UNIVERSITY OF WESTMINSTER
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Feature Selection for Classification of Join Candidates in the Cairo Genizah

by
RIMA LAIDANI
W13098913

Supervised by
ELIA EL-DARZI

Submitted in partial fulfilment of the requirements of
the Dept of Business Information Systems
of the University of Westminster
for award of the Master of Science

October 2011

DECLARATION

I, Rima Laidani declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This has not previously been accepted as part of any other degree submission.

Signed : _____

Date : _____

FORM OF CONSENT

I, Rima Laidani hereby consent that this Project, submitted in partial fulfilment of the requirements for the award of the MSc degree, if successful, may be made available in paper or electronic format for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed : .....

Date : .....

Table Of Contents

Index of Tables

Abstract

The Cairo Genizah is a large collection of medieval fragments of manuscripts, discovered in the late 19th century. Most of the fragments were found in a poor state of conservation and rarely bound together as in their original state. Friedberg Genizah Project (FGP) engaged in digitalizing the complete corpus of the Genizah collection. The FGP's Computerization Unit is working on automatically finding manuscripts that once belonged to the same work by testing pairs of images based on information they extract from images. The term “join” designates a pair of images containing fragments from the same book. Newly discovered joins need to be validated by expert domains.

The goal of this project is to integrate the effort of the Friedberg Genizah Project (FGP) Computerization Unit in discovering "joins" with data mining techniques to extract useful knowledge from data. A dataset was derived consisting of the validated "joins" immersed in a multitude of "non joins", random pairs of images. Several feature selection methods are considered to: 1) weight the importance of attributes in discriminating a “join” from a “non join” and 2) to find useful subsets of attributes for a specific classification purpose. Several search techniques to traverse the space of feature subsets are considered, such as Best–First Forward Selection, Best–First Backward Elimination and Genetic Algorithm. Several supervised machine learning algorithms are employed and compared in terms of comprehensibility of results and estimated accuracy: decision tree C4.5, rule–learner RIPPER, ensemble classifiers obtained with bagging and boosting techniques.

Acknowledgements

I would like to thank Dr. Roni Schweka (Genzaim, The Friedberg Genizah Project Computerization Unit) for providing me with the data I used in this research and for his precious time; Prof. Elia El-Darzi for his precious advices and his supervision; Dr. Ben Outhwaite (Taylor-Schechter Genizah Research Unit, Cambridge) for sending me the images included in this work.

I would like to acknowledge the permission of the Syndics of Cambridge University Library for the reproduction of the images of manuscripts from the Taylor-Schechter and Mosseri Collections.

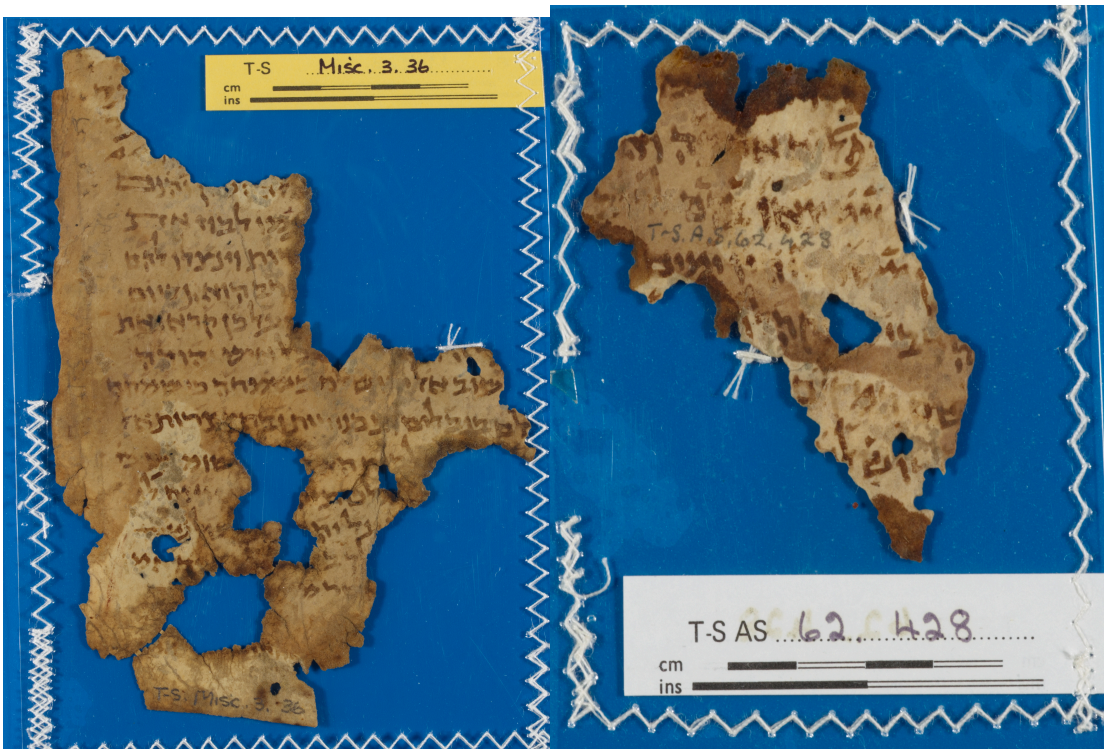
Chapter 1– Introduction

1.1 Application Domain – Genizah

The Cairo Genizah is a collection of approximately 350,000 medieval fragments of manuscripts that was discovered in the late 19th century in Ben Ezra Synagogue in Old Cairo. The material found have an immense impact in scholars involved with studies in “Bible, rabbinic, liturgy, history, philosophy”, and daily life in Mediterranean basin during the Middle Ages (Lior et *al.*, 2011). However, the collection is nowadays spread all around the world in about 75 libraries and private collections (Wolf, et. *al*., 2011). Moreover, the state of conservation can be extremely poor. For these reasons, it is scholarly extremely demanding grouping together fragments that once belonged to the same work.

The Friedberg Genizah Project (FGP) aims at advancing Genizah research by digitalizing the complete corpus of the Genizah collection. The availability of such an important collection of digital images lead the FGP's Computerization Unit and researches from the Blatvanik School of Computer Science (Tel Aviv) to develop an algorithm that would automatically test pairs of images to check whether they origin from the same book or not, *i.e.* whether they are "joins" or "non joins”. The algorithm decides whether a pair of images represents a “join” by computing several similarity scores, such as similarity in handwriting, content, script styles and physical measurements. Each discovered “join” needs to be validated by domain experts. The algorithm sometimes detects as “join” a pair of images that are not truly joins (False Positives), making the validation process too costly.

A “join” is either a set of two or more fragments belonging to the same manuscript (*codex-join*) or a set of fragments written by the same scribe but not necessarily belonging to the same manuscript (*scribe-join*). In what follows, unless otherwise specified, the term *join* will refer to *codex-join.*



Example of a discovered “join”, matching two very damaged fragments of manuscript (Passages from the Second Book of Chronicles, chapters 20 and 21)

1.2 Purpose of the Thesis

The goal of this project is to integrate the effort of the Friedberg Genizah Project (FGP) Computerization Unit in discovering "joins" with data mining techniques to extract useful knowledge from data.

The dataset used in this project was derived by randomly matching two images to produce a "non join" and by generating all possible combinations of pairs of images from the same book. The proportion "join"/"non join" was kept unbalanced 1:9, although in the reality the proportion is way more imbalanced (there are 61,249,825,000 possible matches between pairs of images).

The purpose of this thesis is twofold: to gain a better insight on how to define the *concept*

"join", and evaluate which attributes are more relevant/useful in discriminating a "join" from a "non join". This would help tuning efficiently the algorithm used to discover new joins. For this reason, I compare several feature selection methods and supervised learning algorithms.

Feature selection (FS) searches for a good subset of features that would maximizes the ability of the system to classify new instances (Bins, Draper, 2001) without loosing important information. FS methods fall into two broad categories: filter methods, wrapper methods (Kohavi, John,1997). The difference between the two approaches is that filters rank attributes or subsets of attributes independently of the classifier to be used afterwards, whereas the wrapper method uses the learning algorithm of interest to evaluate the usefulness of the candidate attributes subsets.

Filter methods usually produce a ranking of variable importance, based on general characteristics in the training set. They can be further classified as univariate and multivariate filters, depending on whether they assess features individually or taken in groups. There are several metrics that can be used to rank attributes or subsets. In this project, metrics used are based on Information Theory, chi-squared test and correlation measure.

Wrapper methods “perform a search over the space of all possible subsets of features, repeatedly calling the induction algorithm as a subroutine to evaluate various subsets of features” (Forman, G., 2003, pp 1291). The wrapper approach is called this way because the feature selection algorithm is “wrapped” around the classifier (Kohavi, John, 1997) of interest.

In this project, FS is applied to: 1) rank input variables individually, based on several metrics, 2) check whether there are redundant features, and 3) find whether there are input variables that are irrelevant by themselves but useful when taken in context with others (*feature interaction*).

Several classification algorithms were applied to: 1) evaluate the goodness of the features subsets returned by the different feature selection methods, 2) to see which technique gives a better insight on the mapping between input attributes and the class, and 2) to see which technique is more likely to be accurate in its prediction. The supervised learning algorithms used were rule–learner RIPPER, a decision tree C4.5 and ensemble classifiers, obtained by combining several versions of C4.5 via *bagging* and *boosting*.

In deciding which algorithm to include in this project, I was mainly driven by the need of getting comprehensible results. For example, in a pre-experimental phase I tried several rule–learners, such as PRISM (Cendrowska, 1987) and PART (Frank, Witten, 1998). Although these algorithms were more accurate compared to RIPPER, I decided to use RIPPER because it yielded more compact and expressive rules. For the same reason, although the ensemble classifier Random Forest (Breiman, 2001) performed better than bagging and boosting, It was not chosen because I could not view the model induced.

1.3 Thesis Overview

Chapter 2–Genizah. The chapter provides a brief overview of the Genizah collection.

Chapter 3–Feature Selection. Section 3.1 presents an overview of feature selection problem. Section 3.2 reports a literature review on the subject. Section 3.3 focuses on the two main categories filter and wrapper. Univariate filters are treated in subsection 3.3.1.1, whereas multivariate filters are treated in subsection 3.3.1.2. The final section 3.3.2 is about wrappers and search methods that can be used to traverse the space of candidate subsets of attributes.

Chapter 4–Classification. Section 4.1 presents an overview of the classification task in data mining. The subsequent sections treats the supervised learning algorithms included in this project: Section 4.2 is about decision trees, section 4.3 about rule-learners that employ a separate–and–conquer strategy, such as RIPPER algorithm, section 4.4 about the ensemble classifiers bagging and boosting.

Chapter 5–Experiment Design. Section 5.1 gives an overview of the dataset used in this project and how it was derived (Section 5.1.2). The remaining two sections are about the experimental design: Section 5.2 presents how filter and wrapper methods were applied, whereas section 5.3 reports how the different classifiers were used to evaluate the goodness of results from feature selection.

Chapter 6–Experiment Results. The chapter reports the results of feature selection methods and how they were evaluated using several classifiers. Section 6.1 presents results from filter

methods, more in detail, subsection 6.1.1 reports the ranking from the univariate filters whereas subsection 6.1.1.1 the evaluation of the results, subsection 6.1.2 reports results of running multivariate filter, while subsection 6.1.2.1 reports its evaluation. Section 6.2 reports results from running wrapper methods and subsection 6.2.1 its evaluation. Section 6.3 presents a summary of the results.

Chapter 7. The chapter gives a conclusion to this thesis.

Chapter 2 – Genizah

The term Cairo Genizah is currently used to describe a collection of approximately 350,000 fragments of manuscripts that were found in the storage room (genizah) of the Ben Ezra Synagogue in the neighborhood of Al-Fusṭāṭ (Old Cairo). According to the Jewish legal principle that prevents the burning of any written piece containing the name of God, the Jews of Old Cairo used to store worn pages of manuscripts in paper or vellum in the attic of the Ben Ezra Synagogue, without distinguishing between religious and secular material. The chronological span of the Genziah material lies between the 5th/6th century to the 19th century, when the storage room was discovered, opened, emptied, and the material it preserved was taken to various destinations. The larger collection of Genizah material (roughly 190,000 fragments) was brought to Cambridge University Library thanks to the efforts of clairvoyant scholars like Solomon Schechter and Charles Taylor, while smaller collections of fragments are now found in over 50 libraries all over the world. The majority of the manuscripts is datable from between the 10th-11th century to the half of the 13th century and is mainly written on paper and vellum. The languages represented in the fragments are Hebrew, Judaeo-Arabic (the form of middle Arabic written in Hebrew letters that was used by Jews in Islamic lands), Arabic, Aramaic, Ladino (Judaeo-Spanish), Judaeo-Persian, Yiddish, Coptic and Ethiopic. Their content is the most varied, spanning from religious texts (in particular fragments of Bibles, Talmudic treatises, commentaries and so on), to juridical and literary works and to very precious pieces of evidence (documents, contracts, private letters, notes) regarding the daily life in Medieval Cairo. When the Genizah fragments arrived to Western and Middle Eastern Libraries, their state of conservation was extremely poor: leaves were creased, covered in dirt and stuck together in bundles; pages once belonging to the same manuscript where scattered around and ended up in different libraries; in many cases, fragments once belonging to the same page of a work, a letter or a document, were then catalogued as single items regardless of their original context. More than half a century of conservation work lead to the actual state in which the fragments – at least the ones in Cambridge University Library - are safely preserved in melinex polyester sheeting and bound in volumes. A century of scholarly efforts lead to the almost complete description of the content of each single fragments, but it is still very hard to discover whether different fragments and leaves once belonged to a single larger manuscript.

The Genizah chamber in the Ben Ezra Synagogue (Al-Fustat, Cairo)

Chapter 3 – Feature Selection

3.1 Introduction

Data dimensionality reduction (DDR) is drawn from the fields of pattern recognition and statistics and aims at reducing the hypothesis search space, improving the accuracy of classifiers and providing a better understanding of the knowledge extracted (Wang, Fu, 2005). DDR can be classified in two general categories: *feature extraction* and *feature selection* (FS). Feature extraction generates artificial features from existing ones. An example of feature extraction is Principal Component Analysis (PCA). PCA produces a number of new attributes (principal components) that account for most of the variance in the existing variables. However, these new functions of the original features are not easy to interpret (Dash, Liu, 1997).

FS, on the other hand, looks for a subset of the existing features that would maximizes the ability of the system to classify new instances (Bins, Draper, 2001) without loosing important information.

Selecting a good subset of relevant attributes can improve not only the speed of the classifier but also it accuracy and the comprehensibility of results (Dash, Liu, 1997; Yu, Liu, 2003; Guyon, Elisseeff, 2003; Liu et al., 2002). Another important advantage of feature selection is that it allows a better insight on the process that produced data (Dash, Liu, 1997; Guyon, Elisseeff , 2003).

The basic idea in feature selection is to detect irrelevant and/or redundant features as they harm the learning algorithm performance (Moore, Lee, 1994). There is no unique definition of relevance, however it has to do with the "discriminating ability of a feature or a subset to distinguish the different class labels" (Dash, Liu, 1997, p. 135). However, as pointed out in (Guyon, Elisseeff, 2003), an irrelevant variable may be useful when taken with others and even two irrelevant variables that are useless by themselves can be useful when taken together.

FS methods fall into three broad categories: *filter* methods, *wrapper* methods (Kohavi, John, 1997), and *embedded methods*. Filter methods are independent of the inductive algorithm to

be used afterwards and they rely on general characteristics of the training data. Wrappers use the classifier to be used afterwards as a black box to evaluate the usefulness of subsets of features. Embedded methods are particular machine learning algorithms that perform variable selection implicitly during the training phase, for example decision trees and rule–learner algorithms such as RIPPER.

The problem of finding an optimal subset of features is computationally infeasible in most real-world problems (Vafaie, De Jong, 1993; Dash, Liu, 1997; Kohavi, John, 1997). In fact, for a dataset consisting of n features, the order of the search space is O(2^n). This makes an exhaustive search too costly and impractical. For this reason, a heuristic search or random search is used to traverse the space of competing features subsets. In the heuristic search, the order of the search space is usually O(n^2) or less, whether in random search, although the search space is O(2^n), fewer subsets are evaluated (Dash, Liu, 1997).

Areas in which feature selection play an important role are text classification, gene expression array analysis, and combinatorial chemistry (Guyon, Elisseeff, 2003; Forman, 2003).

3.2 Literature Review

Kira and Rendell (1992) introduced the method Relief. Relief is based on a statistical. It weights features drawing its inspiration by the instance-based learning algorithms. The algorithm first requires the user to set a number of instances to be randomly drawn from the training set. Then it assigns a weight, initialized to zero at the beginning, to all attributes. For each instance in the sample, say x, it looks for its Near Hit, *i.e.* the instance closest to it (based on the Euclidean distance) and that has its same class and the Near Miss, *i.e.* the instance closest to it that has a different class value. An attribute becomes more relevant, and its weight is increased, if it assumes different values for x and the Near Miss, and is made less relevant if its value is different for x and Near Hit. This process is repeated for all instances in the sample. Finally, the algorithm selects a subset of attributes having weights greater than a specified threshold.

Relief algorithm is robust against noise, it requires O(*n*) to find a good subset, where *n* is the sample size, it deals with both numerical and categorical attributes. However, it fails in detecting redundant features and it works only with binary classes.

Almuallim and Dietterich (1992) introduced a new algorithm: Focus. Focus algorithm performs

an exhaustive search through the space, looking for the minimal subset of features that is sufficient to classify all instances in the training data. The criterion used to evaluate a subset is the *consistency*. The disadvantages of this method are its sensitivity to noise and the exponential explosion of the search space when there are more than 25-30 features (Koller, Sahami, 1996).

Vafai and De Jong (1992, 1993) investigated the importance of feature selection to improve the performance of rule induction algorithms in the domain of image processing and proposed using genetic algorithm to explore the space of all subsets of features.

Cardi (1993) proposed a FS method based on the use of a decision tree (decision tree method DTM). The method consists in running a decision tree over the training data and in retaining only the attributes that appear in the pruned tree. This approach aimed at improving the performance of nearest-neighbor algorithm.

John, Kohavi and Pfleger (1994) provided a formal definition of relevant/irrelevant features. More importantly, they proposed that the selection of a good subset of features should depend on the induction algorithm to be used afterwards.

Langley (1994) presented an overview of feature selection methods.

Moore and Le**e** (1994) introduced the schemata search, "a new method for quickly finding families of relevant features".

Koller and Sahami (1996) addressed both theoretical and practical aspects of feature selection. They proposed a filter based on Information Theory. They proved experimentally that their method was efficient in detecting irrelevant and redundant attributes, it dealt well with noise and, being a filter method, it did not incur in the computational cost typical of a wrapper method.

Dash and Liu (1997) gave an overview of many existing feature selection methods from the 1970s to that moment. The authors identified four steps in a typical feature selection method, *i.e*. generation, evaluation, stopping criteria and validation. They classified the evaluation functions in five categories: distance, information (or uncertainty), dependence, consistency,

and classifier error rate. Moreover, they group the generation procedures into three groups: complete, heuristic, and validation approaches.

Blum and Langley (1997) concentrated on both selecting relevant attributes and relevant examples.

Kohavi and John (1997) introduced the wrapper method. In the wrapper approach, the feature selection algorithm is a wrapper around the induction algorithm. The algorithm searches for a good subset using the estimated accuracy of the classifier itself as evaluation function. In fact, according to the authors, FS should consider how that specific classifier and that particular training set (domain) interact. The accuracy of the classifier was evaluated using a 5–fold cross–validation.

Das (2001) investigated the advantages and limitations of filter and wrapper methods for feature selection and proposed a new hybrid algorithm, which they claimed being competitive with the wrapper methods but much faster.

Xing et al. (2001) proposed a hybrid method to solve the problem of feature selection in the field of molecular biology.

Forman (2003) presented an empirical comparison of twelve feature selection methods in the context of text classification. The author introduced a new metric for variable ranking: Bi-Normal Separation.

Guyon and Elisseeff (2003) presented an excellent introduction to variable selection.

3.3 Filter–Wrapper

In this project, I will focus on filters and wrappers methods.

Filters rank attributes or subsets of attribute without invoking any classifier. Wrappers need to call the same classifier as many times as the number of candidates it evaluates. For this reason, filter methods are much faster and they yield a more general result than wrapper methods. They are simple and scalable (Guyon, Elisseeff, 2003). However, wrapper methods return a subset that is particularly appropriate for that classifier and that training set, resulting

in a better performance.

3.3.1 Filter

Filter methods can be divided into two subcategories: univariate methods and multivariate methods, depending on whether they assess features individually or in subsets. Univariate and multivariate filters are also known as weighting algorithms and subset search algorithms (Yu, Liu, 2003).

Several metrics can be used to assess the worthiness of a single features. Some popular metrics are entropy (or Information Gain), Chi–Squared test, Gain Ratio. Univariate filter can help detecting irrelevant features but can do nothing against redundant features. Redundant features adversely affect the speed and the accuracy of learning algorithms. To overcome the limitations of univariate filters, Hall (1999) proposed a multivariate filter (CFS) that would detect not only irrelevant features but also redundant ones. The metric used by CFS is correlation.

3.3.1.1 Univariate Filter

Here are presented two univariate filters used in the project. One is based on Information Gain metric and the other on the Chi–Squared test.

3.3.1.1.1 Information–based Filter

This kind of filters are based on the information–theoretical concept of *entropy,* a measure of the uncertainty of a random variable (Yu, Liu, 2003). In general, the entropy of a random variable X is defined as

$$H(X) = -\sum_i P(x_i)\log_2\big(P(x_i)\big)$$

and the entropy of X after observing values of another random variable Y is defined as

$$H(X|Y) = -\sum_j (P(y\_i))\ \sum_i (P(x_i|y_i)\log_2(P(x\_i|y\_i))$$

where $P(x_i)$ is the prior probabilities for all values of X, and $P(x_i|y_i)$ is the posterior probabilities of X given the values of Y (Yu, Liu, 2003).

The quantity by which the entropy of X decreases thanks to the additional information about X

provided by Y is called the *information gain* or *entropy* and is given by

$$IG(X|Y) = H(X) - H(X|Y).$$

This type of filter evaluate the information gain from a feature, *i.e.* the difference between the prior uncertainty and the expected posterior uncertainty about the class using that feature. It "measures the decrease in entropy when the feature is given versus absent" (Forman, 2003).

3.3.1.1.2 Chi-squared Filter

The chi–squared method assesses features individually by computing their chi-squared statistic with respect to the class. This test is used to test whether there is a enough statistical evidence at a given significance level (say 0.05) to consider two categorical variables independent from each other. The method requires numerically-valued attributes to be discretized.

This approach ranks the χ^2values with the largest one on top, as the larger χ^2 value, the more important the feature is (Liu et al, 2002). Chi–squared test is a test that evaluates the distance from the expected distribution of a feature if that feature is assumed independent from the class (Forman, 2003).

What follows is a general overview of the chi–squared test.

In general, to compute the test between a variable A with r levels and a variable B having c levels, we need to 1) state the hypotheses, 2) specify the significance level, 3) analyze data, 4) interpret results.

The null hypothesis H_0 states that the two variables A and B are independent, whereas the alternative hypothesis H_a states that the two variables are not independent.

H_0: Variable A and Variable B are independent.

H_a: Variable A and Variable B are not independent.

Usually a significance level of 0.05 or 0.01 is used.

Analyzing the data involves computing the degree of freedom, the expected frequencies, the test statistic χ^2, and the P–value associated with χ^2 statistic.

Degrees of freedom is computed as $(r-1)(c-1)$.

Expected frequencies need to be computed for each level of one variable at each level of the

other variable. The expected frequency for level x of variable A and level y for variable B is given by the product between the number of observations at level x of variable A and the number of observations at level y of variable B, divided by the total number of observations.

Once $r \times c$ expected frequencies have been computed, the test statistic χ^2 is computed as follows:

- For each level x of variable A at level y of variable Y, compute the difference between the observed and expected count for this combination of levels, square it, and divide it by the expected frequency.

- Compute the sum of the values obtained in the previous step.

- Compute the P–value associated to the χ^2 test with that degree of freedom.

If the P-value is less than the significance level chosen, then there is enough statistical evidence to reject the null hypothesis, otherwise, we cannot reject it.

3.3.1.2 Multivariate Filter – Correlation–Based Filter (CFS)

CFS is based on correlation measures or dependence measures. A correlation measure evaluates the ability to predict the value of one variable knowing the value of another variable (Dash, Liu, 1997). The method evaluates the worth of subset of attributes based on the criterion that a good subset of attributes contains features "highly correlated with the class, yet uncorrelated with each other" (Hall, 1999). A high inter–correlation between input variables indicates redundancy and redundancy harms both the learning time and the accuracy of classifiers. Thus, it may happen that a highly relevant feature is not included in the subset because it is redundant (Guyon, Elisseeff, 2003; Kohavi, John, 1997).

The method computes a matrix of feature–class correlations and feature–feature correlations. For each subset of attributes *S* being evaluated and containing *k* features, a merit metric is computed as

$$MeritS = k\frac{rcf}{\sqrt{(k + k(k-1)rff)}}$$

,

where *rcf* is the average feature–class correlation, and *rff* is the average feature-feature inter–correlation (Hall, 1999). Thus, the higher the average correlation feature–class and the lower the average correlation feature–feature, the higher the merit of the subset.

3.3.2 Wrapper

Wrapper methods evaluate subset of attributes based on their usefulness to a given classifier. Wrappers are conceptually very simple. To use this feature selection technique, one needs to decide: 1) how to search the space of all possible subsets of variables and how to halt it, 2) how to estimate the accuracy of the classifier used called by the wrapper, and 3) which classifier to use as a black box (Guyon, Elisseeff, 2003). The accuracy of the classifier used as a black box is usually estimated using the hold-out method or cross-validation.

This section reviews some possible searches that can be used to traverse the space of feature subsets.

3.3.2.1 Search methods

An exhaustive search is computationally infeasible even for datasets with a small number of attributes. In fact, this would mean running 2^n times the classification algorithm on the dataset, where *n* is the number of attributes.

3.3.2.1.1 Greedy Search

The simplest search technique is the hill–climbing (greedy) forward selection or backward elimination. When the forward selection is used, attributes are progressively included into larger and larger subsets, whereas when backward elimination is used, one starts with the entire set of variables and progressively removes the least promising one. When a greedy search is applied, one never revisits former decisions in the light of new findings (Guyon, Elisseeff, 2003) and the search stops whenever adding or removing a feature decreases the estimated accuracy of the classifier. Thus, the search stops in a local optimum.

The advantage of starting from an empty set of attributes is that it is less computationally expensive. In fact, running the classifier on dataset with few attributes is much faster than running it starting from the entire set of attributes. However, starting from the entire set of attributes and trying to remove one at a time allows to capture higher order (more than two) interacting features, *i.e.* features that might be irrelevant when taken alone but that become useful when taken together. Forward selection is to be preferred if one suspects that only few variables are relevant and backward elimination is to be preferred if one suspects that only few variables are expected to be irrelevant (Moore, Lee, 1994).

Alternative searches are best–first, branch–and–bound, simulated annealing, genetic

algorithms, schemata–search, beam–search, race, etc.

In my project I focused on best–first search and genetic algorithm.

3.3.2.1.2 Best–First Search

Best–first search selects the most promising node encountered so far that has not been expanded (Kohavi, John,1997). This search doesn't terminate when the performance starts to drop but keeps a list of all attribute subsets evaluated so far, sorted in order of the performance measure, so it can revisit an earlier configuration. Although best–first search is less prone to remain stuck in local optima than greedy forward selection, it is not clear whether it is better for feature selection (Kohavi, John,1997).

3.3.2.1.3 Genetic Search

Genetic algorithms are optimization techniques that mimic the evolutionary law of "survival of the fittest".

When applying genetic algorithm to the search of an optimal feature subset, a single subset can be viewed as an individual within the population. Subsets of features can be represented as binary strings (Moore, Lee, 1994): if the *ith* variable is included in the subset, then the position *i* in the string will assume value 1, otherwise 0. For example (1,1,0,0) denotes a subset in which only the first two variables are included.

Individuals are evaluated by using a fitness function. Pairs of high performing individuals are selected for mating and the resulting pairs of offspring (partially or completely) replace the old generation. The idea is to produce new solutions that retain many of the good features of their parents and eventually perform better. The population is maintained of approximately the same size (Vafaie, De Jong, 1993).

The offspring are produces using two main genetic operators: crossover and mutation:

- Crossover randomly selects a point within the strings representing the parents and swaps all the bits after that point between the two.
- Mutation randomly changes one or more bit of an individual to introduce perturbation in the population. Perturbation allows to explore new areas in the search space and help avoiding remaining trapped in a local optimum. Thus mutation prevents search stagnation.

One need to specify a crossover rate, *i.e.* the probability that two individuals will swap part of

their encoding and the mutation rate, *i.e.,* the chance that a bit or more within an individual will be flipped.

One need to tune the parameters:

- crossover rate, value close to 1 allows a large number of offspring (the choice is not critical) and,
- the mutation rate, which is very critical. An optimal choice of the mutation rate will allow the search to explore intensively the good regions, avoiding remaining trapped in local optima. When it is set larger than 0.25 it will be difficult to explore adequately the good regions, and the search will perform more like a random search; when it is set too small, the search may not be enough randomized to escape local optima. A reasonable value is 0.01.

Chapter 4 – Classification

4.1 Introduction

Predictive models can be grouped in two major groups: *classification* and *regression* models. Both aim at building models that predicts the value of a variable knowing the values of other variables. Both models accept in input a set of training data. Each training instance has several attributes, one of which is the variable to be predicted. In classification, this variable is categorical and it is called *class* variable, in regression the variable is real–valued and is known as *dependent variable*. The remaining attributes are known as *features*, *attributes*, *input variables*, *predictors*, *explanatory variables*, etc. Predictive models learn, using the training data at hand, a mapping from the input variables to the dependent variable (Hand et *al.*, 2001). The resulting model is then used to predict the value of the dependent variable for a new instance of which all the independent variables are known.

Classification has been successfully applied to many areas, such as “scientific experiments, medical diagnosis, fraud detection, credit approval, target marketing” (Nong, 2003), computer and network system security (Pietraszeka, 2005), etc.

4.2 Decision Trees

Decision Trees are a popular family of supervised learning algorithms. They origin from the field of decision theory and statistics (Rokach, Maimon, 2005).

Decision trees are directed graphs with a root, internal nodes, branches and leaves (also known as *terminal nodes* or *decision nodes*). All internal and terminal nodes have exactly one incoming branch. The root and the internal nodes have two or more branches leading to their child nodes.

The process of building a tree model from the training set is known as *tree induction* or *tree growing*. The most commonly used approach is the greedy top–down method. The basic idea is to recursively “test on attributes to partition the training data into smaller and smaller subsets until each subset contains instances that belong to a single class” (Quinlan, 1990).

A region of the instance space is associated to each node. The general algorithm starts with

the entire training set and an empty model. It selects a “best” attribute and generates a node for it. The algorithm performs a test on the attribute’s values and, based on the outcome of this test, it partitions the instances at that node in two or more subspaces that are associated to newly created child nodes. This process iterates recursively at each node. The tree induction stops when all instances in a node belong to the same class or if it is not worth to continue partitioning the training data further. Each leaf node has associated a class label, which is the (majority) class of the instances that are associated to that node.

The choice of the best attribute at each node is mainly based on the class distribution of the records before and after the test (Pang-Ning, et *al.*, 2005). Most of the measures used are based on the difference between the degree of impurity at the parent node and the weighted sum of the degrees of impurity at the child nodes after splitting. The weights are given by the relative proportion of instances at the child nodes. The bigger the difference, known also as *gain*, the better the split.

One common measure of impurity at node *t* is the *entropy*, defined as:

$$Entropy\ (t) = -\sum_{i=1}^{c} p(i|t)\log_2(p(i|t))$$

where $p(i|t)$ is the proportion of instances at node *t* that belong to the class *i* (i=1,..,c).

Other impurity measures are Gini Index and Classification error (Pang-Ning, et *al.*, 2005).

When the measure of impurity is entropy, gain is also known as *information gain.*

To classify a new instance, this is propagated down the tree and it is labelled accordingly to the class label in the leaf it reaches.

Many aspects relative to building an optimal decision tree were proved to be intractable (Murthy, 1998). For example, Hyafil and Rivest (1976) proved that the problem of finding an optimal binary tree, where optimal refers to a tree with the minimum number of tests to classify an object, was NP–complete. Tu and Chung (1992) proved formally that the problem of finding the optimal decision tree with the smallest size is NP–complete. Thus, all existing algorithms accept a suboptimal decision tree.

It is important to address the issue of overfitting, a situation that arises when one uses an overly complex model that works perfectly on the training set and that performs poorly on new and unseen set. There are two general approaches to tackle this issue: prepruning and

postpruning. Prepruning stops growing a tree as soon as a stopping criteria is met. Typical stopping criteria are: 1) The maximum tree depth has been reached, 2) the minimum number of instances in a node is reached, 3) the best splitting criteria is below an acceptable threshold. However, it is not easy to tune efficiently these early stopping criteria. For this reason, most of learning algorithms perform a postpruning, *i.e.* they induce a complex model and then they simplify it to the point where it is more likely that the tree models patterns in the underlying data rather than random noise in the training set.

Rokach and Maimon (2005) delineated the advantages and disadvantages of decision trees in their survey on decision trees algorithms. The advantages are: 1) decision trees are easy to understand ("self–explanatory"), 2) they accept both numerical and categorical input variables, 3) they are nonparametric, they do not make any assumption about the data and the class distribution, 4) they are robust against noise, and 5) they can handle missing values. The disadvantages are: 1) most of the decision tree algorithm require a categorical class, 2) decision trees are unstable algorithms, *i.e.* small perturbation in the training set can lead to a completely different model, 3) because of the recursive partitioning of the training data, it may happen that at the lower levels of the tree the number of instances is too little to make statistically relevant assumptions about the class (*data fragmentation)*, and 4) the *replicated subtree problem*. This problem is due to the fact that decision trees cannot express in a compact way disjunctions, as they test one attribute at a time. The result is that they may be way more complex than necessary. Please refer to Appendix B for an example.

There are several decision trees algorithms, such as CHAID (Kass, 1980), CART (Breiman et *al.*, 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993).

In my project I focus on C4.5 (Quinlan, 1993).

Quinlan's C4.5 (1993) is an evolution of his ID3 (1986) algorithm. Both algorithms "have served as the primary decision trees" in the machine learning community (Neville, 1999).

C4.5 performs multi–way splits on input variables. The metric used to select the best attribute is the *gain ratio*, given by the reduction in entropy normalized by the entropy of the split. The normalization is done to overcome the fact that information gain tends to favor attributes with many distinct values. The algorithm handles missing values and numeric attributes. C4.5 performs postpruning based on two operators: *subtree replacement* and *subtree raising* (Witten, Frank, 2005). In the first case, a node is substituted with a leaf, in the second case

with another subtree below it. To decide whether to prune a node or not, the algorithm compares the error that would be expected on independent data when the node is left and when the node is substituted. A typical way to estimate these errors would be to retain some training instances for validation purpose. However, this would cause building a tree based on fewer data. C4.5 uses the training set itself to estimate this error and uses a *pessimistic estimate* of the error by taking the upper confidence limit rather than considering the confidence range.

4.3 Rule–Learners

Rule–learners induce a set of rules from a training data. Training data consists of labeled data, *i.e.* "positive" and "negative" examples of a target concept (class). In this project, positive examples are “joins”, negative ones are “non–joins”. Rules are chosen such that they will discriminate well on unseen data. These learners try to present the knowledge gained from the data in a way that facilitates both interpretation and taking decision (Wang, Fu, 2005).

In this project I will focus on the family of rule–learners algorithms that adopt the *separate–and–conquer strategy*, also known as *covering* algorithms (Pagallo, Haussler, 1990). In particular, I will focus on the RIPPER algorithm (Cohen, 1995). From now on I will use the terms covering algorithm and separate–and–conquer algorithm interchangeably.

The general separate–and–conquer algorithm induces one rule at a time. Each rule covers a portion of the training data. It is called separate–and–conquer algorithm because training examples that were covered by the last induced rule are removed (separated) from the training set before a new rule is induced to cover (conquer) new training data. The goal is to describe almost all of the positive examples in the training data and almost none of the negative examples. The resulting rule set describes the class in terms of rules that perform tests on the input attributes.

An alternative to the induction of rules directly from the training data is to derive rules from decision trees, *i.e.* associate a rule to each leaf. The rule obtained that way would consist of all the tests along the path from the root till that node. However, rules derived straightforward

from a decision tree are far too complex and hard to understand (Quinlan, 1987; Pagallo, Haussler, 1990; Furnkranz, 1996). This is due to the decision tree cannot express succinctly the concept of disjunctions. This problem is known as *replicated subtree* (Pagallo, Haussler, 1990). Please refer to Appendix B for an example.

A basic form of the separate–and–conquer algorithm is given by the PRISM algorithm (Cendtowska, 1987). It starts with an empty set of rules and iteratively adds rules until all positive examples are covered. The way a single rule is induced is the following: conditions (tests on particular attributes) are greedily added to its body in the effort to include as many positive examples as possible while excluding as many negative examples as possible until no negative example is covered by the rule. The ruleset induced in this way covers all positive examples (*completeness*) and no negative example (*consistency*) (Furnkranz, 1996).

Various separate-and-conquer algorithms differ in the way a single rule is learnt.

Although they share the basic idea of PRISM, many covering algorithms avoid to induce a *complete* and *consistent* ruleset, that would overfit data and perform poorly on new and unseen instances. The goal is then to induce a ruleset that covers as many positive examples as possible while covering as few negative examples as possible. There are two general approaches to address the problem of overfitting: prepruning and postpruning. Prepruning deals with noise during the learning phase, while postpruning addresses this problem after inducing a complex ruleset (Furnkranz, 1997). In the context of separate–and–conquer algorithms, prepruning strategy favors the induction of simple rules with wide coverage rather than complex rules with low coverage; on the other hand, postpruning allows learning complete and consistent rules and then simplifies the complex ruleset trying to discard rules or conditions that are actually modeling noise in the training data rather than trends in the domain (Furnkranz, 1997). The training set is usually split in a *growing* set and a *pruning* set. Typically, the worthiness of a rule or condition is evaluated on the pruning set, which was not used during the induction phase. This general approach is known as REP, reduced error pruning. As for other learning algorithm, prepruning applied to rule induction may prevent from discovering good conditions to add to rules, thus postpruning is usually preferred (Witten, Frank, 2005).

However, postpruning strategies are inefficient in that they spend time learning a rule set that is then simplified (Furnkranz, 1997).

4.3.1 Background RIPPER

Furnkfranz and Widmer (1994) introduced IREP algorithm. IREP stands for *incremental reduced error pruning,* and was introduced to overcome some limits of the REP approach. As previously mentioned, REP approach splits the training data into a growing set, which is approximately two–third of the original data, and a pruning set. It generates a ruleset that overfits the growing set and then tries to simplify it by repeatedly discarding the last condition within a rule or a rule itself. The next condition or rule to be pruned depends on the reduction of error as estimated on the pruning set. A disadvantage of this approach is that the induction does not consider instances in the pruning set and some important rules may be pruned because the pruning set is not enough representative. IREP splits the training set into two partitions right after a rule is induced. IREP was proved experimentally to be as good as REP in terms of accuracy but much faster. The major difference between IREP and REP is that IREP does not wait till a complex ruleset is induced before pruning it back but it simplifies a single rule as soon as it is induced.

Cohen (1995) proposed RIPPER (for *repeated incremental pruning to produce error reduction*) algorithm by performing some changes to the algorithm IREP: 1) he changed the metric used to guide the pruning phase, 2) he added a new criterion to stop adding rules to the ruleset, and 3) he introduced a post-process phase to optimize the ruleset induced. His experiments showed that RIPPER was competitive with C4.5 rules in terms of accuracy and performed better on noisy dataset. Moreover, he showed that RIPPER was faster than C4.5 rules.

Furnkranz (1996) presented an overview of rule learning algorithms that use a separate–and–conquer strategy.

4.3.2 RIPPER

RIPPER repeats the following steps for each class at a time, moving from the last frequent one to the most frequent.

RIPPER splits the training set into a growing set and a pruning set. It grows a rule by greedily adding conditions that maximizes the information gain and stops as soon as a rule covers negative examples. Whenever the rule is induced, it is immediately pruned back considering

the deletion of conditions in a last–to–first order. The next condition to be deleted depends on how much the estimated error on the pruning set increases. All covered instances by the rule are removed from the training set and this is randomly split into a growing and pruning sets. The process continues adding rules to the ruleset until there are no uncovered instances or the last rule induced has an estimated error above 50% or the description length of ruleset and examples is 64 bits greater than the smallest description length found so far.

After the ruleset was induced, RIPPER starts a complex optimization stage that aims at that rules work well together (Frank, Witten, 1998).

4.4 Ensemble Classifiers

Bagging (Breiman, 1996), for bootstrap aggregating, and *boosting* (Freund, Shapire, 1996) are two methods that allow to create composite classifiers by combining many, say T, versions of the same learner. Each version is called “weak” or “base” classifier. The different versions of the same classifier derive from allowing the learner system to focus on different subsets of the same training data during T replications. In each replication, bagging performs a sampling with replacement from the training data whereas boosting uses all the training data but maintains a weight for each record such that records that were misclassified in the previous iteration are more likely to be used to train the next classifier.

Bagging and boosting can be applied in both classification or regression problems. Given a new instance, the output of T models are combined by a voting strategy (for classification) or by averaging their outputs (for prediction). In bagging, all classifiers have the same weight, whereas in boosting, the more accurate the single classifier the more important its vote.

These methods are known to increase the accuracy of the learning algorithm. The reason for this can be searched in the *bias–variance decomposition* (Witten, Frank, 2005). The expected error of a classifier is the sum of a *bias* and a *variance* component. The *bias* relates to the persistent error of that particular learning algorithm, whereas the *variance* is due to that particular training set, which is inevitably finite and not fully representative of the whole population. Combining multiple models reduces the *variance* component of the expected error.

Breiman (1996) pointed out that bagging improves the accuracy of a learning algorithm when this is unstable, *i.e.,* little perturbation in the training set can lead to very different learnt models. Decision trees are known to be unstable, thus bagging improves the accuracy of these learners.

4.4.1 Bagging

Let n be the number of examples in the training dataset. For each of the T iterations, the algorithm applies the *bootstrap*[1] method to derive a training and test sets of approximately 63.2% and 36,8% of the original dataset. The algorithm applies a learning model to each of the training set and stores the resulting model. When it comes to classify a new instance, its class is predicted by each of the T models and the algorithm selects the most popular class.

4.4.2 Boosting

The major idea of boosting is to "maintain a distribution of weights over the training data" (Freund, Shapire, 1996). Weight works as the probability of an instance to occur and is manipulated by the algorithm at the end of each round t=1,..,T in order to force the next learner to focus on examples that were misclassified by the previous classifier.

Let *n* be the size of the training data. The algorithm starts by assigning the same weight $1/n$ to each instance. For each of the T iterations, boosting calls the weak learner on the training data, stores the model, computes its error *e* as the sum of the weights of the instances that it misclassified and, stores *e*. If the error is 0% or higher than 50%, boosting stops learning new models. Otherwise, the weights of the instances are adjusted to give more weight to instances that were misclassified by the last learner. This is done by multiplying the weight of each correctly classified instance by a factor $\frac{e}{1-e}$.

Then, boosting normalizes all the weights such that their sum is 1.

---

[1] Lets assume that we are given a training dataset of size n. The *bootstrap* method samples n instances with replacement from the original training dataset to create a new training data of the same size. Because it samples with replacement, it is certainly possible that some of the instances in the original dataset will be picked up more than once whereas others will not be considered: these ones will constitute the test set. The proportion of the test instances can be evaluated as follows:

Each time the algorithm chooses an instance, the probability that a record is not picked up is $(1-\frac{1}{n})$, and after *n* times is $\left(1-\frac{1}{n}\right)^n \sim \frac{1}{e} = 0.368$ (Witten, I.H., Frank, E. 2005). Thus, the test set consists of 36.8% of the original dataset and the training set consists of the remaining 63.2% (even if its size is n).

The estimated error combines the test error and training error according to this equation:

$$e = 0.632\ e_{test\ instances} + 0.368\ \mathrm{e}_{\text{training instances}}$$

When it comes to classify a new record, boosting uses the majority vote technique but weights the vote of a single learner according to the metric $-\log\left(\frac{e}{1-e}\right)$.

Chapter 5 – Experimental Design

This chapter reviews the dataset used in this thesis, the way it was created and the experimental design.

The dataset was derived in SAS BASE, whereas feature selection and classification were performed in WEKA. WEKA is an open source, Java–based data mining package developed at the University of Waikato (New Zeeland). It can be downloaded at http://www.cs.waikato.ac.nz/ml/weka.

5.1 Dataset

5.1.1 Dataset Description

The dataset consists of 133,481 examples and 15 features: 14 input variable and 1 class label (JOIN). Each instance regards a pair of images of manuscripts and is either a “join” or a “non join”, depending on whether or not the two fragments belong to the book originally.

Among the input variables, eleven represent the absolute difference in physical properties of the two images, such as the average height of the lines, average spacing between the lines, top margin, bottom margin, left margin, right margin, the height of the image, the width of the image, the height of the written portion and the width of the written portion and finally the number of lines. Among the remaining three input variables, COMPLETENESS_CALC is computed multiplying a completeness score of the two images, BIFOLIO assumes value 1 if none or both the fragments are bifolio and 0 otherwise, QTY_CONC is computed concatenating a quality description of both images.

Approximately 90% of the instances in the dataset represents “non join” (120,438) and the remaining 10% (13,043) represents “join”. The dataset was created imbalanced in order to reflect some how the reality in which among all the possible combinations of pairs of images only a small portion will be a “join”.

The table 5.1 summarizes the characteristics of the fifteen features in the dataset.

Feature	Type	Missing	Min	Max	Mean	Std Dev	Distinct Value
AVG_LINE_HEIGHT_DIFF	Num	0%	0	0.8	0.1	0.1	58,777
AVG_LINE_SPAC_DIFF	Num	0%	0	10.3	0.2	0.2	80,377
TOP_MRGN_DIFF	Num	0%	0	20.9	0.7	0.9	105,770
BOTTOM_MRGN_DIFF	Num	0%	0	21.5	0.9	1.0	108,575
LEFT_MRGN_DIFF	Num	0%	0	23.6	1.9	2.3	112,853
RIGHT_MRGN_DIFF	Num	0%	0	22.3	1.9	2.3	112,930
HEIGHT_DIFF	Num	0%	0	36.1	3.3	2.9	111,524
WIDTH_DIFF	Num	0%	0	27.7	3.9	3.4	112,561
WIDTH_WRITTEN_DIFF	Num	0%	0	21.5	1.9	1.6	113,874
HEIGHT_WRITTEN_DIFF	Num	0%	0	37	2.7	2.2	115,912
N_LINES_DIFF	Num	0%	0	109	8.6	7.5	76
COMPLETENESS_CALC	Num	0%	0	1	0.6	0.2	1,966
BIFOLIO	Boolean	0%	0	1	NA	NA	2
QTY_CONC	Char	0%	NA	NA	NA	NA	49
JOIN	Char	0%	NA	NA	NA	NA	2

Table 5.1: Summary of dataset.

It can be seen that none of the attributes assume a missing value. A part from QTY_CONC and BIFOLIO, which are respectively categorically–valued and Boolean, all input attributes are numerically–valued. It can be seen that the average values of the numerical attributes are commensurate.

5.1.2 How the dataset was derived

Images relative to Genizah fragments of manuscripts can be viewed as organized around four main concepts:

- joins
- classmarks
- fragments
- images

A join is a set of two or more classmarks, a classmark is a set of 1 or more fragments and each fragment has 2 images, recto and verso. Classmarks belonging to the same join

consists of fragments of manuscripts from the same original work.

The dataset used in this project was derived starting from two datasets provided by the FGP team. The first dataset, say IMAGES, consists of all images available at the FGP. Each instance is univocally identified by the image id (FGPI_IMG_NUM) and belongs to a unique classmark (INV_ID). All the physical measurements relative to images are available in this first table. The second table, say JOINS, consists of two fields: JOIN_ID and INV_ID. Thus, different classmarks are closely grouped together into joins.

The final dataset created consists of all possible combination of two images belonging to the same join ("joins") and a number of random matches of two images that are not known to belong to the same work ("non joins").

To obtain this dataset, the following steps were carried in SAS BASE:

- the two datasets IMAGES and JOINS were imported in SAS
- the table IMAGES was split into two tables of the same size, say IMAGES_1 and IMAGES_2. All fields in the first table were renamed with a suffix "_1" and all fields in the second table were renamed with a suffix "_2". An incremental id was added to both tables and used to join them. The table obtained, say IMAGES_NON_JOIN, consisted of half of the rows from the original dataset and twice the number of features. At this point, the absolute difference in the physical properties of the two images could be computed and a new field was added (JOIN), assuming the value "non join" for all the rows.
- To obtain all possible combination of two classmarks (INV_ID) belonging to the same join (JOIN_ID), the table JOINS was duplicated into two tables, say JOIN_1 and JOIN_2, renaming INV_ID with a suffix "_1" and "_2" accordingly. The two tables were then joined by JOIN_ID, which is equivalent to obtain all combination of two inventory ids from the same join. To eliminate duplications and instances consisting of the same inventory id, the table was sorted by JOIN_ID, INV_ID_1, INV_ID2 and only the rows having INV_ID_2 greater than INV_ID1 were retained. This last table was then merged to IMAGES to retrieve all information relative to inventory ids. At this point all absolute differences between the physical properties of the images were computed. A field JOIN was added to this table, say IMAGES_JOIN, assuming only the value "join".
- The final dataset was obtained reading from both IMAGES_NON_JOIN and IMAGES_JOIN.
- It has been carried out a final check to see whether randomly matched images to create "non joins" were not actually "joins".

Please refer to the Appendix A for the SAS code implemented.

5.2 Feature Selection

Features selection methods fall into two main groups: filters and wrappers. Filters produce a ranking of attributes or subsets of attributes based on general characteristics of the data, whereas wrappers evaluate the usefulness of a subset of attributes with respect to a particular domain and a particular learner. Filter methods can be further classified as univariate or multivariate depending on whether they assess features individually or taken in groups.

Section 5.2.1 reviews the experimental design for filter methods, 5.2.2 reviews the experimental design for wrapper methods and 5.3 reviews how results from feature selection were evaluated using classification.

5.2.1 Filter methods

5.2.1.1 Univariate technique

The worth of an attribute was evaluated based on two metrics:

- its Chi-squared statistic with respect to the class. This tests whether each input variable is independent from the class JOIN.
- its information gain (IG) with respect to the class.

The method was performed using 10-fold cross validation, *i.e.* performing a ranking separately for each of 10 folds of the dataset and then averaging the rankings.

5.2.1.2 Multivariate technique: Correlation–Based Filter (CFS)

An exhaustive search through the space of attribute subsets was performed on each of 10-folds of the dataset to find a good subset of features that were “highly correlated to the class, yet uncorrelated to each other” (Hall, 1999). The output of the filter is a list of attributes with an indication saying how many times (in how many folds) that attribute was selected to be part of the optimal subset.

5.2.2 Wrapper Method

The wrapper method was applied using as black box inducers a decision tree C4.5 (Quinlan, 1993) and rule-learner RIPPER (Cohen, 1995).

The accuracy of the classifier embedded within the feature selection process was estimated on a separate test set (1/3 of the original dataset) that was not seen during its induction.

The attribute space was searched through the best-first (BeFi) algorithm and the genetic algorithm (GA). As previously mentioned in chapter three, BeFi search doesn't terminate when the performance starts to fall but rather keeps a track of all attribute subsets evaluated so far, sorted in order of the performance measure, so that it can revisit an earlier configuration. The length of the list was set equal to five. Both directions of the search, Forward Selection (FoSe) and Backward Elimination (BaEl), were considered. Genetic algorithm was run setting the crossover probability to 0.6, mutation probability 0.033 and population size 20. The maximum number of generations was set to 20.

5.3 Evaluation of Feature Selection: Classification

The worth of the subsets of features returned by the different FS methods were evaluated using several machine learning algorithms. The basic idea was to compare the performance (accuracy) of a classifier on the original dataset and on the newly obtained dataset containing only the subset of features returned by the feature selection method.

The ranking of attributes' importance returned by univariate filters were evaluated by running a classification tree C4.5 on datasets consisting of the top *n* (n=1,2,..,14) attributes.

To estimate the goodness of correlation–based CFS filter, C4.5, bagged trees, boosted trees[2] and RIPPER were run ten times on the two datasets consisting of all the attributes and the dataset consisting of only those attributes that were included in the optimal subset. This was

[2] Bagged trees and boosted trees are abtained by combining ten decision trees C4.5 via bagging and boosting respectively.

done to obtain a confidence interval for the estimated accuracy for each classifier. Moreover, a paired t-tests on each shuffle of the dataset was performed to compare the accuracy of the 4 inducers at significance level 0.05. The experiment was conduced using the Experimenter tool in WEKA.

The same procedure was followed to evaluate the goodness of the subsets returned by wrapper methods. The subsets returned by wrapper using C4.5 as evaluation function and employing a BeFiFoSe, BeFiBaEl, and GA search, were assessed by running 10 times a C4.5, a boosted tree and a bagged tree on the datasets containing all the attributes and on datasets consisting of only the attributes in the “optimal” subsets. The goodness of subsets returned by the wrapper embedding a RIPPER and searching via BeFiFoSe, BeFiBaEl and GA were assessed by running 10 times RIPPER on the original dataset and on a datasets with only the “optimal” features.

Final note. All Feature selection methods were run on the entire datasets, whereas classification methods were run on a random stratified subsample of the dataset (50%) for computational complexity reasons.

Chapter 6 – Experimental Results

6.1 Filters

6.1.1 Univariate Filters

Univariate filters were used to produce a ranking for each of the ten folds in which the dataset was partitioned and their results were averaged over the folds.

Table 6.1 reports the average ranking of attributes using χ^2 $test$ and IG as evaluation metrics. The standard deviation is reported between brackets. The results obtained are very similar, a part from small local "swaps".

FEATURE	Avg Ranking 10–Fold (χ^2)	Avg Ranking 10–Fold (Information Gain)
AVG_LINE_SPAC_DIFF	1.4+-0.49	1.4+-0.49
AVG_LINE_HEIGHT_DIFF	1.6+-0.49	1.6+-0.49
WIDTH_DIFF	3.1+-0.3	3.1+-0.3
HEIGHT_DIFF	3.9+-0.3	3.9+-0.3
WIDTH_WRITTEN_DIFF	5.4+-0.66	5+-0
LEFT_MRGN_DIFF	6.6+-1.02	7.2+-0.75
HEIGHT_WRITTEN_DIFF	6.8+-0.87	6.3+-0.46
RIGHT_MRGN_DIFF	7.2+-0.98	7.5+-0.67
BOTTOM_MRGN_DIFF	9.1+-0.3	9.4+-0.49
TOP_MRGN_DIFF	9.9+-0.3	10.4+-0.66
COMPLETENESS_CALC	11+-0	10.2+-0.87
N_LINES_DIFF	12+-0	13+-0
QTY_CONC	13+-0	12+-0
BIFOLIO	14+-0	14+-0

Table 6.1: Average ranking over ten folds using χ^2 test and IG

The 5 most important features are the same in both methods. The 6th and 7th most important features are swapped in the two rankings and the same for the 10th and 11th and 12th and 13th. The features in the 8th, 9th and 14th position are the same in both ranking.

Being the ranking very similar, from now on only χ^2 test will be considered.

Time taken to return these rankings was about 1 minute.

6.1.1.1 Evaluation Univariate Filters

To evaluate the result of the filter method, a classification tree C4.5 was applied to the dataset consisting of the *n* most important features (*n*=1,2,..,14).

The results are summarized in Table 6.2.

Top n Features	Estimated Accuracy on holdout set	n° Leaves	Size Tree	Time build model (sec)
1	93.20%	103	205	6.15
2	94.24%	593	1,185	10.37
3	94.77%	697	1,393	12.51
4	95.15%	843	1,685	17.61
5	95.34%	923	1,845	24.05
6	95.36%	966	1,931	26.13
7	95.49%	984	1,967	29.38
8	**95.57%	1,031	2,061	38.29
9	95.35%	1,063	2,125	36.33
10	95.34%	1,119	2,237	43.16
11	95.44%	1,145	2,289	50.59
12	95.47%	1,291	2,581	60.46
13	95.28%	1,822	2,750	56.24
14	95.33%	1,862	2,830	78.32

Table 6.2 Performance of C4.5 using the top n features returned by univariate filter.

It can be seen that just by using the best attribute (the difference in the average spacing between lines) the estimated accuracy on the hold out set is quite high (93,20%). The

estimated accuracy increases while including the next best attributes and reaches its maximum when the eight most relevant feature are taken together. This represents an increase of 0.24% respect to the baseline accuracy when all attributes are included. When the 9th most important feature is added, BOTTOM_MRGN_DIFF, the accuracy drops to its baseline value 95,33% and remains stable around this value when the 10th attribute, TOP_MRGN_DIFF, is added to the subset. The accuracy starts to increase slightly when the completeness measure and the quality information are added (95.47%) before dropping again to the base value when all attributes are considered.

Chart 6.1 reports the estimated accuracy of C4.5 built considering the top *n* features (n=1,..,14).

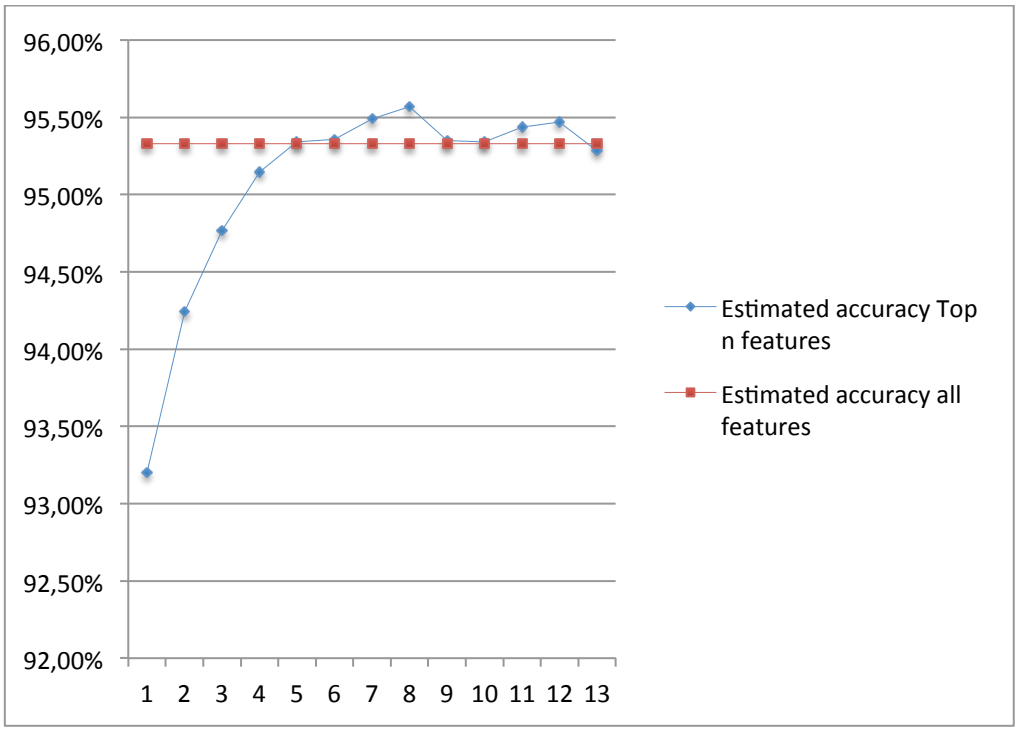


Chart 6.1: Estimated accuracy of C4.5 using the top n features returned by univariate filter.

6.1.2 Multivariate Filter

CFS was run against ten separate folds of the entire dataset and returned the number of times (in how many folds) a given attribute was chosen to be part of the optimal subset.

Table 6.3 reports the results of CFS.

Attribute	Number of folds	In the optimal subset
HEIGHT_DIFF	10	Yes
WIDTH_DIFF	10	Yes
AVG_LINE_HEIGHT_DIFF	10	Yes
AVG_LINE_SPAC_DIFF	10	Yes
LEFT_MRGN_DIFF	10	Yes
RIGHT_MRGN_DIFF	10	Yes
TOP_MRGN_DIFF	10	Yes
BOTTOM_MRGN_DIFF	10	Yes
WIDTH_WRITTEN_DIFF	10	Yes
HEIGHT_WRITTEN_DIFF	10	Yes
COMPLETENESS_CALC	0	No
QTY_CONC	0	No
N_LINES_DIFF	0	No
BIFOLIO	0	No

Table 6.3: Number of times an attribute was chosen to be part of the optimal set.

The selected features correspond to the 10 most relevant features returned by the univariate filters in the previous section.

6.1.2.1 Evaluation Multivariate Filter

Let $dataset_1$ represent the original dataset and $dataset_2$ the dataset consisting of the features returned by CFS.

Decision tree C4.5, bagged tree, boosted tree and RIPPER were run ten times against $dataset_1$ and $dataset_2$ and the estimated accuracies are reported in table 6.4. At each run, datasets were randomly split in training (66%) and test sets (34%). The table reports the average accuracy over ten runs and its standard deviation between brackets. WEKA computes a paired t-test to compare the performance of the different classifiers with respect to the base classifier, here C4.5. Notations v and * indicate that there is enough statistical evidence (at confidence level 0.05) to conclude that the result is higher or lower than the baseline (C4.5) estimate.

DATASET	C4.5	BAGGED TREE	BOOSTED TREE	RIPPER
$dataset_1$	95.09 (0.18)	96.18 (0.14) v	97.12 (0.09) v	94.50 (0.11) *
$dataset_2$	95.00 (0.11)	96.20 (0.13) v	96.97 (0.10) v	94.29 (0.12) *

Table 6.4: Estimated accuracy of the four classifiers on the original dataset and on the dataset consisting of only the optimal subset of features returned by CFS.

It can be seen that, a part from the bagged tree, CFS decreased the estimated accuracy of all classifiers. The classifier mainly impacted is RIPPER, which accuracy is decreased by 0.21%.

It can also be seen that there is enough statistical evidence to conclude that bagged tree and boosted tree are more accurate on both datasets than C4.5, whereas RIPPER is worse in both datasets.

Table 6.5 reports the averaged User CPU Time (sec) over ten runs for each classifier on the two datasets. This CPU time is related to the induction phase.

DATASET	C4.5	BAGGED TREE	BOOSTED TREE	RIPPER
$dataset_1$	10.77 (1.13)	83.06 (3.22) v	122.50 (7.18) v	89.52 (18.59) v
$dataset_2$	7.95 (0.75)	66.52 (1.88) v	184.27 (23.27) v	86.07 (18.61) v

Table 6.5: Averaged user CPU time to train the four classifiers on the original dataset and on the dataset consisting of only the optimal subset of features returned by CFS.

It can be seen that, a part from the boosted tree, CFS improved the learning time of all classifiers, particularly the one of the bagged tree (by 16.54 sec). It can also be seen that, on both datasets, the time required to induce the bagged tree, the boosted tree and RIPPER is statistically longer than for C4.5.

Table 6.6 reports the averaged complexity, in terms of number of rules produced, of C4.5 and RIPPER over ten runs.

DATASET	C4.5	RIPPER
$dataset_1$	976.90 (109.50)	31.70 (3.06)*
$dataset_2$	583.00 (28.55)	34.50 (3.98)*

Table 6.6: Averaged number of rules produced by RIPPER and C4.5 when run against the original dataset and the one consisting of only the optimal subset of features returned by CFS.

It can be seen that CFS improves the complexity of C4.5, both in terms of mean and standard deviation. In fact, at the expense of a small reduction in accuracy by 0.09%, the new tree produced 394 less rules. CFS slightly increases the complexity of RIPPER, requiring about 3 more rules to learn the concept of “join”. From this table it can also be seen how RIPPER produces more compact rulesets compared to C4.5 (about 945 less rules when looking at $dataset_1$).

6.2 Wrapper

Wrapper method was applied using as evaluator a C4.5 and RIPPER.

Although the number of attributes is small (14), it was prohibitive to run the wrapper method with an exhaustive search. This would have meant running each learning algorithm incorporated in the wrapper method $2^{14} = 16{,}384$ times. For this reason, the space of subsets of attributes was searched heuristically.

Three types of searches were considered:

- with a BeFi search, starting with no attribute (forward selection) BeFiFoSe
- with a BeFi search, starting with all the attributes (backward elimination) BeFiBaEl
- with a GA search.

Table 6.7 summarizes the subsets returned by the different wrapper methods.

Learning Algorithm	Search Method	"Optimal" Subset	Time taken to find subset
C4.5	BeFiFoSe	HEIGHT_DIFF,WIDTH_DIFF, AVG_LINE_HEIHGT_DIFF, AVG_LINE_SPAC_DIFF, LEFT_MRGN_DIFF, WIDTH_WRITTEN_DIFF, COMPLETENESS_CALC	21 minutes
C4.5	BeFiBaEl; GA	HEIGHT_DIFF, WIDTH_DIFF, N_LINES_DIFF, AVG_LINE_HEIGHT_DIFF, AVG_LINE_SPAC_DIFF, LEFT_MRGN_DIFF, RIGHT_MRGN_DIFF, BOTTOM_MRGN_DIFF, WIDTH_WRITTEN_DIFF, COMPLETENESS_CALC, QTY_CONC	35minutes/50 minutes
RIPPER	BeFiFoSe; BeFiBaEl; GA	All attributes	More than 18 Hours

Table 6.7: Optimal subsets returned by wrapper employing BeFiFoSe, BeFiBaEl and GA for C4.5 and RIPPER

The subset returned by the wrapper using C4.5 and employing a BeFiBaEl was the same as the one employing a GA and consisted of 11 features. The wrapper using C4.5 and employing

a BeFiFoSe returned a subset of only 7 features. The time taken to find the “optimal” subset with BeFiFoSe was 21 minutes, whereas took longer searching with BeFiBaEl (35 minutes) and with GA (about 50 minutes).

It can be seen that the “optimal” subset returned by BeFiFoSe consisted of the first six most relevant features returned by the univariate filter and one irrelevant feature, the completeness measure.

The subset returned by BeFiBaEl or GA consisted of the first five most relevant features returned by the univariate filter and three irrelevant features filtered out by CFS. It can be seen how wrapper may prefer irrelevant features to more relevant ones, mainly if the relevant features not selected are redundant. However, the features excluded were not redundant as they were not filtered by the multivariate filter CFS. Thus, combining the less relevant features between them or with the remaining features included in the optimal set resulted in a higher accuracy estimate than including just the most relevant features. This is probably due to the presence of higher order interactions between features. This kind of interactions can be detected only by the wrapper and especially when it starts with the entire set of attributes (backward elimination).

Running the wrapper method using RIPPER’s performance as internal evaluator of subsets and searching via BeFiFoSe, BeFiBaEl and GA returned as “optimal” subset one consisting of all attributes. Wrapper using RIPPER required a long time to return the optimal solution, ranging from 18 hours to one day.

6.2.1 Evaluation Wrapper

6.2.1.1 Wrapper using Decision Tree C4.5

Let $dataset_1$ represent the original dataset, $dataset_3$ the dataset consisting of only those features returned by the wrapper using BeFiFoSe and $dataset_4$ the dataset consisting of only those returned using BeFiBaEl or GA.

Table 6.8 reports the estimated accuracy of C4.5, bagged and boosted tree over ten runs against the three datasets $dataset_1, datase_2, dataset_3$.

DATASET	C4.5	BAGGED TREE	BOOSTED TREE
$dataset_1$	95.09 (0.18)	96.18 (0.14)	97.12 (0.09)
$dataset_3$	95.22 (0.19)	96.16 (0.09)	96.97 (0.08)
$dataset_4$	96.10 (0.17)	96.14 (0.13)	97.05 (0.09)

Table 6.8: Estimated accuracy of C4.5, bagged tree and boosted tree on the entire dataset, on the dataset returned by wrapper searching forward and on the dataset searching backwards or via genetic algorithm.

It can be seen from the table that wrapper method improved the accuracy of C4.5, mostly when it searched backwards or via genetic algorithm. The estimated accuracy averaged over the runs is 1.01% higher than the accuracy when all features are included in the dataset. The fact that the major improvement is obtained searching the space backward or including randomization may suggest the presence of higher order interactions, that by searching the space starting from no attribute and adding one attribute at a time, cannot be found.

Bagging or boosting C4.5 working on subsets returned by the wrapper decreased their estimated accuracies. The highest decrease is registered when boosting ten C4.5 working on datasets consisting of the small subset returned by BeFiFoSe.

Table 6.9 reports the averaged CPU time for inducing the three supervised learners over ten runs on the three datasets. It can be seen that almost all the classifiers required less time to build their model when using only the subsets of attributes returned by the wrappers. BeFiFoSe improved the performance of both C4.5 and the bagged tree: C4.5 required 4.41 seconds less to learn the model (which performs better also in terms of accuracy) and the bagged tree required about 24 seconds less. BeFiFoSe degraded the learning time of the boosted tree, which required about 12 more seconds. BeFiBaEl or GA improved the performance of all three classifiers, particularly the boosted tree, which required about 33 less seconds. The paired t-tests prove that inducing bagged tree or boosted tree requires more time than inducing a single decision tree.

DATASET	C4.5	BAGGED TREE	BOOSTED TREE
dataset_1	10.77 (1.13)	83.06 (3.22) v	122.50 (7.18) v
dataset_3	6.36 (0.53)	59.12 (2.68) v	134.99 (16.38) v
dataset_4	8.42 (0.84)	66.25 (4.82) v	89.73 (3.79) v

Table 6.9:User CPU time required to learn C4.5, bagged tree and boosted tree on the entire dataset, on the dataset returned by wrapper searching forward and the dataset searching backwards or via genetic algorithm.

Table 6.10 shows the complexity, measured in terms of number of leaves or rules, of the decision tree C4.5 on the three datasets.

DATASET	C4.5
$dataset_1$	976.90 (109.50)
$dataset_3$	526.40 (34.85)
$dataset_4$	1001.50 (82.99)

Table 6.10: Number of rules produced by C4.5 on the entire dataset, on the dataset returned by wrapper searching forward and the dataset searching backwards or via genetic algorithm.

It can be seen that wrapper traversing the search space via BeFiFoSe gives better result in terms of number of rules produced than the one using BeFiBaEl or GA. In fact, the number of leaves when BeFiFoSe is employed is about 475 below the number of rules produced by C4.5 using the subset returned by BeFiBaEl or GA. BeFiFoSe improved the complexity of C4.5 without feature selection by about 450 leaves.

6.2.1.2 Wrapper using RIPPER

As seen in section 6.2, the wrapper using RIPPER as internal evaluator of the usefulness of subsets of features returned the entire set of attributes.

6.3 Summary of Performance of CFS and Wrappers for the four Classifiers

The following tables summarizes the averaged accuracy, user CPU time and complexity (number of rules) of C4.5, bagged tree and boosted tree when run ten times against 1) the original dataset, 2) the dataset consisting of only the features returned by CFS, 3) the dataset consisting of only the features returned by the wrapper performing BeFiFoSe, and 4) the dataset with the subset of features returned by the wrapper performing BeFiBaEl or GA. The estimated performance, user CPU time and complexity are reported also for RIPPER, that was run against the first two datasets. In the case of RIPPER $dataset_1$ represents both the original data set and the dataset consisting of the features returned by the wrapper method. Performance for a given classifier that were better than its baseline performance on the original set presented in green, otherwise in red. The best performance for a given classifier is emphasised in bold.

Accuracy

	C4.5	Bagging	Boosting	RIPPER
$Dataset_1$	95.09(0.18)	96.18(0.14)	97.12(0.09)	**94.50(0.11)**
$Dataset_2$	95.00(0.11)	**96.20(0.13)**	96.97(0.10)	94.29(0.12)
$Dataset_3$	95.22(0.19)	96.16(0.09)	96.97(0.08)	
$Dataset_4$	**96.10(0.17)**	96.14(0.13)	97.05(0.09)	

Table 6.11: Estimated accuracy of C4.5, bagged tree, boosted tree, and RIPPER on the entire dataset, the dataset consisting of optimal subset returned by CSF and on the datasets returned by wrapper searching forwards and backwards.

User CPU Time Training

	C4.5	Bagging	Boosting	RIPPER
$Dataset_1$	10.77(1.13)	83.06(3.22)	122.50(7.18)	89.52(18.59)
$Dataset_2$	7.95(0.75)	66.52(1.88)	184.27(23.27)	**86.07(18.61)**
$Dataset_3$	**6.36(0.53)**	**59.12(2.68)**	134.99(16.38)	
$Dataset_4$	8.42(0.84)	66.25(4.82)	**89.73(3.79)**	

Table 6.12: User CPU time required for learning C4.5, bagged tree, boosted tree and RIPPER on the entire dataset, the dataset consisting of optimal subset returned by CSF and on the datasets returned by wrapper searching forwards and backwards.

Complexity Model – Number Rules

	C4.5	RIPPER
$Dataset_1$	976.90(109.50)	**31.70(3.06)**
$Dataset_2$	583.00(28.55)	34.50(3.98)
$Dataset_3$	**526.40(34.85)**	
$Dataset_4$	1001.50(82.99)	

Table 6.13: Number of rules produced by C4.5 and RIPPER on the entire dataset, the dataset consisting of optimal subset returned by CSF and on the datasets returned by wrapper searching forwards and backwards.

It can be seen that C4.5's best performance in terms of accuracy was obtained using the wrapper method searching backwards, its best time performance and complexity were obtained by using the wrapper searching forward.

It seems that combining multiple C4.5 via bagging or boosting after having performed feature selection on the dataset based on wrappers using a single C4.5 results in a lower estimated accuracy.

Boosting requires also more time to build the model when run against datasets consisting of features returned by feature selection methods. I think this is due to the fact that being an

iterative process that focus more and more on “hard” examples, it has to build more and more complex model, before being able to include the less relevant features.

RIPPER performs best in terms of accuracy when using the wrapper method rather than the multivariate CFS method. Its learning time is reduced using CFS rather than wrapper (by about 3.45 sec). The best complexity is achieved when it is run on the entire dataset (it produces three less rules). An example of the rules produced by RIPPER with and without CFS are reported in Appendix C.

Chapter 7 – Conclusion

The goal of this project was to integrate the effort of the Friedberg Genizah Project (FGP) Computerization Unit in discovering "joins" with data mining techniques to extract useful knowledge from data. Data used in the project was derived from results of the FGP's algorithm, which runs against a large dataset of digitalized images of medieval manuscripts. The algorithm extracts several information about the images, such as the average spacing between lines in an image, average height of lines, number of lines, etc. There are other information extracted automatically that unfortunately were not available for this project, such as information on the handwriting, content, style scripting. The information extracted was used by them to test whether the content of pairs of images could be or not be from the same book. In the first case, the pair of images is labeled as "join", in the second case as "non join". A batch of newly discovered "joins" were then submitted to the attention of domain experts to validate their findings.

The idea of this project was to use the validated "joins" to: 1) train several supervised learning algorithms to learn the concept of "join", and 2) to assess the importance of features in discriminating between "joins" and "non joins".

A dataset was derived consisting of the validated "joins" immersed in a multitude of "non joins", random pairs of images. The proportion "join"/"non join" was kept unbalanced 1:9.

Several feature selection methods were used, both to estimate features' general *relevance*, based on the characteristics of data, and their *usefulness* for a particular classification purpose.

Univariate filters based on chi–squared test and information gain were considered to return a ranking of variables importance. These filters produced a ranking for each of the ten folds in which the dataset was partitioned and their results were averaged over the folds. Results from the two univariate filters were very similar. It was seen that, by only using the most relevant feature in the dataset, the difference between the average spacing between lines in the two images, a decision tree C4.5 could reach a classification accuracy of 93.2%. Univariate filters do not consider interactions between features so that it is impossible to say whether there are redundant features or features that can be useful only in presence of others.

For this reason, CFS correlation–based multivariate filter (Hall, 1999) was employed. An exhaustive search through the space of variables subsets was employed to allow CFS to find a subset of attributes that individually were highly correlated to the class but uncorrelated between them. CFS was run against ten separate folds of the entire dataset and returned the number of times (in how many folds) a given attribute was part of the optimal subset. CFS returned the top ten most relevant features. I concluded that the features excluded from the optimal subset, QTY_CONC, COMPLETENESS_CALC, BIFOLIO, N_LINES_DIFF, were not redundant, as they were not in any way derived from other features. Thus, CFS excluded the less promising features, based on their individual ability to discriminate between a "join" and a "non join". To evaluate the goodness of the optimal subset returned by CFS, a decision tree C4.5, a bagged tree, a boosted tree and RIPPER were used. CFS degraded the accuracy performance of all learners, a part from the bagged tree. However, CFS improved the learning time of all learners, except the boosted tree. CFS fails in detecting features that are relevant only when they assume a restricted set of values (locally relevant) or that are relevant only in presence of others features (Hall, 1999). By applying CFS, I conclude that there were not redundant features in the domain and that there are feature interaction and features locally predictive (for example QTY_CONC).

Wrapper approach was employed to evaluate the usefulness of a subset of features to a particular classifier for this domain. Wrapper calls a specified classifier each time a subset of attributes needs to be evaluated. The classifiers used as black box inside the wrapper were decision tree C4.5 and rule–learner RIPPER. Although wrapper is very simple conceptually, it is very expensive from a computational point of view. For this reason, it was prohibitive to search extensively through the space of features subsets. That is why a genetic algorithm and best-first search were considered. Best-first was employed both starting from: 1) an empty set of features and searching forward and, 2) from the entire set of features and moving backwards. Wrapper using RIPPER as internal evaluator returned as "optimal" subset the entire set of attributes. Wrapper internally embedding C4.5 and searching forward returned a subset of 7 attributes, including one irrelevant feature filtered by CFS: COMPLETENESS_CALC. Wrapper searching backwards or via genetic algorithm returned a larger "optimal" subset, consisting 11 attributes, three of which were filtered by CFS (COMPLETENSS_CALC, N_LINES, QTY_CONC).

In this project, the following classifiers were used: decision tree C4.5, bagged tree, boosted tree and, rule–learner RIPPER. The inducer that gave a better insight on the mapping between input variables and the class was RIPPER. RIPPER produces a set of IF-THEN-

ELSE rules that are way more compact in this case than the rules extracted by a decision tree C4.5. They are very useful in that they give an immediate idea of what defines a "join" and in which regions of the search space to focus to find other "joins".

An experiment consisting in running ten times each classifier against each of the datasets consisting of subsets of features returned by CFS and wrappers was performed and a paired t-test on the estimated accuracy, user CU time and complexity of the models was carried out in WEKA to compare their performance.

C4.5 performed better in terms of accuracy when the dataset used consisted of the subset of features returned by the wrapper searching backwards (or genetic algorithm). These may prove that there is a higher (than two) order interactions between features. In fact, searching forward may detect second order interaction, while starting from the entire set of attributes and trying to remove one at a time can help detecting higher order. Paired t-tests showed that there was statistical evidence (at confidence level 0.05) to conclude that boosted and bagged trees performs better than a single C4.5. These tests reported also that RIPPER performs worse than C4.5 in terms of estimated accuracy but also that the complexity of the ruleset induced by RIPPER was better than the one returned by C4.5. From the experiment done, the ensemble classifiers obtained by combining ten decision trees C4.5 via bagging and boosting applied to datasets consisting of subsets of features “optimal” for a *single* C4.5 performed worse than when applied to the dataset consisting of all features.

In general, classifiers performed surprisingly well, although despite the fact that only physical features of fragments were taken into account. The best classifier in terms of accuracy was the boosted tree, which achieved an estimated accuracy of 97.12%.

FGP’s Computerization team found these results very promising. The estimated accuracies of classifiers were high if considered that only the physical features of images were considered in this project. Future extensions could be to include information about the author, handwriting, content and style. Classifiers that work well on imbalanced datasets could be investigated. Moreover, classifiers could be used attaching a higher cost to false positive. Finally, I would like to further investigate about statistical test to compare the same classifier’s accuracies when run against different datasets, for example, those consisting of only a subset of attributes because of feature selection.

Bibliography

Almuallim, H., Dietterich, T.G., (1992). Learning with many irrelevant features. In: *Proceedings of Ninth National Conference on Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, 547–552.

Bins, J., Draper, B.A., (2001). Feature Selection from Huge Features Sets. In: *8th IEEE International Conference on Computer Vision*, 159-165.

Blum, A. L., Langley, P., (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*. **97**(1–2), 245–271.

Breiman, et *al*., (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Breiman, L., (1994). Bagging predictors. Technical Report 421, Department of Statistics, University of California at Berkeley.

Breiman, L., (1996). Out-of-bag Estimation. Technical report, Department of Statistics, University of California at Berkeley.

Breiman, L.(2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Cardie, C., (1993). Using decision trees to improve case-based learning. In: *Proceedings of Tenth International Conference on Machine Learning*, 25–32.

Cendrowska, J., (1987). PRISM: An algorithm for inducing modular rules. *Int. J. Man–Machine Studies*. **27**, 349–370.

Cohen, W., (1995). Fast effective rule induction. In: *Proceedings of the Twelfth International Conference on Machine Learning*. 115–123.

Das, S. (2001). Filters, wrappers and a boosting-based hybrid for feature selection. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann Publishers, 74–81.

Dash, M., Liu, H., (1997). Feature selection for classification. *Intelligent Data Analysis*. **1**(1–4), 131–156.

Demsar, J., (2006). Statistical comparison of classifiers over multiple data sets. *JMLR* 7:1–30.

Forman, G., (2003). An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*. **3**, 1289–1305.

Frank, E., Witten, I. H., (1998). Generating Accurate Rule Sets Without Global Optimization. In: *Fifteenth International Conference on Machine Learning*. 144-151.

Freund, Y., Schapire, R.E., (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In: *Proceedings 2nd European Conference on Computational Learning Theory*, Springer, Berlin (1995), 23–37.

Freund, Y., Schapire, R.E., (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*. **14**(5), 771–780.

Furnkranz, J., (1996). Separate–and–conquer rule learning. Technical Report TR–96–25, Austrian Research Institute for Artificial Intelligence, Vienna.

Furnkranz, J., (1997). Pruning algorithms for rule learning. *Machine Learning*, **27**(2), 139–171.

Furnkranz, J., (2005). ROC `n' Rule Learning – Towards a Better Understanding of Covering Algorithms. *Machine Learning*. **58** (1), 39–77.

Furnkranz, J., Widmer, G., (1994). Incremental Reduced Error Pruning. In: *Machine Learning: Proceedings of the Eleventh Annual Conference*, 70–77.

Guyon, I., Elisseeff, A., (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*. **3**, 1157–1182.

Hall, M.A., (1999). Correlation-based feature selection machine learning, Ph.D. Thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Hand, D. J.,Smyth, P., Mannila, H. (2001). *Principles of data mining*. Cambridge, MA: MIT Press.

Hyafil, L., Rivest, R.L., (1976). Constructing optimal binary decision trees is NP-complete. Inform. Process. Lett. 5(1), 15–17.

Janecek, A., (2009). Efficient Feature Reduction and Classification Methods, Ph.D, Thesis, Department of Computer Science, Universitat Wien, Austria.

Japkowicz, J., (2008). Classifier evaluation: A need for better education and restructuring. In: Proceedings of the 3rd Workshop on Evaluation Methods for Machine Learning, ICML 2008, Helsinki, Finland, 2008.

John, G. H., Kohavi, R. , Pfleger, K., (1994). Irrelevant features and the subset selection problem. In: Cohen, W., Hirsh, H. eds. Machine Learning: Proceedings of the Eleventh International Conference. San Francisco, CA: Morgan Kaufmann Publishers, 121–129.

John, G.H., (1997). Enhancements to the data mining process, Ph.D. Thesis, Computer Science Department, Stanford University, CA, USA.

Kass, G. V., (1980). An exploratory technique for investigating large quantities of categorical data*. Applied Statistics*. **29**(2), 119–127.

Kira, K., and Rendell, L.A., (1992). A practical approach to feature selection. In: Sleeman, D.H., Edwards, P. *Proceedings of the Ninth International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann Publishers, 249–256.

Kohavi, R., John, G.H., (1997). Wrappers for Feature Subset Selection. *Artificial Intelligence Journal*. **97**(1-2), 273-324.

Kohavi, R., JOHN, G.H., (1997). Wrappers for feature subset selection. *Artificial Intelligence*. **97**(1–2), 273–324.

Koller, D., Sahami, M., (1996). Toward optimal feature selection. In: *Proceedings of the Thirteenth International Conference on Machine Learning*. Morgan Kaufmann, 284–292.

Langley, P., (1988). Machine learning as an experimental science. *Machine Learning*, **3**, 5–8.

Langley, P., (1994). Selection of relevant features in machine learning. In: *Proceedings of the AAAI Fall Symposium on Relevance*.

Liu, H., et al. (2002). A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. *Genome Inform.* **13**, 51–60.

Lui, H., Yu, L., (2003). Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. In ICML, pages 856–863

Michalski, R.S. (1969). On the quasi-minimal solution of the covering problem. In: *Proceedings of the 5th International Symposium on Information Processing*. Bled, Yugoslavia. 1969. 125–128.

Moore, A.W., Lee, M.S., (1994). Efficient algorithms for minimizing cross validation error. In: *Proceedings of Eleventh International Conference on Machine Learning*, Morgan Kaufmann, New Brunswick, New Jersey, 190–198.

Murthy, S. K., (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery.* **2**(4), 345–389**.**

Neville, P.G., (1999). Decision Trees for Predictive Modeling. *SAS Institute Inc.*

Nong, Y., (2003). Handbook of data mining. Mahwah, NJ: Lawrence Erlbaum Associates.

Pagallo, G., Haussler, D., (1990). Boolean feature discovery in empirical learning. *Machine Learning*. **5**, 71–99.

Pang-Ning, T. Steinbach, M., Kumar, V., (2005). Introduction to Data Mining. Addison Wesley, 1st edition.

Pietraszeka, T., Tannera, A., (2005). Data Mining and Machine Learning Towards Reducing False Positives in Intrusion Detection. *Information Security Technical Report.* **10**, 169–183.

Quinlan, J.R., (1986). Induction of decision trees. *Machine Learning.* **1**, 81-106.

Quinlan, J.R., (1987). Generating production rules from decision trees. In: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*. Milan, Italy. Morgan Kaufmann. 304–307.

Quinlan, J.R., (1990). Probabilistic decision trees. In: Kodratoff, Y., Michalski, R. (eds.) *Machine learning: An artificial intelligence approach, volume III*. San Mateo, CA: Morgan Kaufmann, 140–152.

Quinlan, J.R., (1993). *C4.5: Programs for machine learning.* San Francisco CA: Morgan Kaufmann.

Quinlan, J.R., (1996). Bagging, Boosting, and C4.5. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence.* 725–730.

Rokach, L., Maimon, O. (2005). Top–Down Induction of Decision Trees Classifiers–A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*. **35**(4), 476- 487.

Tu, P.L., Chung, J.Y., (1992). A New Decision-Tree Classification Algorithm for Machine Learning. In: *Proceedings of the 4th IEEE International Conference on Tools with Artificial*

Intelligence, 1992.

Vafai, H., De Jong, K., (1992). Genetic algorithms as a tool for feature selection in machine learning. In: *Proceedings 4th International Conference on Tools with Artificial Intelligence.* 200-203.

Vafai, H., De Jong, K., (1993). Robust feature selection algorithms. In: *Proceedings 5th International Conference on Tools with Artificial Intelligence.* 356-363.

Witten, I.H., Frank, E., (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd ed. San Francisco CA: Morgan Kaufmann.

Wolf, L., et *al.,* (2009). Automatically Identifying Join Candidates in the Cairo Genizah. Post ICCV workshop on eHeritage and Digital Art Preservation. 2009.

Wolf, L., et *al.,* (2011). Automatic paleographic exploration of Genizah manuscripts**.** In: Codicology and Palaeography in the Digital Age II. 2011, Norderstedt: Books on Demand, Germany.

Xing, E., Jordan, M., Karp, R. (2001). Feature selection for high-dimensional genomic microarray data. *Proceedings of the Eighteenth International Conference on Machine Learning.* San Francisco, CA: Morgan Kaufmann Publishers, 601–608.

Xu, L., Yan, P., Chang, T., (1988). Best first strategy for feature selection. In: *Proceedings 9th International Conference on Pattern Recognition.* 14-17 Nov 1988. 706-708.

Appendix A

The following code was developed to import information relative to images and joins into a SAS library and tables. The main instructions used were *data steps,* to read and write datasets, *proc sort,* to sort tables, *merge statements,* to join tables and *proc sql* to use native SQL language in SAS environment.

SAS CODE

```
/*Allocate the Library*/
LIBNAME LIBRERIA "C:\Documents and Settings\Administrator\Desktop\Tesi Images\TABELLE";

/*Write a dataset reading from the input file containing all images*/


DATA IMPORT_IMAGES;
INFILE "C:\Documents and Settings\Administrator\Desktop\Tesi Images\DATI\IMAGES.TXT" DELIMITER='09'X DSD;
INPUT INV_ID  : $10.
      IMG_NAME  : $60.
      FGPI_IMG_NUM :  $
      DPI
      DPI_GRID
      RESZ_SCL
      NUM_FRAG
      INIT_BB_H
      INIT_BB_W
      BB_ARRAY : $
      FN_ROT_ANG
      HEIGHT
      WIDTH
```

```
        BIF_LOCX1
        BIF_LOCX2
        BIF_LOCY1
        BIF_LOCY2
        NUM_DESC
        FRAG_QTY : $30.
        N_TEXT_COMP
        N_LINES
        AVG_LINE_H
        AVG_LINE_SPAC
        LEFT_MRGN
        RIGHT_MRGN
        TOP_MRGN
        BOTTOM_MRGN
        COMPLETENESS : COMMAX15.;
RUN;

/*Keep only some of the features*/
/* Compute new attributes, such as BIFOLIO, WIDTH_WRITTEN, HEIGHT_WRITTEN*/
DATA IMAGES;
SET IMPORT_IMAGES;
KEEP
INV_ID
IMG_NAME
FGPI_IMG_NUM
DPI
DPI_GRID
HEIGHT
WIDTH
BIF_LOCX1
BIF_LOCX2
```

```
BIF_LOCY1
BIF_LOCY2
Bif
FRAG_QTY
N_LINES
AVG_LINE_H
AVG_LINE_SPAC
LEFT_MRGN
RIGHT_MRGN
TOP_MRGN
BOTTOM_MRGN
width_written
height_written
COMPLETENESS ;
if (completeness <1) and (dpi <>0 or dpi_grid <>0) and (dpi <700)   ;
if dpi <>0 then do ;
 height=height/dpi;
 width=width/dpi;
 LEFT_MRGN=LEFT_MRGN/dpi;
 RIGHT_MRGN=RIGHT_MRGN/dpi;
 TOP_MRGN=TOP_MRGN/dpi;
 BOTTOM_MRGN=bottom_mrgn/dpi;
AVG_LINE_H   =  AVG_LINE_H/dpi;
AVG_LINE_SPAC =   AVG_LINE_SPAC/dpi;
end;
if dpi_grid <>0 then do ;
 height=height/dpi_grid;
 width=width/dpi_grid;
 LEFT_MRGN=LEFT_MRGN/dpi_grid;
 RIGHT_MRGN=RIGHT_MRGN/dpi_grid;
 TOP_MRGN=TOP_MRGN/dpi_grid;
```

```
 BOTTOM_MRGN=bottom_mrgn/dpi_grid;
AVG_LINE_H   =  AVG_LINE_H/dpi_grid;
AVG_LINE_SPAC =   AVG_LINE_SPAC/dpi_grid;
end;
Bif='N';
if ( BIF_LOCX1 ne ""or BIF_LOCX2 ne ""  or BIF_LOCY1 ne "" or BIF_LOCY2 <> "" ) then do;
Bif ='Y';
end;
width_written= width - (left_mrgn+ right_mrgn);
height_written=height -(top_mrgn+bottom_mrgn);
RUN;

/*First Split of the dataset*/
/*Creation of an incremental attribute id to be used later as a key in the inner join*/

DATA IMAGES_1 (RENAME=(                    INV_ID=INV_ID_1
                                    IMG_NAME=IMG_NAME_1
                                    FGPI_IMG_NUM=FGPI_IMG_NUM_1
                                    HEIGHT=HEIGHT_1
                                    WIDTH=WIDTH_1
                                       BIF_LOCX1=BIF_LOCX1_1
                                    BIF_LOCX2=BIF_LOCX2_1
                                    BIF_LOCY1=BIF_LOCY1_1
                                    BIF_LOCY2=BIF_LOCY2_1
                                    BIF=BIF_1
                                    FRAG_QTY=FRAG_QTY_1
                                    N_LINES=N_LINES_1
                                    AVG_LINE_H=AVG_LINE_H_1
                                    AVG_LINE_SPAC=AVG_LINE_SPAC_1
                                    LEFT_MRGN=LEFT_MRGN_1
                                    RIGHT_MRGN=RIGHT_MRGN_1
```

```
                                TOP_MRGN=TOP_MRGN_1
                                BOTTOM_MRGN=BOTTOM_MRGN_1
                                width_written = width_written_1
                                height_written = height_written_1
                                COMPLETENESS=COMPLETENESS_1)  );
SET  IMAGES (OBS=120438);
id+1;
RUN;

/*Second Split of the dataset*/
/*Creation of an incremental attribute id to be used later as a key in the inner join*/

DATA  IMAGES_2 (RENAME=(INV_ID=INV_ID_2
                        IMG_NAME=IMG_NAME_2
                        FGPI_IMG_NUM=FGPI_IMG_NUM_2
                        HEIGHT=HEIGHT_2
                        WIDTH=WIDTH_2
                        BIF_LOCX1=BIF_LOCX1_2
                        BIF_LOCX2=BIF_LOCX2_2
                        BIF_LOCY1=BIF_LOCY1_2
                        BIF_LOCY2=BIF_LOCY2_2
                        BIF=BIF_2
                        FRAG_QTY=FRAG_QTY_2
                        N_LINES=N_LINES_2
                        AVG_LINE_H=AVG_LINE_H_2
                        AVG_LINE_SPAC=AVG_LINE_SPAC_2
                        LEFT_MRGN=LEFT_MRGN_2
RIGHT_MRGN=RIGHT_MRGN_2
TOP_MRGN=TOP_MRGN_2
                        BOTTOM_MRGN=BOTTOM_MRGN_2
                        width_written = width_written_2
```

```
                    height_written = height_written_2
                    COMPLETENESS=COMPLETENESS_2));
    SET   IMAGES (FIRSTOBS=120439);
id+1;
RUN;


PROC SORT DATA=  IMAGES_1;
by id;
run;

PROC SORT DATA=  IMAGES_2;
by id;
run;

/*Inner Join between the two splits Key: id*/

data     _MERGE  (drop = id) ;
merge    IMAGES_1
       IMAGES_2    ;
BY ID;
IF INV_ID_1 NE INV_ID_2;
RUN;

/*Computation of the attributes that represent the difference in physical characteristics of the
images*/


DATA MERGE_DIFF;
SET _MERGE;
HEIGHT_DIFF = abs(HEIGHT_1 - HEIGHT_2);
```

```
WIDTH_DIFF = abs(WIDTH_1 - WIDTH_2);
N_LINES_DIFF = abs(N_LINES_1 - N_LINES_2) ;
AVG_LINE_DIFF = abs(AVG_LINE_H_1 - AVG_LINE_H_2);
AVG_LINE_SPAC_DIFF = abs(AVG_LINE_SPAC_1 - AVG_LINE_SPAC_2);
LEFT_MRGN_DIFF = abs(LEFT_MRGN_1 - LEFT_MRGN_2);
RIGHT_MRGN_DIFF = abs(RIGHT_MRGN_1 - RIGHT_MRGN_2);
TOP_MRGN_DIFF = abs(TOP_MRGN_1 - TOP_MRGN_2);
BOTTOM_MRGN_DIFF = abs( BOTTOM_MRGN_1-BOTTOM_MRGN_2);
WIDTH_WRITTEN_DIFF=abs(width_written_1 - width_written_2);
HEIGHT_WRITTEN_DIFF=abs( height_written_1 -  height_written_2);
COMPLETENESS_CALC =( COMPLETENESS_1 * COMPLETENESS_2);
RUN;

/*Final dataset representing “non join”*/
/*creation of the attribute JOIN”*/


DATA  IMAGE_DIFF_JOIN;
SET   MERGE_DIFF  ;
JOIN = 'NON_JOIN';
if inv_id_1 ne '';
RUN;

/* Write a dataset reading from the input file containing all true joins */

DATA  IMPORT_JOINS;
INFILE  "C:\Documents  and  Settings\Administrator\Desktop\Tesi  Images\DATI\JOINS.TXT"
DELIMITER='09'X DSD;
INPUT JOIN_ID
INV_ID  : $10.  ;
RUN;
```

```
proc sort data= import_joins out=  imp_join_srtd;
by join_id;
run;

/*first copy of  IMPORT_JOINS*/

data imp_join_srtd1 (rename=inv_id=inv_id_1);
set  imp_join_srtd;
run;

/*second copy of  IMPORT_JOINS*/

data  imp_join_srtd2 (rename=inv_id=inv_id_2) ;
set imp_join_srtd;
run;

/*inner join between the two copy Key: JOIN_ID*/

proc sql;
create table joins as
select * from imp_join_srtd1 m1, imp_join_srtd2 m2
where   m1.join_id=m2.join_id;
quit;


proc sort data= joins  out= joins_srtd;
by join_id inv_id_1 inv_id_2;
run;
```

```
/*delete duplicated information*/

data fine;
set joins_srtd;
if (inv_id_1 ne inv_id_2) and (inv_id_2>inv_id_1);
run;


proc sort data= fine out= fine_ndk nodupkey ;
by inv_id_1 inv_id_2 ;
run;


proc sort data=  fine_ndk;
by join_id  inv_id_1 inv_id_2;
run;

data im1 (rename =(
INV_ID=INV_ID_1
                                IMG_NAME=IMG_NAME_1
                                FGPI_IMG_NUM=FGPI_IMG_NUM_1
                                HEIGHT=HEIGHT_1
                                WIDTH=WIDTH_1
                                        BIF_LOCX1=BIF_LOCX1_1
                                BIF_LOCX2=BIF_LOCX2_1
                                BIF_LOCY1=BIF_LOCY1_1
                                BIF_LOCY2=BIF_LOCY2_1
                                BIF=BIF_1
                                FRAG_QTY=FRAG_QTY_1
                                N_LINES=N_LINES_1
                                AVG_LINE_H=AVG_LINE_H_1
```

```
                                AVG_LINE_SPAC=AVG_LINE_SPAC_1
                                LEFT_MRGN=LEFT_MRGN_1
                                RIGHT_MRGN=RIGHT_MRGN_1
                                TOP_MRGN=TOP_MRGN_1
                                BOTTOM_MRGN=BOTTOM_MRGN_1
                                width_written = width_written_1
                                height_written = height_written_1
                                COMPLETENESS=COMPLETENESS_1));

set images;
run;


proc sort data= im1   ;
by inv_id_1;
run;


proc sort data=  fine_ndk out= fine_ndk_1;
by inv_id_1;
run;

/*retrieve all information relative to the first INV_ID */


data   fine_ndk1;
merge im1 (in=im1)
      fine_ndk_1 (in=fine);
by inv_id_1;
if im1 =1 and fine=1;
run;
```

```
data im2 (rename =(
INV_ID=INV_ID_2
                                IMG_NAME=IMG_NAME_2
                                FGPI_IMG_NUM=FGPI_IMG_NUM_2
                                HEIGHT=HEIGHT_2
                                WIDTH=WIDTH_2
                                 BIF_LOCX1=BIF_LOCX1_2
                                BIF_LOCX2=BIF_LOCX2_2
                                BIF_LOCY1=BIF_LOCY1_2
                                BIF_LOCY2=BIF_LOCY2_2
                                BIF=BIF_2
                                FRAG_QTY=FRAG_QTY_2
                                N_LINES=N_LINES_2
                                AVG_LINE_H=AVG_LINE_H_2
                                AVG_LINE_SPAC=AVG_LINE_SPAC_2
                                LEFT_MRGN=LEFT_MRGN_2
                                RIGHT_MRGN=RIGHT_MRGN_2
                                TOP_MRGN=TOP_MRGN_2
                                BOTTOM_MRGN=BOTTOM_MRGN_2
                                width_written = width_written_2
                                height_written = height_written_2
                                COMPLETENESS=COMPLETENESS_2));

set images;
run;


proc sort data= im2   ;
by inv_id_2;
run;
```

```
proc sort data=  fine_ndk out= fine_ndk_2;
by inv_id_2;
run;

/*retrieve all information relative to the second INV_ID */

data   fine_ndk2;
merge im2 (in=im2)
      fine_ndk_2 (in=fine);
by inv_id_2;
if im2 =1 and fine=1;
run;



proc sort data=fine_ndk2    ;
by join_id;
run;

proc sort data=fine_ndk1    ;
by join_id;
run;


data finefine;
merge fine_ndk1 (in=f1)
fine_ndk2  (in=f2);
by join_id;
if f1 and f2;
run;
```

```
/* Computation of the attributes that represent the difference in physical characteristics of the images */
/*Set JOIN=”join”*/

data  joins_diff (drop=join_id);
set  finefine;
HEIGHT_DIFF = abs(HEIGHT_1 - HEIGHT_2);
WIDTH_DIFF = abs(WIDTH_1 - WIDTH_2);
N_LINES_DIFF = abs(N_LINES_1 - N_LINES_2) ;
AVG_LINE_DIFF = abs(AVG_LINE_H_1 - AVG_LINE_H_2);
AVG_LINE_SPAC_DIFF = abs(AVG_LINE_SPAC_1 - AVG_LINE_SPAC_2);
LEFT_MRGN_DIFF = abs(LEFT_MRGN_1 - LEFT_MRGN_2);
RIGHT_MRGN_DIFF = abs(RIGHT_MRGN_1 - RIGHT_MRGN_2);
TOP_MRGN_DIFF = abs(TOP_MRGN_1 - TOP_MRGN_2);
BOTTOM_MRGN_DIFF = abs(BOTTOM_MRGN_1-BOTTOM_MRGN_2);
COMPLETENESS_CALC = (COMPLETENESS_1 * COMPLETENESS_2);
WIDTH_WRITTEN_DIFF=abs(width_written_1 - width_written_2);
HEIGHT_WRITTEN_DIFF=abs( height_written_1 -  height_written_2);
JOIN='JOIN';
run;

/*Write a dataset with all “non join” and “join”*/

data dataset;
set  IMAGE_DIFF_JOIN
    joins_diff  ;
run;

/*compute the concatenation of the quality descriptions and set BIFOLIO=1 if both or none of the images are bifolio, 0 otherwise*/
```

```
data dataset2;
set dataset;
length   QTY_CONC  $90.;
QTY_CONC= frag_qty_1||"-"||frag_qty_2;
if bif_1 ='Y' and bif_2='Y' then do;
        bif=1;
end;
if (bif_1 ='Y' and bif_2='N') or (bif_1 ='N' and bif_2='Y') then do;
        bif= 0;
end;
if bif_1 ='N' and bif_2='N' then do;
        bif=1;
end;
run;

data dataset2;
set dataset2;
QTY_CONC =compress(QTY_CONC) ;
run;

/* set QTY_CONC=”a-b” = QTY_CONC=”b-a” */

data libreria.dataset2;
set dataset2;
if QTY_CONC='-Empty' then QTY_CONC ='Empty-';
if QTY_CONC='-Noisy' then QTY_CONC ='Noisy-';
if QTY_CONC='-NotText' then QTY_CONC ='NotText-';
if QTY_CONC='-Problematic' then QTY_CONC ='Problematic-';
if QTY_CONC='-SuspectedProblemtic' then QTY_CONC ='SuspectedProblemtic-';
if QTY_CONC='-good' then QTY_CONC ='good-';
```

if QTY_CONC='-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-';

if QTY_CONC='-suspectedEmpty' then QTY_CONC ='suspectedEmpty-';

if QTY_CONC='Empty-Noisy' then QTY_CONC ='Noisy-Empty';

if QTY_CONC='Empty-NotText' then QTY_CONC ='NotText-Empty';

if QTY_CONC='Empty-Problematic' then QTY_CONC ='Problematic-Empty';

if QTY_CONC='Empty-SuspectedProblemtic' then QTY_CONC ='SuspectedProblemtic-Empty';

if QTY_CONC='Empty-good' then QTY_CONC ='good-Empty';

if QTY_CONC='Empty-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-Empty';

if QTY_CONC='Empty-suspectedEmpty' then QTY_CONC ='suspectedEmpty-Empty';

if QTY_CONC='Noisy-NotText' then QTY_CONC ='NotText-Noisy';

if QTY_CONC='Noisy-Problematic' then QTY_CONC ='Problematic-Noisy';

if QTY_CONC='Noisy-SuspectedProblemtic' then QTY_CONC ='SuspectedProblemtic-Noisy';

if QTY_CONC='Noisy-good' then QTY_CONC ='good-Noisy';

if QTY_CONC='Noisy-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-Noisy';

if QTY_CONC='Noisy-suspectedEmpty' then QTY_CONC ='suspectedEmpty-Noisy';

if QTY_CONC='NotText-Problematic' then QTY_CONC ='Problematic-NotText';

if QTY_CONC='NotText-Suspected Problemtic' then QTY_CONC ='SuspectedProblemtic-NotText';

if QTY_CONC='NotText-good' then QTY_CONC ='good-NotText';

if QTY_CONC='NotText-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-NotText';

if QTY_CONC='NotText-suspectedEmpty' then QTY_CONC ='suspectedEmpty-NotText';

if QTY_CONC='Problematic-SuspectedProblemtic' then QTY_CONC ='SuspectedProblemtic-Problematic';

if QTY_CONC='Problematic-good' then QTY_CONC ='good-Problematic';

if QTY_CONC='Problematic-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-Problematic';

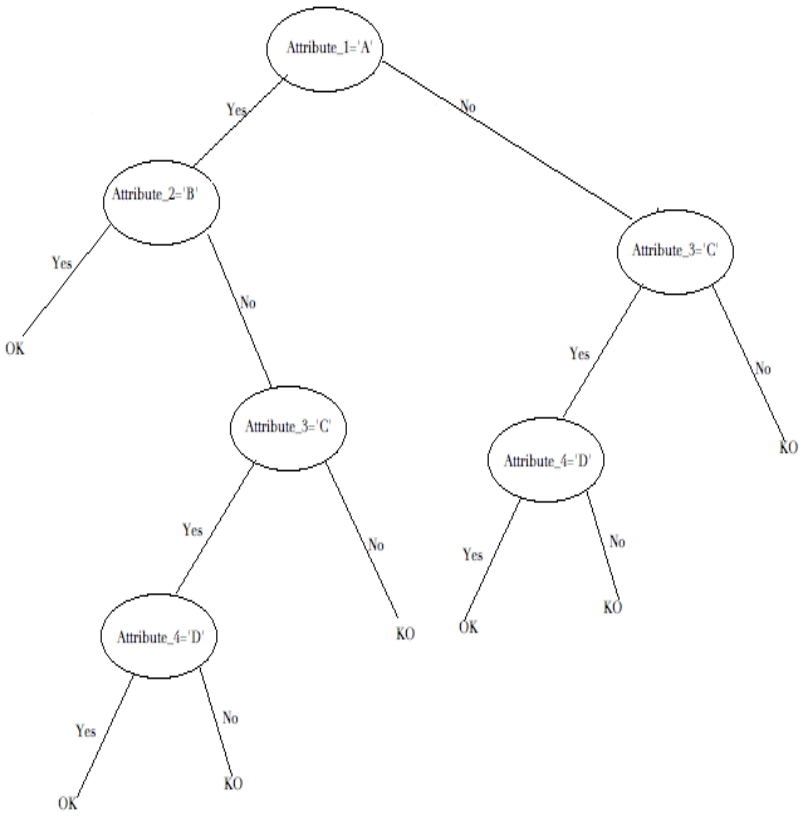
if QTY_CONC='Problematic -suspected Empty' then QTY_CONC ='suspectedEmpty-Problematic';

```
if QTY_CONC='SuspectedProblemtic-good' then QTY_CONC ='good-SuspectedProblemtic';
if QTY_CONC='SuspectedProblemtic-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-SuspectedProblemtic';
if QTY_CONC='SuspectedProblemtic-suspectedEmpty' then QTY_CONC ='suspectedEmpty-SuspectedProblemtic';
if QTY_CONC='SuspectedProblemtic-suspectedNoisy' then QTY_CONC ='suspectedNoisy-SuspectedProblemtic';
if QTY_CONC='good-stdRowsHeightlessthan5' then QTY_CONC ='stdRowsHeightlessthan5-good';
if QTY_CONC='good-suspectedEmpty' then QTY_CONC ='suspectedEmpty-good';
if QTY_CONC='stdRowsHeightlessthan5-suspectedEmpty' then QTY_CONC ='suspectedEmpty-stdRowsHeightlessthan 5';
run;
```

Appendix B

What follows is an example of the problem replicated subtree. The decision tree needed to represent the simple rule:

$if\ \big((Attribute_1 = 'A'\ and\ Attribute_2 = 'B')\ OR\ (Attribute_3 = 'C'\ and\ Attribute_4 = 'D'\ )\big) then\ class = 'OK'$otherwise class = 'KO'



it has 13 nodes and seven leaves. It can be seen that the subtree with root $\text{Attribute}_3 = 'C'$ is replicated.

Appendix C

This appendix reports rules learnt by RIPPER rule–inducer when no feature selection is applied and when CFS correlation–based multivariate filter is applied. The numbers in the parenthesis represent correctly classified instances/ incorrectly classified instances in the validation set.

RIPPER without CFS

(WIDTH_DIFF <= 0.449075) and (AVG_LINE_SPAC_DIFF <= 0.048739) and (HEIGHT_DIFF <= 0.302982) and (WIDTH_DIFF <= 0.126667) => JOIN=JOIN (1309.0/24.0)

(WIDTH_DIFF <= 1.03937) and (COMPLETENESS_Measure >= 0.6975) and (AVG_LINE_SPAC_DIFF <= 0.048739) and (width_written_diff <= 0.484271) and (WIDTH_DIFF <= 0.325382) and (AVG_LINE_DIFF <= 0.021957) => JOIN=JOIN (541.0/22.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 1.390411) and (COMPLETENESS_Measure >= 0.7189) and (width_written_diff <= 0.535088) and (AVG_LINE_SPAC_DIFF <= 0.031496) and (HEIGHT_DIFF <= 0.603349) and (WIDTH_DIFF <= 0.325382) and (width_written_diff <= 0.300336) => JOIN=JOIN (32.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.102427) and (WIDTH_DIFF <= 1.432432) and (COMPLETENESS_Measure >= 0.7426) and (width_written_diff <= 0.545675) and (AVG_LINE_SPAC_DIFF <= 0.046452) and (height_written_diff <= 0.692668) and (WIDTH_DIFF >= 0.872966) and (RIGHT_MRGN_DIFF <= 1.046409) => JOIN=JOIN (143.0/2.0)

(WIDTH_DIFF <= 1.03937) and (COMPLETENESS_Measure >= 0.7396) and (LEFT_MRGN_DIFF <= 0.662341) and (COMPLETENESS_Measure >= 0.8544) and (AVG_LINE_SPAC_DIFF <= 0.153509) => JOIN=JOIN (708.0/169.0)

(AVG_LINE_SPAC_DIFF <= 0.086667) and (WIDTH_DIFF <= 1.073068) and (COMPLETENESS_Measure >= 0.7426) and (LEFT_MRGN_DIFF <= 0.662341) and

(height_written_diff <= 0.824561) and (WIDTH_DIFF <= 0.679288) and (AVG_LINE_DIFF <= 0.023661) => JOIN=JOIN (108.0/18.0)

(AVG_LINE_DIFF <= 0.047381) and (WIDTH_DIFF <= 0.988333) and (COMPLETENESS_Measure >= 0.7426) and (width_written_diff <= 0.491228) and (TOP_MRGN_DIFF >= 0.472752) and (Qty_1_2 = stdRowsHeightlessthan5-good) and (COMPLETENESS_Measure >= 0.7921) => JOIN=JOIN (45.0/2.0)

(AVG_LINE_DIFF <= 0.031798) and (WIDTH_DIFF <= 0.976378) and (AVG_LINE_SPAC_DIFF <= 0.022222) and (width_written_diff <= 0.483221) and (HEIGHT_DIFF <= 0.850394) and (RIGHT_MRGN_DIFF <= 0.362416) and (COMPLETENESS_Measure >= 0.513) => JOIN=JOIN (99.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.079456) and (COMPLETENESS_Measure >= 0.735) and (LEFT_MRGN_DIFF <= 0.653509) and (COMPLETENESS_Measure >= 0.8188) and (COMPLETENESS_Measure <= 0.8256) and (BOTTOM_MRGN_DIFF <= 0.614173) => JOIN=JOIN (59.0/0.0)

(AVG_LINE_DIFF <= 0.031496) and (WIDTH_DIFF <= 0.961901) and (AVG_LINE_SPAC_DIFF <= 0.033899) and (HEIGHT_DIFF <= 1.190789) and (width_written_diff <= 0.408163) and (AVG_LINE_DIFF <= 0.009252) and (width_written_diff <= 0.207211) => JOIN=JOIN (66.0/11.0)

(AVG_LINE_SPAC_DIFF <= 0.094674) and (WIDTH_DIFF <= 1.484649) and (COMPLETENESS_Measure >= 0.6612) and (RIGHT_MRGN_DIFF <= 0.403509) and (BOTTOM_MRGN_DIFF <= 0.264715) and (RIGHT_MRGN_DIFF <= 0.1596) and (COMPLETENESS_Measure <= 0.7826) => JOIN=JOIN (116.0/25.0)

(AVG_LINE_DIFF <= 0.028198) and (AVG_LINE_SPAC_DIFF <= 0.023622) and (width_written_diff <= 0.789035) and (WIDTH_DIFF <= 1.079449) and (width_written_diff <= 0.483333) and (COMPLETENESS_Measure >= 0.5694) and (HEIGHT_DIFF >= 1.408163) and (height_written_diff <= 2.294335) and (AVG_LINE_SPAC_DIFF <= 0.013605) => JOIN=JOIN (41.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.086667) and (AVG_LINE_DIFF <= 0.027888) and (HEIGHT_DIFF <= 1.040816) and (width_written_diff <= 0.500089) and (AVG_LINE_SPAC_DIFF <= 0.033333) and (Qty_1_2 = good-good) and (AVG_LINE_DIFF <= 0.010526) => JOIN=JOIN (176.0/49.0)

(AVG_LINE_DIFF <= 0.047381) and (WIDTH_DIFF <= 1.484649) and (COMPLETENESS_Measure >= 0.7189) and (LEFT_MRGN_DIFF <= 0.252344) and (width_written_diff <= 0.73775) and (HEIGHT_DIFF >= 1.462585) and (RIGHT_MRGN_DIFF >= 0.668027) => JOIN=JOIN (73.0/5.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (AVG_LINE_DIFF <= 0.027888) and (RIGHT_MRGN_DIFF <= 0.44) and (width_written_diff <= 0.844595) and (width_written_diff <= 0.129649) and (height_written_diff <= 0.207667) => JOIN=JOIN (37.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (AVG_LINE_DIFF <= 0.028025) and (HEIGHT_DIFF <= 0.761031) and (width_written_diff <= 0.5) and (BOTTOM_MRGN_DIFF <= 0.186667) and (width_written_diff <= 0.218158) and (LEFT_MRGN_DIFF <= 0.606667) and (TOP_MRGN_DIFF >= 0.023265) => JOIN=JOIN (25.0/4.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (AVG_LINE_DIFF <= 0.028025) and (RIGHT_MRGN_DIFF <= 0.450131) and (width_written_diff <= 0.826667) and (AVG_LINE_SPAC_DIFF <= 0.007075) and (AVG_LINE_DIFF <= 0.006803) and (AVG_LINE_DIFF >= 0.006579) => JOIN=JOIN (27.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (AVG_LINE_DIFF <= 0.028025) and (RIGHT_MRGN_DIFF <= 0.44) and (width_written_diff <= 0.71) and (width_written_diff <= 0.126443) and (BOTTOM_MRGN_DIFF <= 0.384777) and (COMPLETENESS_Measure >= 0.7176) and (TOP_MRGN_DIFF >= 0.373018) => JOIN=JOIN (27.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.086667) and (AVG_LINE_DIFF <= 0.028025) and (RIGHT_MRGN_DIFF <= 0.645722) and (width_written_diff <= 0.789035) and (AVG_LINE_SPAC_DIFF <= 0.025591) and (WIDTH_DIFF <= 0.23) and (RIGHT_MRGN_DIFF >= 0.388158) and (BOTTOM_MRGN_DIFF >= 0.366332) => JOIN=JOIN (22.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (AVG_LINE_DIFF <= 0.028018) and (RIGHT_MRGN_DIFF <= 0.44) and (LEFT_MRGN_DIFF <= 1.307087) and (COMPLETENESS_Measure >= 0.6816) and (AVG_LINE_SPAC_DIFF >= 0.046667) and (width_written_diff >= 1.270843) and (HEIGHT_DIFF <= 2.912281) => JOIN=JOIN (131.0/22.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (WIDTH_DIFF <= 2.002272) and (AVG_LINE_DIFF <= 0.042949) and (width_written_diff <= 1.505051) and (AVG_LINE_SPAC_DIFF <= 0.034014) and (Qty_1_2 = good-good) and (LEFT_MRGN_DIFF <= 0.373333) and (height_written_diff <= 0.445324) and (COMPLETENESS_Measure <= 0.68) => JOIN=JOIN (31.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.094595) and (AVG_LINE_DIFF <= 0.028018) and (RIGHT_MRGN_DIFF <= 0.446667) and (width_written_diff <= 1.503937) and (LEFT_MRGN_DIFF <= 1.36) and (AVG_LINE_DIFF >= 0.019691) and (WIDTH_DIFF >= 1.150555) and (BOTTOM_MRGN_DIFF >= 0.90604) => JOIN=JOIN (54.0/9.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 1.913043) and (COMPLETENESS_Measure >= 0.8188) and (LEFT_MRGN_DIFF <= 0.415187) and (height_written_diff >= 1.499657) and (width_written_diff <= 1.412626) and (width_written_diff >= 0.877193) and (LEFT_MRGN_DIFF >= 0.14182) => JOIN=JOIN (58.0/8.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (AVG_LINE_DIFF <= 0.027517) and (HEIGHT_DIFF <= 1.328859) and (width_written_diff <= 0.440945) and (LEFT_MRGN_DIFF >= 5.106667) and (AVG_LINE_SPAC_DIFF <= 0.041339) and (TOP_MRGN_DIFF <= 0.42) => JOIN=JOIN (63.0/15.0)

(AVG_LINE_SPAC_DIFF <= 0.095614) and (AVG_LINE_DIFF <= 0.027524) and (HEIGHT_DIFF <= 1.047619) and (WIDTH_DIFF <= 0.461923) and (AVG_LINE_SPAC_DIFF <= 0.020408) and (TOP_MRGN_DIFF <= 0.075298) => JOIN=JOIN (38.0/8.0)

(AVG_LINE_SPAC_DIFF <= 0.095614) and (WIDTH_DIFF <= 2.462998) and (COMPLETENESS_Measure >= 0.744) and (COMPLETENESS_Measure >= 0.874) and (HEIGHT_DIFF >= 2.839912) and (N_LINES_DIFF <= 3) and (LEFT_MRGN_DIFF <=

0.984649) => JOIN=JOIN (52.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.094488) and (AVG_LINE_DIFF <= 0.023663) and (width_written_diff <= 0.834343) and (AVG_LINE_SPAC_DIFF <= 0.013605) and (TOP_MRGN_DIFF <= 0.206667) and (BOTTOM_MRGN_DIFF >= 1.160191) and (height_written_diff <= 0.263078) => JOIN=JOIN (21.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.094595) and (AVG_LINE_DIFF <= 0.024123) and (width_written_diff <= 0.832364) and (AVG_LINE_SPAC_DIFF <= 0.022222) and (WIDTH_DIFF >= 5.558559) and (width_written_diff <= 0.063684) and (LEFT_MRGN_DIFF <= 0.415436) => JOIN=JOIN (19.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.095581) and (AVG_LINE_DIFF <= 0.047619) and (HEIGHT_DIFF <= 1.775175) and (WIDTH_DIFF <= 1.355591) and (width_written_diff <= 0.802162) and (HEIGHT_DIFF >= 1.46) and (AVG_LINE_SPAC_DIFF >= 0.07809) and (AVG_LINE_SPAC_DIFF <= 0.08) => JOIN=JOIN (28.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.095614) and (WIDTH_DIFF <= 2.040587) and (HEIGHT_DIFF <= 0.666667) and (WIDTH_DIFF <= 0.452381) and (height_written_diff <= 0.387755) and (BOTTOM_MRGN_DIFF <= 0.055) => JOIN=JOIN (53.0/18.0)

(AVG_LINE_DIFF <= 0.047619) and (AVG_LINE_SPAC_DIFF <= 0.086667) and (WIDTH_DIFF <= 2.251969) and (width_written_diff <= 1.088435) and (RIGHT_MRGN_DIFF <= 0.382713) and (HEIGHT_DIFF <= 1.017695) and (height_written_diff >= 2.095008) and (N_LINES_DIFF <= 6) => JOIN=JOIN (26.0/3.0)

(AVG_LINE_DIFF <= 0.047619) and (AVG_LINE_SPAC_DIFF <= 0.086667) and (HEIGHT_DIFF <= 1.32) and (AVG_LINE_DIFF <= 0) => JOIN=JOIN (51.0/14.0)

(AVG_LINE_SPAC_DIFF <= 0.120157) and (AVG_LINE_DIFF <= 0.047619) and (LEFT_MRGN_DIFF <= 0.466216) and (width_written_diff <= 0.741361) and (RIGHT_MRGN_DIFF <= 0.19) and (height_written_diff <= 0.633333) and (TOP_MRGN_DIFF >= 0.538246) and (WIDTH_DIFF >= 0.413333) => JOIN=JOIN (21.0/1.0)

=> JOIN=NON_JOIN (62440.0/2832.0)

RIPPER with CFS

(WIDTH_DIFF <= 0.449075) and (AVG_LINE_SPAC_DIFF <= 0.048739) and (HEIGHT_DIFF <= 0.3) and (width_written_diff <= 0.242424) and (RIGHT_MRGN_DIFF <= 0.110236) => JOIN=JOIN (1039.0/2.0)

(WIDTH_DIFF <= 0.986518) and (AVG_LINE_SPAC_DIFF <= 0.048894) and (width_written_diff <= 0.535354) and (WIDTH_DIFF <= 0.255207) and (AVG_LINE_DIFF <= 0.023661) and (HEIGHT_DIFF <= 1.90179) => JOIN=JOIN (762.0/38.0)

(WIDTH_DIFF <= 1.342145) and (AVG_LINE_SPAC_DIFF <= 0.057018) and (width_written_diff <= 0.535354) and (AVG_LINE_DIFF <= 0.024009) and (RIGHT_MRGN_DIFF <= 0.36) and (RIGHT_MRGN_DIFF >= 0.163676) and (TOP_MRGN_DIFF <= 0.426667) => JOIN=JOIN (270.0/29.0)

(WIDTH_DIFF <= 1.079456) and (AVG_LINE_SPAC_DIFF <= 0.047244) and (HEIGHT_DIFF <= 0.868421) and (width_written_diff <= 0.528131) and (AVG_LINE_SPAC_DIFF <= 0.024547) and (RIGHT_MRGN_DIFF <= 0.30303) and (AVG_LINE_DIFF <= 0.021858) and (RIGHT_MRGN_DIFF <= 0.07931) => JOIN=JOIN (47.0/2.0)

(WIDTH_DIFF <= 0.99542) and (AVG_LINE_SPAC_DIFF <= 0.061773) and (HEIGHT_DIFF <= 0.925439) and (AVG_LINE_SPAC_DIFF <= 0.02193) and (HEIGHT_DIFF <= 0.103509) => JOIN=JOIN (154.0/21.0)

(WIDTH_DIFF <= 1.03937) and (AVG_LINE_SPAC_DIFF <= 0.102427) and (width_written_diff <= 1.080062) and (AVG_LINE_SPAC_DIFF <= 0.022544) and (AVG_LINE_DIFF <= 0.031496) and (TOP_MRGN_DIFF <= 0.160465) and (TOP_MRGN_DIFF >= 0.127193) and (width_written_diff >= 0.243714) => JOIN=JOIN (40.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.425197) and (width_written_diff <= 1.145695) and (height_written_diff <= 0.824561) and (AVG_LINE_SPAC_DIFF <= 0.024631) and (AVG_LINE_DIFF <= 0.020408) and

(WIDTH_DIFF <= 0.874016) and (height_written_diff >= 0.195946) and (AVG_LINE_SPAC_DIFF <= 0.007162) and (width_written_diff >= 0.401007) => JOIN=JOIN (66.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.079449) and (width_written_diff <= 0.535354) and (height_written_diff <= 1.692982) and (AVG_LINE_SPAC_DIFF <= 0.028864) and (AVG_LINE_DIFF <= 0.027211) and (width_written_diff <= 0.17411) and (TOP_MRGN_DIFF >= 0.291391) => JOIN=JOIN (47.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.438596) and (width_written_diff <= 1.081915) and (WIDTH_DIFF <= 0.449075) and (HEIGHT_DIFF <= 0.816316) and (BOTTOM_MRGN_DIFF <= 0.164502) and (RIGHT_MRGN_DIFF <= 0.298246) and (HEIGHT_DIFF >= 0.662281) => JOIN=JOIN (51.0/2.0)

(AVG_LINE_DIFF <= 0.047381) and (WIDTH_DIFF <= 1.484649) and (width_written_diff <= 0.797281) and (AVG_LINE_SPAC_DIFF <= 0.125984) and (RIGHT_MRGN_DIFF <= 0.382713) and (BOTTOM_MRGN_DIFF <= 0.337314) and (BOTTOM_MRGN_DIFF >= 0.225926) and (AVG_LINE_DIFF >= 0.020101) and (LEFT_MRGN_DIFF >= 0.341732) and (TOP_MRGN_DIFF >= 0.102362) => JOIN=JOIN (70.0/5.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.438596) and (width_written_diff <= 0.987523) and (LEFT_MRGN_DIFF <= 0.506667) and (height_written_diff <= 0.785088) and (width_written_diff <= 0.499487) and (AVG_LINE_SPAC_DIFF <= 0.055118) and (height_written_diff <= 0.312299) and (RIGHT_MRGN_DIFF >= 0.8415) and (AVG_LINE_SPAC_DIFF >= 0.031149) => JOIN=JOIN (40.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.445175) and (width_written_diff <= 0.987523) and (AVG_LINE_DIFF <= 0.04386) and (WIDTH_DIFF <= 0.266803) and (AVG_LINE_SPAC_DIFF <= 0.020408) and (AVG_LINE_DIFF <= 0.008246) and (RIGHT_MRGN_DIFF <= 0.342105) => JOIN=JOIN (39.0/5.0)

(AVG_LINE_SPAC_DIFF <= 0.102427) and (WIDTH_DIFF <= 1.484649) and

(width_written_diff <= 1.342105) and (AVG_LINE_DIFF <= 0.04386) and (LEFT_MRGN_DIFF <= 0.503937) and (height_written_diff <= 0.791565) and (TOP_MRGN_DIFF >= 0.496063) and (width_written_diff <= 0.401575) => JOIN=JOIN (81.0/21.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.484649) and (AVG_LINE_DIFF <= 0.04386) and (LEFT_MRGN_DIFF <= 0.533608) and (WIDTH_DIFF <= 0.473333) and (height_written_diff <= 0.300493) and (height_written_diff >= 0.204082) and (AVG_LINE_SPAC_DIFF <= 0.026667) => JOIN=JOIN (27.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.445175) and (width_written_diff <= 1.086614) and (AVG_LINE_DIFF <= 0.04386) and (AVG_LINE_SPAC_DIFF <= 0.02193) and (RIGHT_MRGN_DIFF <= 0.198246) and (BOTTOM_MRGN_DIFF <= 0.148238) and (LEFT_MRGN_DIFF <= 0.211936) => JOIN=JOIN (39.0/7.0)

(AVG_LINE_SPAC_DIFF <= 0.102427) and (WIDTH_DIFF <= 1.484649) and (AVG_LINE_DIFF <= 0.04386) and (width_written_diff <= 1.342105) and (RIGHT_MRGN_DIFF <= 0.403509) and (AVG_LINE_SPAC_DIFF <= 0.013605) and (LEFT_MRGN_DIFF >= 0.527014) and (WIDTH_DIFF <= 0.283784) => JOIN=JOIN (27.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.118687) and (WIDTH_DIFF <= 1.966341) and (LEFT_MRGN_DIFF <= 0.533608) and (AVG_LINE_DIFF <= 0.040802) and (width_written_diff <= 1.561297) and (AVG_LINE_DIFF >= 0.03937) and (AVG_LINE_DIFF <= 0.039474) => JOIN=JOIN (54.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.438596) and (width_written_diff <= 0.797281) and (AVG_LINE_DIFF <= 0.043773) and (width_written_diff <= 0.133858) and (AVG_LINE_SPAC_DIFF >= 0.092715) and (AVG_LINE_DIFF <= 0.015853) and (WIDTH_DIFF >= 0.065459) => JOIN=JOIN (40.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.095713) and (WIDTH_DIFF <= 1.445092) and (AVG_LINE_DIFF <= 0.031717) and (LEFT_MRGN_DIFF <= 0.404514) and (height_written_diff <= 0.756023) and (WIDTH_DIFF <= 0.633555) and (TOP_MRGN_DIFF <= 0.085776) and (height_written_diff <= 0.109506) => JOIN=JOIN (27.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.094674) and (width_written_diff <= 0.756581) and (AVG_LINE_DIFF <= 0.031496) and (AVG_LINE_SPAC_DIFF <= 0.047769) and (height_written_diff <= 0.548662) and (TOP_MRGN_DIFF <= 0.246667) and (width_written_diff <= 0.210884) and (height_written_diff <= 0.204595) => JOIN=JOIN (84.0/13.0)

(AVG_LINE_SPAC_DIFF <= 0.094667) and (WIDTH_DIFF <= 1.965757) and (AVG_LINE_DIFF <= 0.048449) and (width_written_diff <= 1.577218) and (LEFT_MRGN_DIFF <= 0.364569) and (WIDTH_DIFF <= 0.266803) and (RIGHT_MRGN_DIFF >= 0.637795) and (width_written_diff <= 0.642544) and (LEFT_MRGN_DIFF <= 0.204724) => JOIN=JOIN (33.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.095713) and (WIDTH_DIFF <= 1.445092) and (AVG_LINE_DIFF <= 0.048246) and (height_written_diff <= 0.609649) and (WIDTH_DIFF <= 0.629921) and (BOTTOM_MRGN_DIFF <= 0.326531) and (AVG_LINE_SPAC_DIFF <= 0.04069) => JOIN=JOIN (103.0/39.0)

(AVG_LINE_SPAC_DIFF <= 0.095713) and (AVG_LINE_DIFF <= 0.031496) and (RIGHT_MRGN_DIFF <= 0.436298) and (LEFT_MRGN_DIFF <= 0.735433) and (width_written_diff <= 1.559055) and (WIDTH_DIFF >= 0.986663) and (AVG_LINE_DIFF >= 0.015748) and (BOTTOM_MRGN_DIFF >= 0.878596) => JOIN=JOIN (63.0/11.0)

(AVG_LINE_SPAC_DIFF <= 0.12) and (AVG_LINE_DIFF <= 0.031496) and (width_written_diff <= 0.747427) and (AVG_LINE_SPAC_DIFF <= 0.022222) and (WIDTH_DIFF >= 5.251078) and (height_written_diff <= 0.42517) and (WIDTH_DIFF <= 7.118403) => JOIN=JOIN (92.0/24.0)

(AVG_LINE_SPAC_DIFF <= 0.095713) and (AVG_LINE_DIFF <= 0.031496) and (HEIGHT_DIFF <= 1.328859) and (width_written_diff <= 0.680272) and (AVG_LINE_SPAC_DIFF <= 0.048661) and (WIDTH_DIFF >= 4.98) and (TOP_MRGN_DIFF <= 0.308725) and (width_written_diff <= 0.41844) and (LEFT_MRGN_DIFF >= 5.48759) => JOIN=JOIN (44.0/8.0)

(AVG_LINE_SPAC_DIFF <= 0.095713) and (WIDTH_DIFF <= 2.562041) and (AVG_LINE_DIFF <= 0.047381) and (LEFT_MRGN_DIFF <= 0.533608) and (RIGHT_MRGN_DIFF <= 0.432021) and (AVG_LINE_SPAC_DIFF >= 0.043155) and (TOP_MRGN_DIFF <= 0.421809) and (width_written_diff >= 1.270843) and (RIGHT_MRGN_DIFF >= 0.296053) and (RIGHT_MRGN_DIFF >= 0.417323) => JOIN=JOIN (33.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.118687) and (WIDTH_DIFF <= 1.944882) and (LEFT_MRGN_DIFF <= 0.364569) and (width_written_diff <= 0.771654) and (AVG_LINE_SPAC_DIFF >= 0.11811) and (WIDTH_DIFF >= 0.755906) => JOIN=JOIN (48.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 1.938272) and (width_written_diff <= 1.342105) and (AVG_LINE_DIFF <= 0.042949) and (RIGHT_MRGN_DIFF <= 0.403509) and (width_written_diff >= 1.270843) and (RIGHT_MRGN_DIFF >= 0.209459) and (LEFT_MRGN_DIFF <= 0.508074) => JOIN=JOIN (43.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 1.966341) and (RIGHT_MRGN_DIFF <= 0.254386) and (HEIGHT_DIFF <= 0.496063) and (WIDTH_DIFF <= 0.391662) and (RIGHT_MRGN_DIFF >= 0.12214) => JOIN=JOIN (92.0/30.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (WIDTH_DIFF <= 2.562041) and (AVG_LINE_DIFF <= 0.047381) and (width_written_diff <= 1.577218) and (LEFT_MRGN_DIFF <= 0.221141) and (HEIGHT_DIFF <= 1.952341) and (HEIGHT_DIFF >= 1.486961) and (AVG_LINE_SPAC_DIFF >= 0.044523) and (width_written_diff <= 0.88189) and (AVG_LINE_DIFF >= 0.020101) => JOIN=JOIN (43.0/2.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 2.040587) and (AVG_LINE_DIFF <= 0.042949) and (RIGHT_MRGN_DIFF <= 0.257687) and (height_written_diff <= 3.14) and (TOP_MRGN_DIFF <= 0.236175) and (AVG_LINE_DIFF >= 0.019121) and (BOTTOM_MRGN_DIFF <= 0.270515) and (BOTTOM_MRGN_DIFF >= 0.190498) => JOIN=JOIN (40.0/8.0)

(AVG_LINE_SPAC_DIFF <= 0.082676) and (AVG_LINE_DIFF <= 0.031496) and (HEIGHT_DIFF <= 1.303134) and (RIGHT_MRGN_DIFF <= 0.432407) and (width_written_diff <= 0.492162) and (BOTTOM_MRGN_DIFF <= 0.138983) and (width_written_diff <= 0.245583) => JOIN=JOIN (35.0/8.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (AVG_LINE_DIFF <= 0.020926) and (HEIGHT_DIFF <= 1.28) and (AVG_LINE_SPAC_DIFF <= 0.034014) and (width_written_diff <= 0.676174) and (RIGHT_MRGN_DIFF >= 5.684564) => JOIN=JOIN (37.0/11.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (AVG_LINE_DIFF <= 0.031496) and (WIDTH_DIFF <= 2.164873) and (width_written_diff <= 1.583972) and (AVG_LINE_SPAC_DIFF <= 0.04844) and (AVG_LINE_DIFF <= 0.008344) and (AVG_LINE_DIFF >= 0.006489) and (AVG_LINE_DIFF <= 0.006711) and (WIDTH_DIFF <= 1.033557) and (TOP_MRGN_DIFF <= 0.171053) => JOIN=JOIN (32.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.095238) and (AVG_LINE_DIFF <= 0.027644) and (HEIGHT_DIFF <= 1.265957) and (HEIGHT_DIFF <= 0.518947) and (AVG_LINE_SPAC_DIFF <= 0.033784) and (AVG_LINE_DIFF <= 0.001312) and (LEFT_MRGN_DIFF >= 5.015673) => JOIN=JOIN (18.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.153649) and (WIDTH_DIFF <= 1.446206) and (LEFT_MRGN_DIFF <= 0.533608) and (width_written_diff <= 0.76895) and (AVG_LINE_SPAC_DIFF >= 0.131748) and (AVG_LINE_SPAC_DIFF <= 0.135319) and (HEIGHT_DIFF >= 1.299213) and (AVG_LINE_DIFF <= 0.051647) and (AVG_LINE_DIFF >= 0.043942) => JOIN=JOIN (43.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 2.48304) and (width_written_diff <= 1.580392) and (LEFT_MRGN_DIFF <= 0.89612) and (AVG_LINE_DIFF <= 0.015748) and (AVG_LINE_DIFF <= 0) and (TOP_MRGN_DIFF >= 0.093333) and (width_written_diff >= 0.78) => JOIN=JOIN (26.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.153549) and (WIDTH_DIFF <= 1.446206) and (LEFT_MRGN_DIFF <= 0.426667) and (height_written_diff <= 1.866667) and (AVG_LINE_SPAC_DIFF >= 0.125984) and (height_written_diff >= 1.651156) and

(BOTTOM_MRGN_DIFF <= 0.394737) => JOIN=JOIN (35.0/7.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (WIDTH_DIFF <= 1.438596) and (LEFT_MRGN_DIFF <= 0.544803) and (AVG_LINE_DIFF <= 0.042949) and (WIDTH_DIFF <= 0.462047) and (AVG_LINE_DIFF >= 0.040541) and (LEFT_MRGN_DIFF >= 0.415187) => JOIN=JOIN (24.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.081769) and (AVG_LINE_DIFF <= 0.031496) and (HEIGHT_DIFF <= 1.043444) and (width_written_diff <= 0.680272) and (HEIGHT_DIFF >= 0.995168) and (RIGHT_MRGN_DIFF <= 0.428571) and (AVG_LINE_DIFF >= 0.012342) => JOIN=JOIN (23.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.102522) and (WIDTH_DIFF <= 2.673713) and (width_written_diff <= 1.337108) and (RIGHT_MRGN_DIFF <= 1.020202) and (LEFT_MRGN_DIFF <= 0.364569) and (AVG_LINE_SPAC_DIFF <= 0.048193) and (width_written_diff <= 0.746667) and (LEFT_MRGN_DIFF >= 0.296667) and (WIDTH_DIFF >= 0.969239) => JOIN=JOIN (28.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.153649) and (RIGHT_MRGN_DIFF <= 0.988189) and (LEFT_MRGN_DIFF <= 0.834646) and (width_written_diff <= 1.322835) and (LEFT_MRGN_DIFF <= 0.220472) and (WIDTH_DIFF <= 0.961901) and (HEIGHT_DIFF <= 1.163271) and (TOP_MRGN_DIFF <= 0.251701) and (AVG_LINE_SPAC_DIFF >= 0.133414) and (BOTTOM_MRGN_DIFF >= 0.539858) => JOIN=JOIN (40.0/3.0)

(AVG_LINE_SPAC_DIFF <= 0.086667) and (AVG_LINE_DIFF <= 0.033333) and (HEIGHT_DIFF <= 1.31) and (WIDTH_DIFF <= 1.182864) and (TOP_MRGN_DIFF >= 0.658606) and (LEFT_MRGN_DIFF <= 0.511303) and (TOP_MRGN_DIFF >= 1.361924) => JOIN=JOIN (41.0/12.0)

(AVG_LINE_SPAC_DIFF <= 0.086793) and (RIGHT_MRGN_DIFF <= 0.888158) and (width_written_diff <= 1.315566) and (LEFT_MRGN_DIFF <= 1.303357) and (AVG_LINE_SPAC_DIFF >= 0.069983) and (width_written_diff >= 1.256579) and (AVG_LINE_SPAC_DIFF <= 0.074561) => JOIN=JOIN (36.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.120157) and (RIGHT_MRGN_DIFF <= 0.982582) and (width_written_diff <= 1.176667) and (LEFT_MRGN_DIFF <= 1.396842) and (LEFT_MRGN_DIFF <= 0.192901) and (RIGHT_MRGN_DIFF >= 0.879502) and (AVG_LINE_SPAC_DIFF >= 0.09415) => JOIN=JOIN (19.0/0.0)

(AVG_LINE_SPAC_DIFF <= 0.082676) and (WIDTH_DIFF <= 1.438596) and (AVG_LINE_DIFF <= 0.016463) and (WIDTH_DIFF <= 0.11193) and (AVG_LINE_DIFF >= 0.007874) and (BOTTOM_MRGN_DIFF <= 0.442982) and (width_written_diff >= 0.910702) and (AVG_LINE_SPAC_DIFF >= 0.02) => JOIN=JOIN (28.0/1.0)

(AVG_LINE_SPAC_DIFF <= 0.13781) and (RIGHT_MRGN_DIFF <= 0.888158) and (LEFT_MRGN_DIFF <= 0.76498) and (width_written_diff <= 1.176334) and (RIGHT_MRGN_DIFF <= 0.155366) and (RIGHT_MRGN_DIFF >= 0.102362) and (LEFT_MRGN_DIFF <= 0.206248) and (RIGHT_MRGN_DIFF <= 0.109649) => JOIN=JOIN (31.0/4.0)

(AVG_LINE_SPAC_DIFF <= 0.095614) and (AVG_LINE_DIFF <= 0.028025) and (LEFT_MRGN_DIFF <= 0.735433) and (RIGHT_MRGN_DIFF <= 0.999123) and (height_written_diff <= 2.985503) and (BOTTOM_MRGN_DIFF <= 0.133858) and (width_written_diff >= 2.551667) and (LEFT_MRGN_DIFF >= 0.565826) => JOIN=JOIN (23.0/0.0)

=> JOIN=NON_JOIN (62583.0/2866.0)