclass case still is uninterpretable.

The CLASSY algorithm incorporates all the advantages of the algorithms mentioned, by using interpretable rule lists similar to RIPPER, but without the need for hyperparameters.

CLASSY also refines the use of the MDL based rule list, proposed in Aoga et al. (2018), by transforming it to work for classification instead of description.

3 Theoretical background

In this section, the theoretical background behind the CLASSY algorithm proposed by Proença and Leeuwen (2020) is discussed. The theory can be divided into three parts: rule list based multiclass classification, multiclass classification using the Minimum Description Length principle, and pattern mining. Combining these three into one algorithm gives the CLASSY algorithm.

3.1 Rule list based multiclass classification

The main objective of the algorithm is to perform multiclass classification. Multiclass classification can be performed using numerous methods, but here probabilistic rule lists are used. Rule list based classification can be described as finding a set of rules that can predict the class of items. A probabilistic rule list is an ordered list where each rule provides a probability distribution for the different classes that are affected by that rule. These lists always end with a default rule, which captures all the items that remained unaffected by every rule. An example of a rule list obtained by the CLASSY algorithm can be found in Table 1.
A rule is made up of two things: an antecedent and a consequent. The antecedent is a certain pattern for the available variables, and the consequent is a categorical distribution for the class labels. The pattern is a logical conjunction of variable values over all variables. For example, if there are three binary variables x_1, x_2 and x_3, a possible pattern could be $[x_1 = 1 \wedge x_3 = 1]$. In this case, this pattern only occurs iff $x_1 = x_3 = 1$, the value of x_2 does not influence the classification. Ultimately, in the probabilistic rule list, such as Table 1, these binary variables correspond to different variable values.

The categorical distribution consists of probabilities for each of the classes that are affected through the antecedent. For example, if the previous mentioned antecedent classifies the item into class A, y_A, or class B, y_B with probabilities 0.30 and 0.70 respectively, the categorical distribution is $(\mathrm{P}[y_A] = 0.30\ ,\ \mathrm{P}[y_B] = 0.70)$. Combining the antecedent and the consequent gives the following general structure for a rule: IF {antecedent} occurs THEN class $\sim$ {consequent}.

Combining all rules and the default rule and transforming the binary variables into variable specific equations, results in the complete probabilistic rule list as shown in Table 1. An item, also called an *instance* consisting of values for each of the available variables, is passed down through the rule list and is classified with the categorical distribution of the first rule that

Table 1: Probabilistic rule list of the *Wine* data set. P(Type x) refers to the probability of the observation being wine type x. Usage refers to the number of examples covered by a certain rule and class label.

Rule		**Antecedent**		**Consequent**	**Usage**
1	IF	[*OD280/OD315 <= 2.097* $\wedge$	THEN	P(*Type 1*) = 0.02	0
		Hue <= 0.853]		P(*Type 2*) = 0.04	1
				P(*Type 3*) = 0.93	41
2	ELSE IF	[*2.171 <Malic-acid <= 3.704* $\wedge$	THEN	P(*Type 1*) = 0.02	0
		Flavanoids <= 1.202]		P(*Type 2*) = 0.11	0
				P(*Type 3*) = 0.77	6
3	ELSE IF	[*688 <Proline* $\wedge$	THEN	P(*Type 1*) = 0.96	45
		13.371 <Alcohol]		P(*Type 2*) = 0.02	0
				P(*Type 3*) = 0.02	0
4	ELSE IF	*13.141 <Alcohol <= 13.295*	THEN	P(*Type 1*) = 0.78	6
				P(*Type 2*) = 0.11	0
				P(*Type 3*) = 0.11	0
5	ELSE IF	*688 <Proline*	THEN	P(*Type 1*) = 0.45	8
				P(*Type 2*) = 0.50	9
				P(*Type 3*) = 0.05	0
Ø	ELSE	ELSE	THEN	P(*Type 1*) = 0.02	0
				P(*Type 2*) = 0.95	61
				P(*Type 3*) = 0.03	1

matches with the variable values. In the case of more than one class being apparent in the categorical distribution, the class with the highest probability is chosen.

In practice, the antecedents and consequents are not given and need to be extracted and estimated from the data set. While the consequent can only be estimated after the antecedents are extracted, its estimation is discussed first as the extraction requires an in depth explanation. The consequents are estimated using a smoothed maximum likelihood estimator, where for each rule the specific probability for each class is estimated. This is done by dividing the number of times a pattern occurs and is classified by that rule with a specific class, by the total number of times a patterns occurs and is classified by that rule. The total number of times a pattern occurs and is classified by a rule is also called the *support* of that pattern. Both the numerator and the denominator are adjusted by adding error terms ϵ and $|\mathcal{Y}|\epsilon$, respectively, where $\mathcal{Y}$ is the set of all available classes.

3.2 Minimum Description Length principle

The extraction of the antecedents is an important process as the antecedents must provide the most accurate model possible. This process consists of two parts: first obtaining all possible antecedents and second finding out which combination of antecedents results in the most accurate and compact rule list. The rule list needs to be compact, because the more compact a rule list is, the more interpretable it is. To achieve this, the Minimum Description Length (MDL) principle is used. The general idea behind the MDL principle is that a model where the data is compressed the most is the best model (Grünwald and Grunwald 2007). For the construction of a rule list, this means the data and model need to be transformed in a way where the MDL principle can be applied.

For the model, the universal code for integers and the uniform code are used to encode the different model components. The universal code for integers is given by $L_{\mathbb{N}} = log(k_0) + log^*(i)$[1] where $log^*(i) = log(i) + loglog(i) + ...$ and $k_0 = 2.865064$. This universal code penalizes an increase in i, so it is used for components where a high number is not desired such as the number of rules or the length of a pattern. The uniform code is used when all elements are the same, as it encodes all elements with an equal length. This code often is in the form of - $\log \frac{\epsilon}{|\mathcal{Y}|\epsilon}$, where the numerator and denominator can take on different values depending on the elements that needs encoding.

The data is encoded using a different approach, namely using the prequential plug-in code. This code is used, because of its asymptotic optimality without the need for prior knowledge of the class probabilities. The prequential plug-in code also automatically provides the smoothed maximum likelihood estimators of the categorical distribution for each consequent mentioned in Section 3.1. The data consists of the so-called instances which contain values for the different variables and the corresponding class labels. The instances are treated as the input of the algorithm so only the class labels need encoding. Each label is given a uniform code at first which is increased by one every time it is counted. The prequential plug-in code for the classes is then formulated as the number of times a class has been present up until a certain point, divided by the total number of classes that have been present up until that point. Similarly as the smoothed maximum likelihood estimator, both the numerator and the denominator are adjusted with an error term ϵ. The prequential plug-in code is then used as a parameter to be used in the universal code for integers to be encoded similarly as the model encoding.

[1] To encode the lengths into bits, all logarithms are base 2

3.3 The algorithm

Obtaining all possible antecedents is done through pattern mining algorithms, in particular, frequent pattern mining algorithms. Frequent patterns are "itemsets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold" Han et al. (2007). Subsequently, frequent pattern mining is the process of extracting these frequent patterns. For the CLASSY algorithm the FP-growth method is used. This method is further explored in Section 4.1.

3.3.1 CLASSY

The final step is to combine the previous mentioned theoretical subjects to specify an approach which iteratively adds the best rule to the rule list from the set of all patterns. To do this, a heuristic algorithm based on the *separate-and-conquer strategy*, discussed by Fürnkranz, Gamberger, and Lavrač (2012), is used. This algorithm starts with a rule list containing only the default rule and iteratively adds the rule that gives the most compression in the data. After a rule is added, the part of the data that it compresses is removed and the new best rule is searched for. These steps are repeated until the data can no longer be compressed.

The mentioned compression is based on the normalized compression gain, which is the absolute gain divided by the number of times the pattern inside the rule occurs. The absolute gain is defined as the difference in length of the encoded model and data before and after the addition of a specific rule. Normalized gain is used in favor of absolute gain, because it has a higher support for rules that contain less instances but are more accurate for predictions.

A useful property for frequent pattern mining algorithm is the anti-monotone property of their support. The property says that if pattern `a` has less conditions than pattern `b`, pattern `a` will occur more than pattern `b`. This implies that the support of pattern `a` is bigger than or equal to the support of pattern `b`. This property makes mining frequent patterns efficient, but it also can be used to remove strictly redundant rules. Given all frequent patterns, if pattern `c` is a strict subset of pattern `d` and their support is the same, pattern `d` is called strictly redundant. It is redundant, because pattern `d` will always be encoded with a larger length and hence will never be preferred above pattern `c`.

Below the CLASSY algorithm is shown in Algorithm 1. For the sake of clarity, the specific notation used in Proença and Leeuwen (2020) is not introduced. Instead verbal explanations of the individual lines are given.

Algorithm 1: CLASSY

Input: Data set, list of candidates (set of frequent patterns)

Output: Multiclass probabilistic rule list

Remove strictly redundant rules;

Initialize the rule list, R, with only the default rule;

repeat

    Select the best rule to add to the rule list based on the normalized compression gain;

    Add the rule to the rule list R;

    Update the candidate list by removing the data affected by the rule and update the gain of the candidate list;

until *No rule improves the compression of the data*;

return *R*

4 Changing CLASSY

The previous section has discussed the three main theoretical subjects of the CLASSY algorithm. To improve the algorithm, these subjects need to be adjusted. The main idea behind the algorithm is to perform multiclass classification using MDL-based rule lists. So those two theoretical subjects will stay untouched, as altering them will lead to a completely different algorithm which is not the purpose of this research. So the third theoretical part, pattern mining, will be the focus for altering the algorithm. Currently, the FP-growth algorithm, proposed by Borgelt (2005), is being used within CLASSY. However there are multiple other competing frequent pattern mining algorithms. For this research, two different algorithms will be tested, namely the Apriori and ECLAT algorithms. Apriori was first introduced by Agrawal, Srikant, et al. (1994), and ECLAT by Zaki (2000). Both algorithms will be explained, as well as FP-growth to illustrate the difference between these algorithms and discuss the advantages and disadvantage of each algorithm.

4.1 FP-growth

Starting with the frequent pattern mining algorithm currently being used within CLASSY, FP-growth. The general idea behind the FP-growth method is that it constructs an FP tree from where the most frequent patterns are mined. The tree consists of nodes that correspond to the different items in the itemsets from the data. Here, an itemset is a possible pattern from the data set, which can be seen as a combination of variables with specific values. Note that an itemset does not necessarily have to contain all the available variables. An item corresponds to

one specific variable-value of the itemset.

The FP tree is constructed in multiple steps: first every item in all observations is counted and ordered in a descending order. Second, all items that have a support value lower than the threshold, and hence are infrequent, are removed from the list of items. Third, for each observation in the data set their frequent items are sorted in descending order. Fourth, the FP tree is constructed one observation at a time where the first node is a 'null' node, the second node is the most frequent item of that observation, the third node is the second most frequent item of that observation and so on. Each adjacent item in an observation is connected to form a path. If a new observation shares a prefix with an itemset that is already in the FP-tree, the new observation shares the already existing prefix in the tree, branching off where it differs, and the count of the nodes that it shares is increased by one. Each node that corresponds to the same item, but is on a different path is horizontally linked. This is important for the eventual mining of the frequent patterns.

From the constructed FP tree, frequent patterns can be mined by first traversing the different paths in the tree from the bottom to the top and examining the count of each of the linked nodes. These paths are called conditional pattern bases. If the count of an item meets the support threshold, they are used to construct a Conditional FP tree. Lastly, from this tree the frequent patterns can be mined directly.

4.2 Apriori

The Apriori algorithm is one of the first algorithms that was proposed for frequent pattern mining. The algorithm is based on the Apriori property, which states that all subsets of a frequent subset must be frequent. The Apriori algorithm uses this the other way around by using the fact that if a subset is infrequent, all of its supersets will also be infrequent.

The first step of the algorithm is the same as for FP-growth, which is counting all individual items in all observations. These items are called 1-itemsets, where the 1 corresponds to the number of items in the itemset. The items that do not satisfy the support threshold are left out for the next iteration. In the next iteration, the remaining items are joined together to form every possible combination. For example, 1-itemsets {a}, {b} and {c} satisfy the support threshold. For the next iteration, the 2-itemsets {ab}, {ac} and {bc} are counted and checked for the support threshold. The itemsets with a support count lower than the threshold are removed, and the 3-itemsets are formed by combining the remaining 2-itemsets with each other. This is continued until the k-itemsets has the size of the largest possible pattern and the most frequent itemset is obtained.

4.3 ECLAT

The Equivalence Class Clustering and bottom-up Lattice Traversal (ECLAT) algorithm is similar to the Apriori algorithm. Where the Apriori algorithm works in a horizontal way, scanning one observation after another, the ECLAT algorithm works vertically. Instead of counting all items mixed together, ECLAT first counts all the observations where the first variable-value occurs and puts those observations into an itemset. This is done for all variable-values, resulting in itemsets for every variable-values. Now, the same as for the Apriori algorithm, these 1-itemsets are checked for satisfaction of the support threshold and those who do not satisfy the threshold are removed. The remaining 1-itemsets are then combined into 2-itemsets like for the Apriori algorithm. This process continues until no more k-itemsets can be constructed, or no more k-itemsets satisfy the support threshold. At this point, the frequent patterns can be obtained.

4.4 Advantages and disadvantages

Each of the algorithms has its advantages and disadvantages. The Apriori and ECLAT algorithms are easy to understand and most of the time quite easy to implement. However, the computation times can differ substantially. For Apriori, the entire database needs to be scanned repeatedly, leading to an increase in memory usage and computation time. The ECLAT algorithm is very similar to Apriori, but since it searches vertically and mines frequent items differently, there is no need to scan the database multiple times as one time is enough. So the computation time of ECLAT is expected to be quicker than Apriori.

The FP-growth algorithm also only requires one scan of the database and there is no need for candidate generation as the patterns are directly mined from the tree. This would lead to an expected computation time faster than the other two algorithms. However, the implementation and execution of the FP-growth algorithm is more difficult.

Experiments will show if these expectations are empirically founded, and if there is a trade-off between computation time and implementation difficulty or if there is a significant difference in the eventual accuracy results for the multiclass classification.

5 Data preparation

This section discusses the several preparations that need to be made to the data sets. This includes normalization and discretization of the data sets in Section 5.1 , as well as the treatment of missing values in Section 5.2.

5.1 Normalization and Discretization

Before the data sets can be used in the algorithm, they first need to be discretized and normalized. Discretization can be described as the process of turning continuous data items into unique integer labels based on a number of sub-ranges where each item inside of a certain range gets the corresponding integer label. But a data set rarely consist of only continuous data items, often nominal data items are also present and normalization is necessary. Normalization is similar to discretization, as it is a process where nominal data items are turned into unique integer labels based on their value.

These processes are made easier by the LUCS-KDD DN software developed by Frans Coenen[2], who made a, Java implemented, user-friendly interface for this. One important aspect of the normalization and discretization is dividing the value range for a certain attribute into an appropriate number of sub-ranges. Each sub-range has its own corresponding number that is given to an attribute value if it is in the sub-range. For example, the values for attribute 1 range from 0 to 10. Dividing this range into five sub-ranges leads to the following sub-ranges: [0,2], [2-4], [4-6], [6-8] and [8-10]. Each sub-range gets a unique number: 11, 12, 13, 14 and 15 respectively. For a certain observation, if the value for attribute 1 is equal to 5.3, its new value becomes 13. A low number of sub-ranges is usually desired as it leads to fewer columns in the final data sets which in turn leads to quicker computation times. But choosing a value smaller than the number of classes can lead to less accurate results, unless the number of classes is relatively big (>5). Experiments by Frans Coenen have shown that a value of 5 provides a good trade-off between accuracy and computation time.

Another useful feature which comes before the normalization and discretization part is row randomization. Some data sets are ordered in some way, either based on class labels or based on some attribute. This can lead to a skewed result, so randomizing the rows is a useful precaution. After this, the data set is ready to be normalized and discretized.

However, the software provides a last option before saving the transformed data, distributing the classes. This feature tries to improve the accuracy by making sure that the distribution between the classes is even for the entire data set. The last thing that needs to be discussed is missing values and how to deal with them.

5.2 Missing values

A recurring problem for some data sets from the UCI Machine Learning Repository is that they contain many missing values. There are multiple options in dealing with missing values,

[2] https://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/lucs-kdd_DN.html

one better than the other. One of the more obvious solutions is deleting the observations that contain the missing values. However this comes with a great loss of information and is hence undesirable.

Another solution is replacing the missing value with a specific value. This value can be a constant for each attribute, or it could be computed based on other values in the missing value's column. Before these options are further explored, first a difference between two different stages of the data set where the missing values can be treated must be discussed. First is the raw data set taken directly from the UCI Repository, and second after the normalization and discretization have taken place. This distinction must be made, because the data sets have different properties and different aspects that need to be accounted for.

Starting with the first stage, the first thing to note is that the most common, and mostly used, way of noting missing values in data sets on the UCI Repository is by placing a '?' on the place of the missing value. This makes it easy to replace the missing value with another fitting value. However this value is restricted due to the way the normalization and discretization software is constructed. The software requires a schema file where every variable's properties are present. This also contains the possible values for each variable or a mention of the variable being continuous. This means the imputed values must follow these properties or the schema file needs to be adjusted.

The first is very difficult to do, as replacing the missing values in a way that they correspond to the properties could lead to biased and inaccurate results for non-continuous variables. The second is also not ideal, as replacing the missing values with estimated values means it is difficult to keep track what values are imputed, especially with many missing values. Hence adjusting the schema file would prove very difficult. On top of this, naming a non-continuous variable a continuous variable in the schema file to overcome this issue leads to the normalization and discretization software to be inaccurate.

The second stage is after the normalization and discretization has taken place, but here it is impossible to replace the missing values. The software does register the missing values, but deletes the single value that is missing. After the normalization and discretization has taken place, the exact location of the missing value in an observation is lost and the only thing that is left is a row with a smaller dimension than the other rows. From this point it seems impossible to impute the missing value.

Hence for the purpose of this research, data sets with missing values are not considered for the computational experiments.

6 Computational experiments

In this section the effect of changing the frequent pattern mining algorithm will be empirically tested based on three performance measures: accuracy, Area Under the Receiving Operating Characteristic Curve (Bradley 1997) and computation time. Accuracy states the correct number of classifications transformed into a percentage. The Area Under the Receiving Operating Characteristic Curve (AUC) is a measure to calculate the overall performance of the classifier. Originally, the AUC can only be used for binary classifiers, but using the weighted AUC it can also be applied for the multiclass case. The weighted AUC takes into consideration the specific class distribution of a data set and weights accordingly.

The computation time is the average over five algorithm repetitions, run on a 64-bit Windows operator, with Intel Core i7-6700HQ CPU at 2.60 GHz and 16.0 GB RAM.

6.1 Data sets

To test the effects of changing the frequent pattern mining algorithm in the CLASSY algorithm twenty distinct data sets are used. These twenty data sets are a mix of data sets also used for computational experiments in Proença and Leeuwen (2020) and new data sets. This mix is chosen to test if CLASSY performs differently compared to the original experiments. All of the data sets are from the UCI Machine Learning Repository[3]. Table 2 contains descriptive statistics of the twenty used data sets, where the new data sets are marked with an asterisk. The data sets vary in size, number of binary variables after normalization and discretization, and number of classes. This gives a broad insight in the effects and performance of the CLASSY algorithm.

6.2 Results

For each of the three different frequent pattern mining algorithms, the algorithm is run multiple times to obtain representative averages. For this experiment, the same parameter values for the minimum support and the maximum pattern length are used as in Proença and Leeuwen (2020). This means the minimum support is fixed at 5% and the maximum pattern length at 4. Table 3 contains the performance measure results for the three frequent pattern mining algorithms for all twenty data sets.

The first notable observation that can be made from Table 3 is that the accuracy and AUC results are exactly the same between the three frequent pattern mining algorithms. To research whether the algorithms mine exactly the same patterns, the total number of pattern candidates

[3] http://archive.ics.uci.edu/ml

Table 2: Descriptive statistics of the data sets used for the computational experiments. N is the number of observations in each data set. V is the set of binary variables. $\mathcal{Y}$ is the set of available classes. The data sets are ordered on the number of classes and on the number of observations after that. New data sets compared to the computational experiments in Proença and Leeuwen (2020) are marked with an asterisk.

Data set	N	$\|V\|$	$\|\mathcal{Y}\|$
Haberman*	306	9	2
Ionosphere	351	155	2
Pima	768	36	2
Tictactoe	958	27	2
Banknotes*	1372	18	2
Iris	150	16	3
Tae*	151	19	3
Wine	178	65	3
Seeds*	210	34	3
Wholesale*	440	29	3
Cmc*	1473	32	3
Waveform	5000	98	3
Pageblocs	5473	41	5
Nursery*	12960	27	5
Shuttle*	43500	43	7
Yeast*	1484	31	10
Led7	3200	14	10
Pendigits	10992	79	10
Chessbig	28056	40	18
Abalone*	4177	38	28

mined by each algorithm is given in Table 4. For *Haberman* and *Banknotes* the same number of patterns is mined, but for the other data sets the number candidates differ. So in general, the three algorithms mine a number of patterns that are unique to them and hence the final set of possible candidates is different.

For the *Haberman* and *Wholesale* data sets, the AUC is equal to 0.5, which means that the the algorithm is unable to discriminate between the prediction of different classes. This is explained due to the fact that the final rule lists for these data sets only contain the default rule, which makes prediction impossible.

The second observation from Table 3 is that the computation times are different for the three frequent pattern mining algorithms. For some data sets, like *Banknotes*, these differences are minimal, while for *Pendigits* the difference is almost two seconds. The second to last row contains the average computation time for each approach. The average computation time for the FP-growth algorithm is the lowest.

Table 3: Classification performance results (5-times repeated average for computation time). The last two rows contain the average values for the performance measures and the ranks of the computation times. New data sets compared to the computational experiments in Proença and Leeuwen (2020) are marked with an asterisk.

	Accuracy			AUC			Computation time (s)		
Data set	FP-growth	Apriori	ECLAT	FP-growth	Apriori	ECLAT	FP-growth	Apriori	ECLAT
Haberman*	0.7353	0.7353	0.7353	0.5000	0.5000	0.5000	0.90	0.92	0.92
Ionosphere	0.9373	0.9373	0.9373	0.9404	0.9404	0.9404	34.22	33.54	33.72
Pima	0.7474	0.7474	0.7474	0.7452	0.7452	0.7452	1.04	1.02	1.10
Tictactoe	0.9833	0.9833	0.9833	0.9759	0.9759	0.9759	1.16	1.20	1.20
Banknotes*	0.8695	0.8695	0.8695	0.9407	0.9407	0.9407	1.00	1.00	1.00
Iris	0.9600	0.9600	0.9600	0.9800	0.9800	0.9800	0.92	0.92	0.98
Tae*	0.4702	0.4702	0.4702	0.6215	0.6215	0.6215	0.90	0.90	1.02
Wine	0.9438	0.9438	0.9438	0.9887	0.9887	0.9887	1.54	1.52	1.54
Seeds*	0.9000	0.9000	0.9000	0.9568	0.9568	0.9568	0.94	0.94	0.90
Wholesale*	0.7818	0.7818	0.7818	0.5000	0.5000	0.5000	1.00	0.96	0.90
Cmc*	0.5431	0.5431	0.5431	0.7045	0.7045	0.7045	1.32	1.32	1.32
Waveform	0.7802	0.7802	0.7802	0.9245	0.9245	0.9245	61.78	62.84	62.34
Pageblocs	0.9281	0.9281	0.9281	0.7397	0.7397	0.7397	2.40	2.40	2.50
Nursery*	0.9361	0.9361	0.9361	0.9929	0.9929	0.9929	3.46	3.56	3.50
Shuttle*	0.9626	0.9626	0.9626	0.9874	0.9874	0.9874	13.66	13.60	13.70
Yeast*	0.5458	0.5458	0.5458	0.7976	0.7976	0.7976	1.70	1.78	1.80
Led7	0.7525	0.7525	0.7525	0.9497	0.9497	0.9497	1.70	1.70	1.72
Pendigits	0.9410	0.9410	0.9410	0.9913	0.9913	0.9913	121.24	123.20	123.13
Chessbig	0.5193	0.5193	0.5193	0.9089	0.9089	0.9089	15.92	16.2	16.26
Abalone*	0.2693	0.2693	0.2693	0.7324	0.7324	0.7324	3.92	3.90	3.96
Average	0.7753	0.7753	0.7753	0.8439	0.8439	0.8439	13.536	13.671	13.676
Rank							1.7	1.9	2.4

To test if these differences are significant, Friedman's test (Friedman 1937) is used. Friedman's test makes use of the ranking between the different approaches for each data set. For example, for the *Ionosphere* data set the ranks for FP-growth, Apriori and ECLAT would be 1, 2, 3 respectively. For equal values, the average is taken for the ranks that the two values would have covered. So for the *Haberman* data set the ranks would be 1, 2.5 and 2.5. These ranks are then used to calculate a test statistic which is used to test the null hypothesis that the three frequent pattern mining approaches perform equally well. The average rank for the three different approaches is given in the last row of Table 3. Looking at the average rank, the FP-growth algorithm performs slightly better than the other two.

The Friedman's test statistic for the computation times is equal to 7.156 with a corresponding p-value equal to 0.028.

To test for a specific difference that is present, a post-hoc analysis is used. The Wilcoxon test together with the Holm procedure is used, because the Wilcoxon test often provides more power as it employs more information (Pereira, Afonso, and Medeiros 2015). The null hypothesis for testing the difference between two frequent pattern mining algorithms, is that the two algorithms perform on par. Taking the FP-growth algorithm as the control group, the p-values for post-hoc analysis with Apriori and ECLAT are 0.391 and 0.073, respectively.

7 Discussion

This section provides a thorough discussion of the results mentioned in Section 6.2, starting with the first two performance measures, the accuracy and AUC results. It was noted that for each data set the results were exactly the same across the three frequent pattern mining algorithms. This could mean that the algorithms all mined the exact same set of candidates, but Table 4 shows that was not the case. However, the accuracy and AUC results being the exact same can be explained through the way the patterns are selected for the rule list. The most frequent patterns are chosen, which means that a pattern which is unique to one of the mining approaches has a high chance of being one of the more infrequent patterns. This means the choice of most frequent patterns comes from the sets of candidates that are overlapping between the algorithms. This leads to the accuracy and AUC results being the same.

For the data sets with an AUC equal to 0.5 (*Haberman* and *Wholesale*) further inspection of the data sets shows that many of the observations contain exactly the same value, leading to only one pattern being frequent and a distinction between different classes to be impossible. A possible explanation for this can be found in the way the data sets are normalized and discretized. For each attribute the value range is divided in at most 5 sub-ranges, where each

Table 4: Total number of possible frequent pattern candidates mined by each frequent pattern mining algorithm. New data sets compared to the computational experiments in Proença and Leeuwen (2020) are marked with an asterisk.

Data set	FP-growth	Apriori	ECLAT
Haberman*	7	7	7
Ionosphere	367489	367918	368127
Pima	535	535	538
Tictactoe	1844	1851	1855
Banknotes*	100	100	100
Iris	134	135	135
Ta *	338	332	340
Wine	15003	14989	14985
Seeds*	1045	1046	1047
Wholesale*	272	273	288
Cmc*	2289	2237	2285
Waveform	87207	87192	84811
Pageblocs	2874	2826	2908
Nursery*	2213	2206	2193
Shuttle*	4486	4388	4383
Yeast*	4815	4761	4844
Led7	2492	2496	2486
Pendigits	106087	106240	105982
Chessbig	1323	1303	1341
Abalone*	7812	7641	7780

sub-range has a different number which is given to the value of the observation if it is in that sub-range. If the range is very broad due to some outliers in the data set, many of the values will fall into the same sub-range, leading to the same patterns being present. Ultimately this leads to the inability to perform predictions and the AUC being equal to 0.5.

The third performance measure for the pattern mining algorithms is the computation time. Table 3 showed that these were different for the three pattern mining algorithms, and this difference was statistically tested using the Friedman's test. The Friedman's test statistic for the computation times is equal to 7.156 with a corresponding p-value equal to 0.028. So the null hypothesis of equal performance is rejected at a 5% significance level. Hence there is a significant evidence of a difference between the three frequent pattern mining algorithms.

The test statistics for the post-hoc analysis of the test, taking the FP-growth algorithm as the control group, are equal to 0.391 and 0.073 for Apriori and ECLAT, respectively. Hence in both cases the null hypothesis can not be rejected at a 5% level, but for ECLAT it can be rejected at a 7.5% significance level. So for Apriori, there is no significant evidence that

FP-growth and Apriori do not perform on par. However, with a slightly higher significance level there is significant evidence that FP-growth and ECLAT do not perform on par. So looking at the results in Table 3, FP-growth performs significantly better than ECLAT at a 7.5% significance level.

The insignificance of the tests on a 5% level can be explained due to the low computation times for most of the data sets where the differences between the frequent pattern mining algorithms are minimal. For data sets like *Ionosphere* or *Chessbig*, the computation time is higher leading to a bigger difference between the algorithms. This can be explained due to these data sets having many different attributes with a big range of possible values. This leads to a large number of binary variables after normalization and discretization and hence a bigger number of patterns that need to be checked for frequency.

When comparing the results of this research to those in Proença and Leeuwen (2020), provided in Appendix A.1, almost all accuracy and AUC results are higher in this research. This is remarkable as the results are obtained using the exact same version of the algorithm. A possible explanation could be that Proença and Leeuwen (2020) use 10 times repeated 10-fold crossvalidation, whereas this research does not. This means for this paper, the entire data set is used for training the algorithm instead of 90%, leading to a better trained algorithm and subsequently higher accuracy and AUC results.

8 Conclusion

This research gives an answer to the research question: '*Can the CLASSY algorithm be improved by using other frequent pattern mining approaches?*' The different frequent pattern mining approaches that were compared are the FP-growth (the current one), Apriori and ECLAT algorithms. The performance of each of the algorithms was tested based on accuracy, AUC and computation times. For the accuracy and AUC, the results were exactly the same across all algorithms. However, the computation times were different so Friedman's test and a post-hoc analysis were used to test for significance. The data sets were significantly different across all three, but the post-hoc analysis provided only evidence for FP-growth and ECLAT differing significantly. Hence using the ECLAT algorithm for pattern mining results in no significant improvement, but rather a significant degeneration of computation times. For the Apriori algorithm, there was no significant evidence that it did not perform on par with FP-growth. This means using Apriori does not result in a significant improvement of computation times.

Concluding, using other frequent pattern mining approaches does not result in significant improvements to the performance measures of the CLASSY algorithm.

8.1 Future research

This research only focused on different frequent pattern mining approaches as a way of improving the CLASSY algorithm. However, there are more methods to mine patterns, based on different principles. For future research fundamentally different pattern mining approaches could be tested, like the beam search algorithm.

The differences between computation times were more significantly present for data sets with a larger number of attributes. Future research could limit the use of data sets to those with a high number of attributes.

Lastly, the CLASSY algorithm contains more areas which can be improved, like the data and model encoding. Future research could be improving the CLASSY algorithm in those areas to make the algorithm more accurate or interpretable for multiclass classification.

References

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. Citeseer. 1994, pp. 487–499.

[2] John OR Aoga, Tias Guns, Siegfried Nijssen, and Pierre Schaus. "Finding probabilistic rule lists using the minimum description length principle". In: *International Conference on Discovery Science*. Springer. 2018, pp. 66–82.

[3] Shahrokh Asadi and Jamal Shahrabi. "RipMC: RIPPER for multiclass classification". In: *Neurocomputing* 191 (2016), pp. 19–33.

[4] M Gethsiyal Augasta and T Kathirvalavakumar. "A novel pruning algorithm for optimizing feedforward neural network of classification problems". In: *Neural processing letters* 34.3 (2011), p. 241.

[5] Nahla Barakat and Andrew P Bradley. "Rule extraction from support vector machines: a review". In: *Neurocomputing* 74.1-3 (2010), pp. 178–190.

[6] Christian Borgelt. "An Implementation of the FP-growth Algorithm". In: *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*. 2005, pp. 1–5.

[7] Andrew P Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[8] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[9] Wray Buntine. "Learning classification rules using Bayes". In: *Proceedings of the sixth international workshop on Machine learning*. Elsevier. 1989, pp. 94–98.

[10] Alberto Cano, Amelia Zafra, and Sebastián Ventura. "An interpretable classification rule mining algorithm". In: *Information Sciences* 240 (2013), pp. 1–20.

[11] William W Cohen. "Fast effective rule induction". In: *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[12] Koby Crammer and Yoram Singer. "On the algorithmic implementation of multiclass kernel-based vector machines". In: *Journal of machine learning research* 2.Dec (2001), pp. 265–292.

[13] Yashesh Dhebar and Kalyanmoy Deb. "Interpretable rule discovery through bilevel optimization of split-rules of nonlinear decision trees for classification problems". In: *IEEE Transactions on Cybernetics* (2020).

[14] Milton Friedman. "The use of ranks to avoid the assumption of normality implicit in the analysis of variance". In: *Journal of the american statistical association* 32.200 (1937), pp. 675–701.

[15] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. *Foundations of rule learning*. Springer Science & Business Media, 2012.

[16] Peter D Grünwald and Abhijit Grunwald. *The minimum description length principle*. MIT press, 2007.

[17] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. "Frequent pattern mining: current status and future directions". In: *Data mining and knowledge discovery* 15.1 (2007), pp. 55–86.

[18] David J Hand and Niall M Adams. "Data mining". In: *Wiley StatsRef: Statistics Reference Online* (2014), pp. 1–7.

[19] Jens Hühn and Eyke Hüllermeier. "FURIA: an algorithm for unordered fuzzy rule induction". In: *Data Mining and Knowledge Discovery* 19.3 (2009), pp. 293–319.

[20] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model". In: *Annals of Applied Statistics* 9.3 (2015), pp. 1350–1371.

[21] Dmitry Malioutov and Kush Varshney. "Exact rule learning via boolean compressed sensing". In: *International Conference on Machine Learning*. PMLR. 2013, pp. 765–773.

[22] Dulce G Pereira, Anabela Afonso, and Fátima Melo Medeiros. "Overview of Friedman's test and post-hoc analysis". In: *Communications in Statistics-Simulation and Computation* 44.10 (2015), pp. 2636–2653.

[23] Hugo M Proença and Matthijs van Leeuwen. "Interpretable multiclass classification by MDL-based rule lists". In: *Information Sciences* 512 (2020), pp. 1372–1393.

[24] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[25] Fadi Thabtah, Peter Cowling, and Yonghong Peng. "MCAR: multi-class classification based on association rule". In: *The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.* IEEE. 2005, p. 33.

[26] Tong Wang. "Multi-value rule sets for interpretable classification with feature-efficient representations". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 10858–10868.

[27] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. "A bayesian framework for learning rule sets for interpretable classification". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 2357–2393.

[28] Zhe Wang and Xiangyang Xue. "Multi-class support vector machine". In: *Support vector machines applications*. Springer, 2014, pp. 23–48.

[29] Yuhanis Yusof and Mohammed Hayel Refai. "MMCAR: Modified multi-class classification based on association rule". In: *2012 International Conference on Information Retrieval & Knowledge Management*. IEEE. 2012, pp. 6–11.

[30] Mohammed Javeed Zaki. "Scalable algorithms for association mining". In: *IEEE transactions on knowledge and data engineering* 12.3 (2000), pp. 372–390.

[31] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. "Interpretable convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8827–8836.

A Appendix

A.1 Comparison with original results

Table 5: Accuracy and AUC results for the original paper (Proença and Leeuwen 2020) and for this new research.

	Accuracy		AUC	
Data set	Original	New	Original	New
Ionosphere	0.89	0.9373	0.88	0.9404
Pima	0.73	0.7474	0.70	0.74523
Tictactoe	0.98	0.9833	0.98	0.9759
Iris	0.95	0.96	0.97	0.98
Wine	0.89	0.9438	0.95	0.9887
Waveform	0.75	0.7802	0.92	0.9246
Pageblocs	0.93	0.9282	0.74	0.7397
Led7	0.74	0.7525	0.94	0.9497
Pendigits	0.92	0.94102	0.99	0.9913
Chessbig	0.50	0.5193	0.90	0.9089

A.2 README.txt for code files

```
This file contains additional explanation on the different code files used
for this thesis.
Two code files were used: one for performing the multiclass classification
(CLASSY algorithm) and the other for Friedman's test and the Wilcoxon
post-hoc procedure.
Both code files are provided in a separate zip-file on SIN-Online.
That zip-file also contains the original README.txt file for clarification.

##############################################################################
1. CLASSY algorithm (mdl_rulelists.py)__

Most of the explanation for this code is in the code file and was written
by the authors. This can also be read for additional explanation, it can be
found in line 11-136.
```

In addition, most of the code has remained untouched as I only changed the pattern mining part, and wrote some additional code to run the algorithm with my own data sets.

||Changing the frequent pattern mining algorithm||

Changing the frequent pattern mining algorithm proved to be very easy, as there were built-in function from the PyFIM package.
The original authors also used this for FP-growth, so I only had to change the function where the patterns were being mined.
This is in line 405 (itemsets.extend([r[0] for r in fpgrowth(data_aux,supp=minsupp,zmin= 2,zmax=maxlhs)])), where 'fpgrowth' can be simply replaced with 'apriori' and 'eclat' to work for the different frequent pattern mining approaches.

||Making the code work for my data sets and other issues||

Importing the data files proved to be a problem, so lines 565-570 are dedicated to importing the data set via numpy and transforming it into a set (instead of a list or numpy array).

The original code required a binary data set (containing 1's and 0's) to be imported, as a parameter for the function 'binary2itemsets', where it is transformed into an itemset, together with two other variables: 'cl' and 'item2sets'.
My data sets were continuous or already in itemset format, hence this function was unusable or unnecessary.
This means I had to code the definition of 'cl' and 'item2sets'.
This is done in lines 572-580.

For binary classification, the function 'label_binarize' provided a vector instead of a matrix, which is why in lines 592-596 I wrote additional code for the binary classification where the correct 'y_aux' is calculated.

The if-statement is there to also work for multiclass classification,
where the original line of code is still present.

The last lines of code are there to provide some other insights
in the classification.

||Running the code||

To run the code, only two things need to be edited for each
different data set.
Line 564: Insert the name of the data set (not necessary for the
code to run, but makes it clearer.
Line 565: Insert the path to the data set, or if the code file is
in the same folder as the data set simply insert the name of the data set.

Changing the frequent pattern mining approach was discussed before,
but again it is simply changing 'fpgrowth' in line 405 to 'apriori' or 'eclat'.

##
__2. Friedman's test and the Wilcoxon post-hoc procedure
(friedmans_and_posthoc.py)__

The computation times for the three frequent pattern mining algorithms
are added manually.
Friedman's test has a built in function from scipy.stats which is utilized.
Wilcoxon post-hoc procedure with Holm adjusted p-values also has a
built-in function from scikit-learn which is utilized.