Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Master's thesis

Training Set Construction Methods

Bc. Tomáš Borovička

Supervisor: Ing. Pavel Kordík, Ph.D.

15th May 2012

Acknowledgements

I would like to sincerely thank my supervisor Pavel Kordik for their help and support. Besides my supervisor, I would like to thank Marcel Jirina for their constructive comments, and for his important support throughout this work.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act 60 no. 121/2000 (copyright law), and with the rights connected with the copyright act included the changes in the act.

In Prague 15th May 2012 ....

Citation of this thesis

Tomáš Borovička. *Training Set Construction Methods: Master's thesis.* Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

In order to build a model, learning algorithms use a training set to set up its parameters. Validation set is used to select the best of learned models. To estimate the performance of a model on unseen data should be used an independent and identically distributed testing set. This thesis is focused on how to select data samples from an original set and place them into the training and testing sets. In the first chapter is an overview of existing approaches and new possible approaches are discussed. The second chapter is focused on experimental comparison of these methods.

Keywords Training set construction, classification, regression, representative set, data splitting, instance selection, class balancing, model learning.

Abstrakt

Učící algoritmy používají pro sestavení modelu tréninkovou množinu. Validační množina je používána pro výběr nejlepšího modelu. Pro stanovení přesnosti modelu na budoucích datech by měla být použita nezávislá testovací množina. Tato práce se zabývá tím, jak co nejlépe vybrat data z původní sady a umístit je do trénovací a testovací množiny. V první kapitole je přehled existujících metod a jsou diskutovány nová možná řešení. Druhá kapitola je zaměřená na experimentální porovnání těchto metod.

Klíčová slova Konstrukce trénovací množiny, klasifikace, regrese, reprezentativní množina, rozdělování množin, redukce dat, vyvažování tříd, učení modelu.

Contents

List of Figures

List of Tables

Introduction

A training set is a special set of labeled data providing known information that is used in the supervised learning to build a classification or regression model. We can imagine each training instance as a feature vector together with an appropriate output value (label, class identifier). A supervised learning algorithm deduces a classification or regression function from the given training set. The deduced classification or regression function should predict an appropriate output value for any input vector. The goal of the training phase is to estimate parameters of a model to predict output values with a good predictive performance in real use of the model.

When a model is built we need to evaluate it in order to compare it with another model or parameter settings or in order to estimate predictive performance of the model. Strategies and measures for the model evaluation are described in section 1.1.

For a reliable future error prediction we need to evaluate our model on a different, independent and identically distributed set that is different to the set that we have used for building the model. In absence of an independent identically distributed dataset we can split the original dataset into more subsets to simulate the effect of having more datasets. Some splitting algorithms proposed in literature are described in section 1.2 and experimentally compared in section 2.4.1.

During a learning process most learning algorithms use all instances from the given training set to estimate parameters of a model, but commonly lot of instances in the training set are useless. These instances can not improve predictive performance of the model or even can degrade it. There are several reasons to ignore these useless instances. The first one is a noise reduction, because many learning algorithms are noise sensitive [2] and we apply these algorithms before learning phase. The second reason is to speed up a model response by reducing computation. It is especially important for instance-based learners such as k-nearest neighbours, which classify instances by finding the most similar instances from a training set and assigning them the dominant class. These types of learners are commonly called lazy learners, memory-based learners or case-based learners [3]. Reduction of training sets can be necessary if the sets are huge. The size and structure of a training set needed to correctly estimate the parameters of a model can differ from problem to problem and a chosen instance selection method [3]. Moreover, the chosen instance selection method is closely related to the classification and regression method. The process of instance reduction is also called instance selection in the literature. A review of instance selection methods is in section 1.3. Experimental comparison in section 2.4.2.

The most of learning algorithms assumes that the training sets, used to estimate the parameters of a model or to evaluate a model, have proportionally the same representation of classes. But many particular domains have classes represented by a few instances while other classes have a large number of representative instances. Methods that deal with the class imbalance problem are described in section 1.4 and experimentally compared in section 2.4.3.

Basic notations

In this section we set up basic notation and definitions used in the document.

A population is a set of all existing feature vectors (features). By S we denote a sample set defined as a subset of a population collected during some process in order to obtain instances that can represent the population.

According to the previous definition the term representativeness is closely related. We can define a *representative set* S^* as a special subset of an original dataset S, which satisfies three main characteristics [4]:

1. It is significantly smaller in size compared to the original dataset.
2. It captures the most of information from the original dataset compared to any subset of the same size.
3. It has low redundancy among the representatives it contains.

A training set is in the idealized case a representative set of a population. Any of mentioned methods is not needed if we have representative subset of the population. But we never have it in practise. We usually have a random sample set of the population and we use various methods to make it as representative as possible. We will denote a training set by R.

In order to define a representative set we can define a *minimal consistent subset* of a training set. Given a training set R, we want to obtain a subset $R^* \subset R$ such that R^* is the smallest set of instances such that $Acc(R^*) \cong Acc(R)$, where $Acc(X)$ denotes the classification accuracy obtained using X as a training set [5].

Sets used for an evaluation of a model are the validation set V, usually used for a model selection, and the testing set T, used for model assessment.

CHAPTER 1

Review

1.1 Model evaluation

The model evaluation is an important but often underestimated part of model building and assessment. When we have prepared and preprocessed data we want to build a model with the ability to accurately predict future observations. We do not want a model that perfectly fits training data, but we need a model that is reliable after deployment in the real use. For this purpose we should have two phases of a model evaluation. In the first phase we evaluate a model **in order to estimate the parameters of the model** during the learning phase. This is a part of the model selection when we select the model with the best results. This phase is also called as the validation phase. It does not necessary mean that we choose a model that best fits a particular set of data. The well learned model captures only the underlying phenomenon, not the noise. A model that captures a noise is called as *over-fitted* [6]. In the second phase we evaluate the selected model **in order to assess the real performance of the model** on new unseen data. Process steps are shown below.

1. Model selection
   a) Model learning (Training phase)
   b) Model validation (Validation phase)
2. Model assessment (Testing phase)

1.1.1 Evaluation methods

During building a model, we need to evaluate its performance in order to validate or assess it as we mention earlier. There are more methods how to check our model, but not all are usually sufficient or applicable in all situations. We should always choose the most appropriate and reliable method for our purpose. Some of common evaluation methods are [7]:

Comparison of the model with physical theory
A comparison of the model with a physical theory is the first and probably the easiest way how to check our model. For example, if our model predicts a negative quantity or parameters outside of a possible range, it points to a poorly estimated model. However, a comparison with a physical theory is not always possible nor sufficient as a quality indicator.

Comparison of model with theoretical or empirical model
Sometimes a theoretical model exists, but may be to complicated for a practical use. In this case, the theoretical model could be used for a comparison or evaluation of the accuracy of the built model.

Collect new data for evaluation
The use of data collected in an independent experiment is the best and the most preferred way for a model evaluation. It is the only way that gives us a real estimate of the model performance on new data. Only new collected data can reveal a bias in a previous sampling process. This is the easiest way if we can easily repeat the experiment and sampling process. Unfortunately, there are situations when we are not capable to collect new independent data for this purpose either due to a high cost of the experiment or another unrepeatability of the process.

Use the same data as for model building
The use the same data for evaluation and for a model building usually leads to an optimistic estimation of real performance due to a positive bias. This is not recommended method and if there is another way it could not be used for the model evaluation at all.

Reserve part of the learning data for evaluation
A reserve part of the learning data is in practise the most common way how to deal with the absence of an independent dataset for model evaluation. As the reserve part selection from the data is usually not a simple task, many methods were invented. Their usage depends on a particular domain. Splitting the data is wished to have the same effect as having two independent datasets. However, this is not true, only newly collected data can point out the bias in the training dataset.

1.1.1.1 Evaluation measures

For evaluating a classifier or predictor there is a large variation of performance measures. However, a measure, good for evaluating a model in a particular domain, could be inappropriate in another domain and vice versa. The choice of an evaluation measure depends on the domain of use and the given problem. Moreover, different measures are used for classification and regression problems. The measures below are shortly described basics for the model evaluation. For more details see [8, 9].

Measures for classification evaluation

The basis for analysing classifier performance is a **confusion matrix**. The confusion matrix describes how well a classifier can recognize different classes. For c classes, the confusion matrix is an $n \times n$ table, which (i, j)th entry indicates the count of instances of the class i classified as j. It means that correctly classified instances are on the main diagonal of the confusion matrix. The simplest and the most common form of the confusion matrix is a two-classes matrix as it is shown in the table 1.1. Given two classes, we usually use a special terminology describing members of the confusion matrix. Terms *Positive* and *Negative* refer to the classes. *True Positives* are positive instances that were correctly classified, *True Negatives* are also correctly classified instances but of the negative class. On the contrary, *False Positives* are incorrectly classified positive instances and *False Negatives* are incorrectly classified negative instances.

		Predicted	
		Positive	Negative
True	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Figure 1.1: Confusion matrix

The first and the most commonly used measure is the **accuracy** denoted as $Acc(X)$. The accuracy of a classifier on a given set is the percentage of correctly classified instances. We can define the accuracy as

$$Acc(X) = \frac{correctly\ classified\ instances}{all\ instances}$$

or in a two-classes case

$$Acc(X) = \frac{TP + TN}{TP + TN + FP + FN} \ .$$

In order of having defined the accuracy, we can define the **error rate** of a classifier as

$$Err(X) = 1 - Acc(X) \ ,$$

which is the percentage of incorrectly classified instances.

If costs of making a wrong classification are known, we can assign different cost or benefit to each correct classification. This simple method is known as **costs and benefits** or **risks and gains**. The *cost matrix* has then the structure shown in Figure 1.2, where λ_{ij} corresponds to the cost of classifying

		Predicted		
		Class i	$\dots$	Class j
True	Class i	λ_{ii}	$\dots$	λ_{ij}
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	Class j	λ_{ji}	$\dots$	λ_{jj}

Figure 1.2: Cost matrix

the instance of class i to class j. Correctly classified instances have usually a zero cost ($\lambda_{ii} = \lambda_{jj} = 0$). Given a cost matrix, we can calculate the cost of a particular learned model on a given test set by summing relevant elements of the cost matrix accordingly to the model's prediction [9]. Here, the cost matrix is used as a measure, the costs are ignored during the classification. When a cost matrix is taken into account during learning a classification model, we speak about a cost-sensitive learning, which is mentioned in section 1.4 in the context of class balancing.

Using the accuracy measure fails in cases, when classes are significantly imbalanced (The class imbalanced problem is discussed in section 1.4). Good examples could be medical data, where we can have a lot of negative instances (for example 98%) and just a few (2%) of positive instances. It gives an impressive 98% accuracy, when we simply classify all instances as negative, which is absolutely unacceptable for medical purposes. The reason for this is that the contribution of a class to the overall accuracy rate is a function of its cardinality, with the effect that rare positives have an almost insignificant impact on the performance measure [10].

Alternatives for the accuracy measure are:
Sensitivity (also called *True Positive Rate* or *Recall*) - the percentage of truly positive instances that were classified as positive,

$$sensitivity = \frac{TP}{TP + FN} \,.$$

Specificity (also called *True Negative Rate*) - the percentage of truly negative instances that were classified as negative,

$$specificity = \frac{TN}{TN + FP} \,.$$

Precision - the percentage of positively classified instances that are truly positive,

$$precision = \frac{TP}{TP + FP} \,.$$

It can be shown that the accuracy is a function of the sensitivity and specificity:

$$accuracy = sensitivity \cdot \frac{TP+FN}{TP+TN+FP+FN} + specificity \cdot \frac{TN+FP}{TP+TN+FP+FN} \ .$$

F-measure combines precision and recall. It is generally defined as

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + \cdot recall}$$

where β specifies the relative importance of precision and recall. The F-measure can be interpreted as a weighted average of the precision and recall. A disadvantage of this measure is that it does not take the true negative rate into account. Another measure, that overcomes disadvantages of the accuracy on imbalanced datasets is the **geometric mean** of class accuracies. For the two-classes case it is defined as

$$gm = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}} = \sqrt{sensitivity \cdot specificity}$$

The geometric mean puts all classes on an equal footing, unfortunately there is no way to overweight any class [10].

The evaluation measure should be appropriate to the domain of use. If is it possible, usually the best way to write a report is to provide the whole confusion matrix. The reader than can calculate the measure which he is most interested in.

Measures for Regression evaluation

The measures described above are mainly used for classification problems rather than for regression problems. For regression problems more appropriate error measures are used. They are focused on how close is the actual model to the ideal model instead of looking if the predicted value is correct or incorrect. The difference between known value y and predicted value $f(x_i)$ is measured by so called **loss functions**. Commonly used loss functions (errors) are described bellow.

The **square loss** is one of the most common measures used for regression purposes, is it defined as

$$l(y_i, f(x_i)) = (y_i - f(x_i))^2$$

A disadvantage of this measure is its sensitivity to outliers (because squaring of the error scales the loss quadratically). Therefore, data should be filtered for outliers before using of this measure. Another measure commonly used in regression is the **absolute loss**, defined as

$$l(y_i, f(x_i)) = |y_i - f(x_i)|$$

It avoids the problem of outliers by scaling the loss linearly. Closely similar measure to the absolute loss is the **ϵ-insensitive loss**. The difference between both is that this measure does not penalize errors within some defined range ϵ. It is defined as

$$l(y_i, f(x_i)) = max(|y - f(x)| - \epsilon, 0)$$

The average of the loss over the dataset is called **generalization error** or **error rate**. On the basis of the loss functions described above we can define the **mean absolute error** and **mean squared error** as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - f(x_i)|$$

and

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

, respectively. Often used measure is also the **root mean squared error**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$$

, which has the same scale as the quantity being estimated. As well as the squared loss the mean squared error is sensitive to outliers, while the mean absolute error is not. When a relative measure is more appropriate, we can use the **relative absolute error**

$$RAE = \frac{\sum_{i=1}^{n} |y_i - f(x_i)|}{\sum_{i=1}^{n} |y_i - \bar{y}|}$$

or the **relative squared error**

$$RSE = \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

1.1.1.2 Bias and Variance

With the most important performance measure - the mean square error, the bias, variance and bias/variance dilemma is directly related. They are described thoroughly in [11]. Due the importance of these characteristics, it is in place to describe them more in detail.

With given statistical model characterized by parameter vector θ we define estimator $\hat{\theta}$ of this model (classification or regression model in our case) as a function of n observations of x and we denote it as

$$\hat{\theta} = \hat{\theta}(x_1, \dots, x_N)$$

The MSE is equal to the sum of the variance and the squared bias of the estimate, formally

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias(\hat{\theta})^2$$

Thus either bias or variance can contribute to poor performance of the estimator.

The **bias** of an estimator is defined as a difference between the expected value of the method and the true value of the parameter, formally

$$Bias(\hat{\theta}) = E[\hat{\theta}] - \theta = E[\hat{\theta} - \theta]$$

In another words the bias says whether the estimator is correct on average. If the bias is equal to zero, the estimator is said to be unbiased. The estimator can be biased for many reasons, but the most common source of an optimistic bias is using of the training data (or not independent data from the training data) to estimate predictive performance.

The **variance** gives us an interval within which the error appears. For an unbiased estimator the MSE is equal to the variance. It means that even though an estimator is unbiased it still may have large MSE if the variance is large.

Since the MSE can be decomposed into a sum of the bias and variance, both characteristics need to be minimized to achieve good predictive performance. It is common to trade-off some increase in the bias for a larger decrease in the variance [11].

1.1.2 Comparing algorithms

When we have learned more models and we need to select the best one, we usually use some of described measures to estimate the performance of the model and then we simply choose the one with the highest performance. This is often sufficient way for a model selection. Another problem is when we need to prove the improvement in the model performance, especially if we want to show that one model really outperforms another on a particular learning task. In this way we have to use a test of statistical significance and verify the hypothesis of the improved performance.

The most known and most popular in machine learning is the **paired t test** and its improved version the **k-fold cross-validated pair test**. In paired t test the originial set S is randomly devided into a training set R and a testing set T. Models M_1 and M_2 are trained on the set R and tested on the set T. This process is repeated k times(usually 30 times [12]). If we assume that each partitioning is drawn independently, then also individual error rates can be considered as different and independent samples from a probability distribution, which follow *t distribution* with k degrees of freedom. Our null hypothesis is that the difference in mean error rates is zero. Then the *Student's t test* is computed as follows

$$t = \frac{\sum_{i=1}^{k} (Err(M_1)_i - Err(M_2)_i)}{\sqrt{Var(M_1 - M_2)/k}}$$

Unfortunately the given assumption is less than true. Individual error rates are not independent as well as error rate differences are not independent, because the training sets and the testing sets in each iteration overlaps. The **k-fold cross-validated pair test** mentioned above is build on the same basis. The difference is in the splitting into a training and a testing set, instead of a random dividing. The original set S is splitted into k disjoint folds of the same size. In each iteration one fold is used for testing and remaining $k-1$ folds for training the model. In this approach each test set is independent of the others, but the training sets still overlaps. For more details see [12].

The improved version, **the 5xcv paired t test**, proposed in [12] performs 5 replications of 2-fold cross-validation. In each replication, the original dataset is divided into two subsets S_1 and S_2 and each model is trained on each set and tested on the other set. This approach solves the problem of overlapping (correlated) folds, which led to poorly estimated means and large t values.

Another approaches described in literature are **McNemar's test** [13], **The test for the difference of two proportions** [14] and many others.

Methods described above consider comparison over one dataset, for comparison of classifiers over multiple datasets see [15].

1.1.3 Dataset comparison

In some cases we need to compare two datasets, if they have the same distributions. For example if we split the original dataset into a training and a testing set, we expect that a representative sample will be in each subset and distributions of the sets will be the same (with a specific tolerance of deviation). If we assess splitting algorithms, one of the criteria will be the capability of the algorithm to divide the original dataset into the two identically distributed subsets.

For comparing datasets distributions we should use a statistical test under the null hypothesis that distributions of the datasets are the same. These tests are usually called **goodness-of-fit** tests and they are widely described in literature [16, 17, 18, 19]. For an univariate case we can compare distributions relatively easily using one of the numerous graphical or statistical tests e.g. histograms, PP and QQ plots, the Chi-square test for a dicrete multinominal distribution or the Kolmogorov-Smirnov non-parametric test. For more details see [20].

A multivariate case is more complicated because generalization to more dimensions is not so straightforward. Generalization of the most cited goodness-of-fit test, the Kolmogorov-Smirnov test, is in [21, 22, 23].

In the case of comparing two subsets of one set, we use a naive approach for their comparison. We suppose that two sets are approximately the same, based on comparing basic multivariate data characteristic. We believe, that for our purpose the naive approach is sufficient. Advantages of this approach are its simplicity and a low computational complexity in comparison with the goodness-of-fit tests. A description of commonly used multivariate data characteristics follows.

The first characteristic is the **mean vector**. Let $\mathbf{x}$ represent a random vector of p variables, and $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ denote the i-th instance in the sample set, the **sample mean error** is defined as

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} x_i = \begin{pmatrix} \bar{\mathbf{x}}_1 \\ \bar{\mathbf{x}}_2 \\ \vdots \\ \bar{\mathbf{x}}_p \end{pmatrix}$$

where n is the number of observations. Thus $\bar{x}_i$ is the mean of the i-th variable on the n observations. The mean of $\mathbf{x}$ over all possible instances in the population is called **population mean vector** and is defined as a vector of expected values of each variable, formally

$$\mu = E(\mathbf{x}) = \begin{pmatrix} E(x_1) \\ E(x_2) \\ \vdots \\ E(x_p) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{pmatrix}$$

Therefore, $\bar{\mathbf{x}}$ is an estimate of μ.

Second characteristic is the **covariance matrix**. Let $s_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$ be a sample covariance between j-th and k-th variable. We define the **sample covariance matrix** as

$$S = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,p} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p,1} & s_{p,2} & \cdots & s_{p,p} \end{pmatrix}$$

Because $s_{jk} = s_{kj}$, the covariance matrix is symmetric and there are variances s_j^2, the squares of standard deviations s_j, on the diagonal of the matrix. Therefore, the covariance matrix is also called **variance-covariance** matrix. As for the mean, the covariance matrix over whole population is called **population covariance matrix** and is defined as

$$\Sigma = cov(\mathbf{x}) = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,p} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p,1} & \sigma_{p,2} & \cdots & \sigma_{p,p} \end{pmatrix}$$

where $\sigma_{jk} = E[(x_j - \mu_j)(x_k - \mu_k)]$.

The covariance matrix contains $p \times p$ values corresponding to all pairs of variables and their covariances. The covariance matrix could be inconvenient in some cases and therefore it can be desired to have one single number as an overall characteristic. One measure summarising the covariance matrix is called **generalized sample variance** and is defined as the determinant of the covariance matrix

$$\textit{generalized sample variance} = |S|$$

The geometric interpretation of the generalized sample variance is a p-dimensional hyperellipsoid centered at $\bar{\mathbf{x}}$.

More details about the multivariate data characteristic can be found in [24].

1.2 Data splitting

In the ideal situation we have collected more independent datasets or we can simply and inexpensively repeat an experiment to collect new ones. We can use independent datasets for learning, model selection and even an assessment of the prediction performance. In this situation we have not any reason to split any particular dataset. But in situation when only one dataset is available and we are not capable to collect new data, we need some strategy to perform particular tasks described earlier. In this section we review several data splitting strategies and data splitting algorithms which try to deal with the problem of absence of independent datasets.

1.2.1 Data splitting strategies

When only one dataset is given, several possible ways how to use available data come into consideration to perform tasks described in section 1.1 (training, validation, testing). We can split available data into two or more parts and use each to perform a particular task. Common practise is to split data into two or three sets:

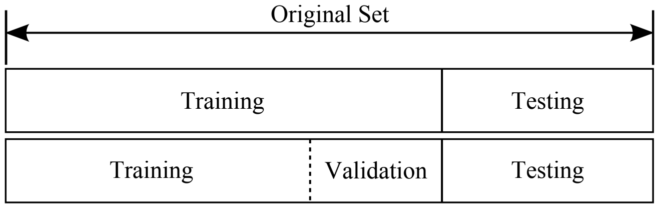


Figure 1.3: Two and three way splitting

Training set - a set used for learning and estimating parameters of the model.

Validation set - a set used to evaluate the model, usually for model selection.

Testing set - a set of examples used to assess the predictive performance of the model.

Let us define following data splitting strategies according to how data used in a process of model building are available.

The null strategy (*Strategy 0*) is when all available data are used for all tasks. Training, selecting and making an assessment on the same data usually leads to over-fitting of the model and to an over-optimistic estimate of the predictive accuracy. The error estimated on the same set as the model was trained is known as **re-substitution error**.

The strategy motivated by the arrival of new data (*Strategy 1*) uses one set for training and the second set, containing the first set and newly collected data, for the assessment. Merging new collected data with the old data loses the independence of model selection and assessment, which can lead to an over-optimistic estimate of the performance of the model.

The most commonly used strategy is to split data into two sets, a training set and a testing set. The training set (also called the estimation set) is used to estimate the parameters of the model and also for model selection (validation). The testing set is then used to assess the prediction performance of the model (*Strategy 2*).

Another strategy (*Strategy 3*) which splits data into two sets uses one set for learning and the second for model selection and to assess its predictive performance.

The use an independent set for each task is generally recommended. This strategy (*Strategy 4*) splits available data into three sets.

Strategy	**Training**	**Validation**	**Testing**
0	All data	All data	All data
1	Part 1	All data	All data
2	Part 1	Part 1	Part 2
3	Part 1	Part 2	Part 2
4	Part 1	Part 2	Part 3

Table 1.1: Data usage in different splitting strategies

1.2.2 Data splitting algorithms

Many data splitting algorithms were proposed. Quality and complexity of algorithms differ and not any approach is superior in general. Data splitting methods and algorithms and their comparison can be found in literature [25, 7, 26, 27]. Some of commonly used algorithms are described bellow.

The holdout method described in [28] is the simplest method that takes an original dataset and splits it randomly into two sets. Common practise is to use one third for testing and the rest for training or half to half. Assuming that the performance of the model increases with the count of seen instances and decreases with the count of left instances apart of the training leads to higher bias and decreases the performance. In other words, both subsets might have different distributions. Moreover, if a dataset is not large enough, and it is usually not, the holdout method is inefficient in the use of data. For example in a classification problem one or more classes might be missing in one of the subsets, which leads to poor estimation of the model as well as to its evaluation. In deal with this some advanced versions use so called stratification. *Stratified sampling* is a probability sampling, where an original dataset is divided into non-overlapping groups called strata, and instances are selected from each strata proportionally to the appropriate probability. It ensures that each class is represented with the same frequency in both subsets. But it still does not prevent inception of the bias in training and testing sets. For better reliability of the error estimation, the methods are repeated and the resulting accuracy is calculated as an average over all iterations. It can positively reduce the bias. The **Repeated holdout** method is also known as Monte Carlo Cross-validation, Random Sub-sampling or Repeated Evaluation Sets.

The most popular resampling method is **Cross-validation**. In **k-fold cross-validation**, the original dataset is splitted into k disjoint folds of the same size, where k is a parameter of the method. In each from k turns one fold is used for evaluation and the remaining $k-1$ folds for model learning as shown in Figure 1.4. As in the repeated holdout method, the resulting accuracy is the average of all turns. As well as holdout method, k-fold cross-validation suffers on a pessimistic bias, when k is small. Increasing the count of folds reduces the bias, but increases the variance of the estimation [11]. Experiments have shown that good results across different domains have the k-fold cross-validation method with ten folds [29], but in general k is unfixed. The k-fold cross-validation is very similar to the repeated holdout method with advantage that all the instances of the original dataset are used for learning the model and even for evaluation.

Leave-one-out cross-validation (LOOCV) is the special case of the k-fold cross-validation in which $k = n$, where n is the size of the original dataset. All test sets have always only one instance. This method makes the best use of data and does not involve any random sub-sampling. According to this, the LOOCV gives nearly unbiased estimates of a model performance but usually with large variability. However, this method is extremely computationally expensive, that makes it often inapplicable.

The **Bootstrap** method was introduced in [30]. The main idea of the

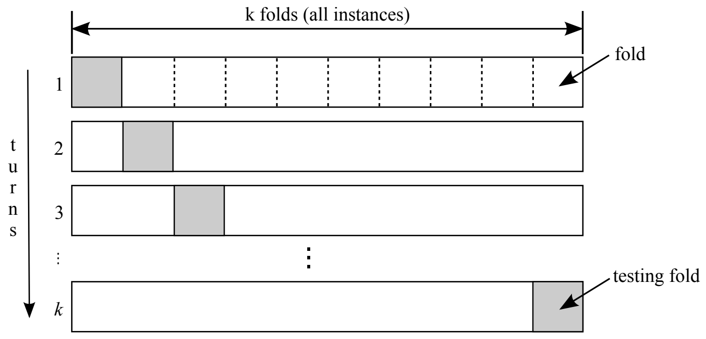


Figure 1.4: Cross-validation

method is described as follows. Given a dataset S of size n, generate B bootstrap samples by uniform sampling (*with replacement*), n instances from the dataset. Notice that sampling with replacement allows to select the same instance more than once. After re-sampling, estimate parameters of a model on each bootstrap sample and than estimate a prediction performance of the model on the original dataset. The overall prediction error is given by averaging these B estimates. Process is schematically shown in Figure 1.5.

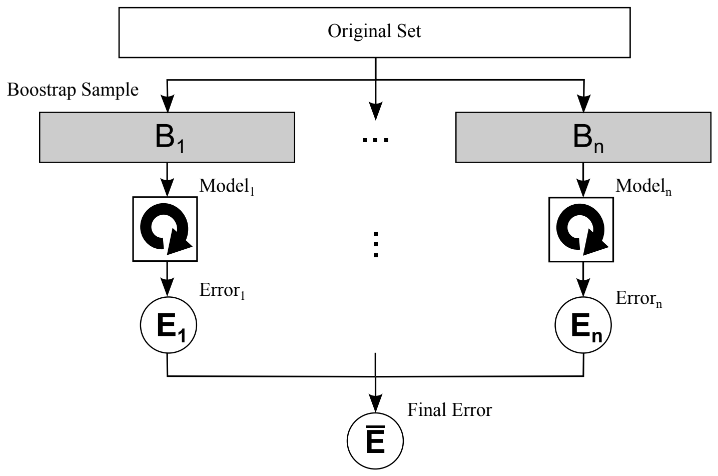


Figure 1.5: Bootstrap

The most known and commonly used approach is the **.632 bootstrap**. The number 0.632 in the name means the expected fraction of distinct instances of the original dataset appeared in the training set. Each instance has a probability of $1/n$ to being selected from n instances ($(1-1/n)$ to not being selected). It gives the probability of $(1-1/n)^n \approx e^{-1} \approx 0.368$ not to be selected after n samples. In other words, we expect that 63.2% instances of the original dataset will be selected for training and 36.8% remaining instances will be used for testing. The *.632 bootstrap estimate* is defined as

$$Acc(T) = \frac{1}{B}\sum_{i=1}^{B}(0.632 \cdot Acc(B_i)_{B_i'} + 0.368 \cdot Acc(B_i)_T)$$

where $Acc(B_i)_{B_i'}$ is the accuracy of the model built with bootstrap sample B_i as the training set and applied to the test set $B_i^{'}$ and $Acc(B_i)_T$ is the accuracy of the same model applied to the original dataset. Comparison of the bootstrap with other methods can be found in literature [31, 30, 32, 33, 34]. The results show that 0.632 bootstrap estimates have usually low variability but with a large bias in comparison with the cross-validation that gives approximately unbiased estimates, but with a high variability. It is also reported that the 0.632 bootstrap works best for small datasets. Some experiments showed that the .632 bootstrap fails in some cases, for more details see [35, 31, 34, 36].

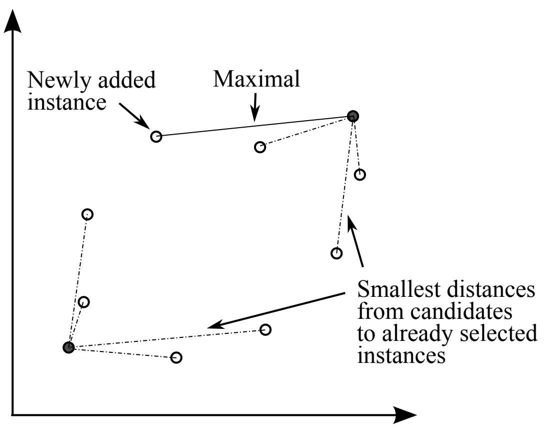
Kennard-Stone's algorithm (CADEX) [37, 38] is used for splitting datasets into two distinct subsets which cover approximately the same region of the factor space defined by the original dataset. Instead of measuring coverage by an explicit criterion, the algorithm follows two guide lines. The first one is that no instance from one set should be to far from any instance of the other set, and the second one is that the coverage should start on the boundary of the factor space. The instances are chosen sequentially and the aim is to select the instances in each iteration to get uniformly distributed instances over the space defined by original dataset. The algorithm works as follows. Let P be the subset of already selected instances and let Q be the dataset equal to T at the beginning. We define $Dist(p, q)$ as the distance from instance $p \in P$ to instance $q \in Q$ and $\Delta_q(P)$ will be the minimal distance from instance q over the set of already selected instances in P.

$$\Delta_q(P) = arg \min_{p \in P}(Dist(p, q))$$

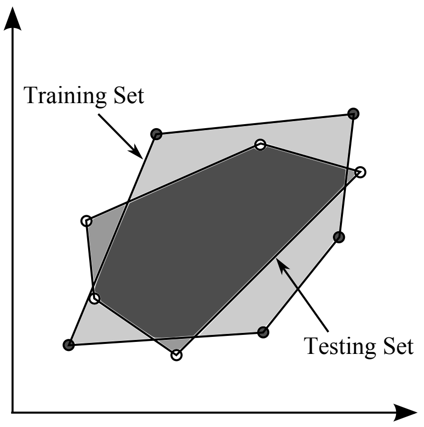
The algorithm starts with adding two most distant instances from Q to P (it is not necessary to select the most distant instances, they can be any instances, but accordingly to the idea of coverage, we usually choose two most distant instances). In each iteration the algorithm selects an instance from the remaining instances in the set Q using the criterion

$$\Delta_Q(P) = arg \max_{q \in Q} \Delta_q(P)$$

In other words, for each instance remaining in the dataset Q find the smallest distances to already selected instances in P and choose the one with the maximal distance among these smallest distances. The process is repeat until enough objects are selected. First iteration of the algorithm is shown in Figure 1.2.2 and in Figure 1.2.2 is final result with area covered by each set. Since the algorithm uses distances it is sensitive to the used metrics and eventual outliers. For classification purposes subsets should be selected from the individual classes [39]. Improved version of CADEX named **DUPLEX** is described in [7].


Newly added instance
Maximal
Smallest distances from candidates to already selected instances

[First iteration]



[Factor space coverage]

Figure 1.6: CADEX

Other methods can be considered when we take into account the following assumption. We suppose that two sets P and Q formed by splitting the original dataset S are as similar as possible when sum of distances of all pairs (one instance from the pair is from P and the other from Q) are minimized. Formally

$$d^* = \arg\min_d \sum_{\{p,q\}\in S} dist(p,q).$$

To find the optimal splitting to the two sets is computationally very expensive. Two heuristic approaches come to mind. The first is a method based on the **Nearest neighbour (NN)** rule. This simple method splits original datasets into two or more datasets by finding the nearest instance (nearest neighbour) of randomly chosen instance and putting each instance into a different subset.

The second heuristics finds the **closest pair** (described in [40]) of instances in S and put one instance into P and the second instance into Q. This is repeated until the set T is empty. The result of these algorithms are two disjoint subsets of the original dataset. The question is how properly will this heuristics work in practice.

1.3 Instance selection

As was mentioned earlier the instance selection is a process of reducing original dataset. A lot of instance selection methods have been described in the literature. In [3] it is argued that instance selection methods are problem dependent and none of them is superior over many problems then others. In this section we review several instance selection methods.

According to the strategy used for selecting instances, we can divide instance selection methods into two groups [5]:

Wrapper methods
The selection criterion is based on the predictive performance or the error of a model (commonly, instances that do not contribute to the predictive performance are discarded from the training set).

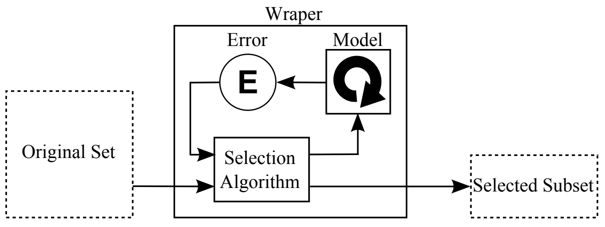


Figure 1.7: Wraper method

Filter methods
The selection criterion is a function that is not based upon an algorithm used for prediction but rather on features of the instance vector.

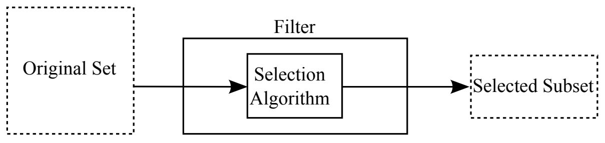


Figure 1.8: Filter method

Other dividing is also used in literature. Dividing of instance selection methods according to the type of application is proposed in [41]. **Noise filters** are focused on discarding useless instances while **prototype selection** is based on building a set of representatives (prototypes). How instance selection methods create final dataset offers the last presented dividing method. **Incremental methods** start with $S = \emptyset$ and take representatives from T and insert them into S during the selection process. **Decremental methods** start with $S = T$ and remove useless instances from S during the selection process. **Mixed methods** combine previous methods during the selection process.

A good review of instance selection methods is in [5, 42]. A comparison of instance selection algorithms on several benchmark databases is presented in [43]. Some of instance selection algorithms are described bellow.

1.3.1 Wrapper methods

The first published instance selection algorithm is probably **Condensed Nearest Neighbour (CNN)** [44]. It is an incremental method starting with new set R which includes one instance per class chosen randomly from S. In the next step the method classifies S using R as a training set. After the classification, each wrongly classified instance from S is added to R (*absorbed*). CNN selects instances near the decision border as shown in Figure 1.9. Unfortunately, due to this procedure the CNN can select noise instances. Moreover, performance of the CNN is not good [41, 45].

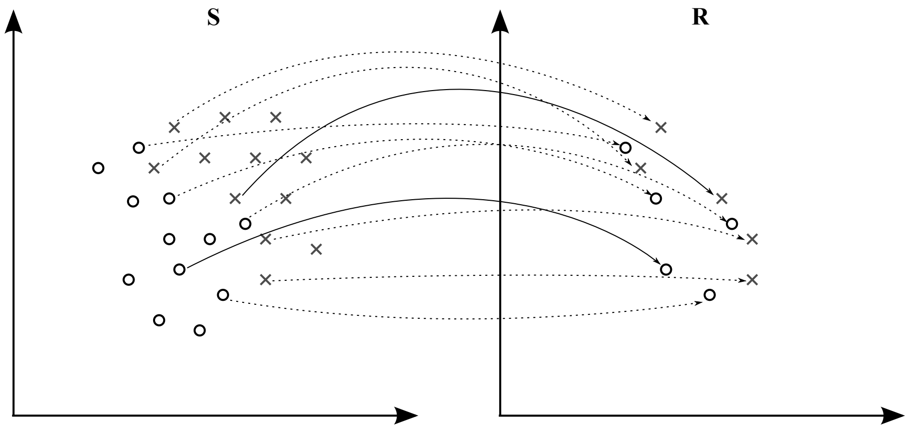


Figure 1.9: CNN - selected instances

Reduced Nearest Neighbour (RNN) is a modification of the CNN introduced by [46]. The RNN is a decremental method that starts with $R = S$ and removes all instances that do not decrease the predictive performance of a model trained using S.

Selective Nearest Neighbour (SNN) [47] is based on the CNN. It finds a subset $R \subset S$ satisfying that all instances are nearer to the nearest neighbour of the same class in R than to any neighbour of the other class in S.

Generalized Condensed Nearest Neighbour (GCNN)[48] is another instance selection decision rule based on the CNN. The GCNN works the same way as the CNN, but it also defines the following absorption criterion: instance x is absorbed if $\|x - q\| - \|x - p\| > \delta$, where p is the nearest neighbour of the same class as x and q is the nearest neighbour belonging to a different class than x.

Edited Nearest Neighbour (ENN) described in [49] is a decremental algorithm starting with $R = S$. The ENN removes a given instance from R if its class does not agree with the majority class of its neighbourhoods. ENN uses k-NN rule, usually with $k = 3$, to decide about the majority class, all instances misclassified by 3-NN are discarded as shown in Figure 1.10. An extension that runs the ENN repeatedly until no change is made in R is known as **Repeated ENN (RENN)**. Another modification of the ENN is **All k-NN** published by [50] It is an iterative method that runs the ENN repeatedly for all $k(k = 1, 2, \ldots, l)$. In each iteration misclassified instances are discarded. Another methods based on the ENN are **Multiedit** and **Editing by Estimating Conditional Class Probabilities** described in [51] and [52], respectively.

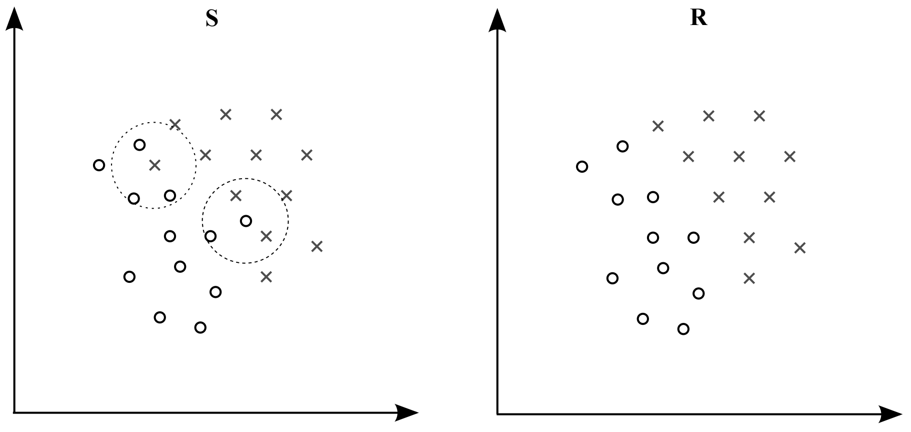


Figure 1.10: ENN - discarded instances (3-NN)

Instance Based (IB1-3) methods were proposed in [53]. The IB2 selects the instances misclassified by the IB1 (the IB1 is the same as the 1-NN algorithm). It is quite similar to the CNN, but the IB2 does not include one instance per class and does not repeat the process after the first pass through a training set like the CNN. The last version, the IB3, is an incremental algorithm extending the IB2. the IB3 uses a significance test and accepts an instance only if its accuracy is statistically significantly greater than the frequency of its class. Similarly, an instance is rejected if its accuracy is statistically significantly lower than the frequency of its class. Confidence intervals are used to determine the impact of the instance (0.9 to accept, 0.7 to reject).

Decremental Reduction Optimization Procedures (DROP1-5) are instance selection algorithms presented in [54]. These methods use an associate that can be defined by function $Associates(x)$ that collects all instances that have x as one of its neighbours. The DROP1 method removes instances from R that do not change a classification of its associates. The DROP2 is the same as the DROP1 but the associates are taken from the original sample set S instead of considering only instances remaining in R as the DROP1 method. The DROP3 and DROP4 methods run a noise filter first and then apply the DROP2 method. The DROP5 method is another version of the DROP2 extended of discarding the nearest opposite class instances.

Iterative Case Filtering (ICF) are described in [3]. They define $LocalSet(x)$ as a set of cases contained in the largest hypersphere centred at x such that only cases in the same class as x are contained in the hypersphere. They defined property $Adaptable(x, x')$ as $\forall x \in LocalSet(x')$. It means that instance x can be adapted to x'. Moreover they define two properties based on the adaptable property called *reachability* and *coverage* and defined as follows.

$$Reachability(x) = x' \in S : Adaptable(x', x)$$

$$Coverage(x) = x' \in S : Adaptable(x, x')$$

The algorithm is based on these two properties. At first, the ICF uses the ENN to filter noise instances then the ICF repeatedly computes defined instance properties and in each iteration removes instances that have $|Reachability(x)| > |Coverage(x)|$. The process is repeated until no progress is observed. Another method based on the same properties, the reachability and coverage, was proposed in [55].

Many other methods were proposed in literature. Some of them are based on evolutionary algorithms (EA)[56, 57, 58, 59], other methods use the support vector machine (SVM) [60, 61, 62, 63] or tabu search (TS) [64, 65, 66].

1.3.2 Filter methods

The **Pattern by Ordered Projections (POP)** method [67] is a heuristic approach to find representative patterns. The main idea of the algorithm is to select only some border instances and eliminate the instances that are not on the boundaries of the regions to which they belong. It uses the function $weakness(x)$, which is defined as the number of times that example x does not represent a border in a partition for every partitions obtained from ordered projected sequences of each attribute, for discarding irrelevant instances that have weaknesses equal to the number of attributes of dataset. The weakness of an instance is computed by increasing the weakness for each attribute, where the instance is not near to another instance with different class.

Another method based on finding border instances is the **Pair Opposite Class-Nearest Neighbour (POC-NN)** [68]. The POC-NN calculates the mean of all instances in each class and finds a border instance p_{b1} belonging to the class C_1 as an instance that is the nearest instance to m_2, which is the mean of class C_2. The same way it finds other border instances.

The **Maxdiff kd trees** described in [69] is a method based on *kd trees* [70]. The algorithm builds a binary tree from an original dataset. All instances are in the root node and child's nodes are constructed by splitting the node by a pivot, which is a feature with the maximum difference between consecutively ordered values. The process is repeated until no node can be split. Leaves of the tree are the output condensed set.

Several methods are based on clustering. They split an original dataset into n clusters and centres of the clusters are selected as instances [62, 42, 71]. Some extensions were proposed. **The Generalized-Modified Chang algorithm (GCM)** merges the nearest clusters with the same class and uses centres of the merged clusters. **The Nearest Sub-class Classifier** method (NSB) [72] selects more instances (centres) for each class using the Maximum Variance Cluster algorithm [73]. Another method is based on clustering. The **Object Selection by Clustering (OSC)** [74] selects border instances in heterogeneous clusters and some interior instances in homogeneous clusters.

Some prototype filtering methods were proposed in the literature. The first described is **Weighting prototype (WP)**[75] method. The WP method assigns a weight to each prototype ($\forall x \in T$) and selects only those with a larger weight than a certain threshold. The WS method uses a gradient descent algorithm for computing weights of instances. Another published prototype method is **Prototype Selection by Relevance (PSR)**[76]. The PSR computes the *relevance* of each instance in T. The most similar instances in the same class are the most relevant. The PSR selects a user defined portion of relevant instances in the class and the most similar instances belonging to the different class - the border instances.

1.4 Class balancing

A dataset is well-balanced, when all classes are represented with the same proportion, but in practise many domains of classification tasks are characterized by a small proportion of positive instances and a large proportion of negative instances, where the positive instances are usually our points of interest. This problem is commonly known as the class imbalance problem.

Although the performance of a classifier over all instances can be high, we are usually interested in classification of positive instances (true positive rate) only, where the classifier often fails, because it tends to classify all instances into the majority class. To avoid this problem some strategy should be used when a dataset is imbalanced.

Class-balancing methods can be divided into the three main groups according to the strategy of their use. Data level methods are used in preprocessing and usually utilize various ways of re-sampling. Algorithm-level methods modify a classifier or a learning process to solve the imbalance. The last strategy is based on combining various methods to increase the performance.

This chapter gives an overview of class balancing strategies and some particular methods. Two good and detailed reviews were published in [77, 78].

1.4.1 Data-level methods

The aim of these methods is to change distributions of classes by increasing the number of instances of the minority class (over-sampling), decreasing the number of instances of the majority class (under-sampling), by combinations of these methods or using other advanced sampling ways.

1.4.1.1 Under-sampling

The first and the most naive under-sampling method is **random under-sampling** [79]. The random under-sampling method balances the class distributions by discarding, at random, instances of the majority class. Because of the randomness of elimination, the method discards potentially useful instances, which can lead to a decrease of the model performance.

Several heuristic under-sampling methods have been proposed in literature, some of them are linked with instance selection metods mentioned in section 1.3. The first described algorithm is **Condensed nearest neighbour (CNN)** [44] and the second is **Wilson's Edited Nearest Neighbour (ENN)**[49]. Both are based on discarding noisy instances.

A method based on the ENN, the **Neighbourhood Cleaning Rule (NCL)** [80], discards instances from the minority and majority class separately. If an instance belongs to the majority class and it is misclassified by its three nearest neighbours' instances (the nearest neighbour rule [44]), then the instance is discarded. If an instance is misclassified in the same way and belongs to the minority class, then neighbours that belongs to the majority class are discarded.

Another method based on the *Nearest Neighbour Rule* is the **One-side Sampling (OSS)** [81] method. It is based on the idea of discarding instances distant from a decision border, since these instances can be considered as useless for learning. The OSS uses 1-NN over the set S (initially consisting of the instances of the minority class) to classify the instances in the majority class. Each misclassified instance from the majority class is moved to S.

The **Tomek Links** [50] focuses on instances near a decision border. Let p,q be instances from different classes and $dist(p, q)$ is the distance between p and q. Pair p, q is called the Tomek link if there is no closer instance of an opposite class to p or q ($dist(p,x) < dist(p,q)$ or $dist(q,x) < dist(p,q)$, where x is the instance of the opposite class than p, respectively q).

1.4.1.2 Over-sampling

The **random over-sampling** is a naive method, that balances class distributions by replication, at random, instances of the minority class. Two disadvantages of this method were described in literature. The first one, the instance replication increases likelihood of the over-fitting [82] and the second, enlarging the training set by the over-sampling can lead to a longer learning phase and a model response [81], mainly for lazy learners.

The most known over-sampling method is **Synthetic Minority Over-sampling Technique (SMOTE)** [82]. The SMOTE does not over-sample with replacement, instead, it generates "synthetic" instances of the minority class. The minority class is over-sampled by taking each instance of the minority class and its nearest neighbour and placing the "synthetic" instance, at random, along the line joining these instances (Figure 1.11). This approach avoids over-fitting and causes that a classifier creates larger and less specific decision regions, rather than smaller and more specific ones. The method based on the SMOTE reported better experimental results in *TP-rate* and *F-measure* [83], the **Borderline_SMOTE**. It over-samples only the borderline instances of the minority class.

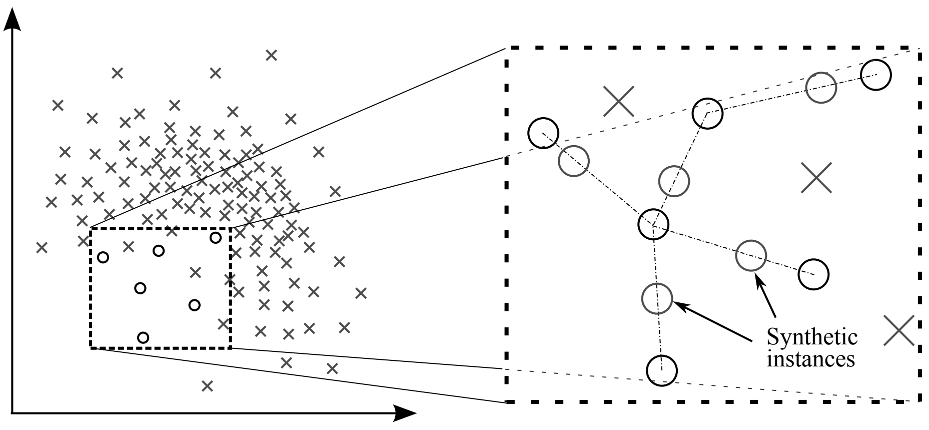


Figure 1.11: SMOTE

1.4.1.3 Advanced sampling

Some advanced re-sampling methods are based on re-sampling of results of the preliminary classification [78].

Over-sampling Algorithm Based on Preliminary Classification (OSPC) was proposed in [8]. It was reported that the OSPC can outperform under-sampling methods and the SMOTE in terms of classification performance [78].

The heuristic method proposed in [84, 85], the **Budget-sensitive progresive sampling algorithm** iteratively enlarges a training set on the basis of performance results from the previous iteration.

A combination of over-sampling and under-sampling methods to improve generalization features of learners was proposed in [80, 86, 83]. A comparison of various re-sampling strategies is presented in [87].

On the idea of combination of under-sampling and over-sampling can be built a simple approach. Count amount of instances in each class and compute the average number of instances, then randomly under-sample the majority class and randomly over-sample the minority class to balance the dataset. The question is how properly will this simple approach work in practise.

1.4.2 Algorithm level methods

Another approach to deal with imbalanced datasets modifies a classifier or a learning process rather than changing distributions of datasets by discarding or replicating instances. These methods are mainly based on overweighting the minority class, discriminating the majority class, penalization for misclassification or biasing the learning algorithm. A short description of published methods follows.

1.4.2.1 Algorithm modification

Ineffectiveness of the over-sampling method when the C4.5 decision tree learner with the default settings is used was reported in [88]. It was noted that under-sampling produces a reasonable sensitivity to changes in misclassification costs and a class distribution when over-sampling produces little or no change in the performance. It was also noted that modifications of C4.5 parameters in relation to the under/over-sampling does have a strong effect on overall performance.

A method that deals with imbalanced datasets by internally biasing the discrimination procedure is proposed in [89]. This method uses a weighted distance function in a classification phase of the k-NN. Weights are assigned to classes such that the majority class has a greater weighting factor than the minority class. This weighting causes that the distance to minority class instances is lower than the distance to instances of the majority class. Instances of the minority class are then used more often when classifying a new instance.

Different approaches using the SVM biased by various ways for dealing with imbalanced datasets were published. The method proposed in [90] modifies a kernel function for this purpose. In [91] it two schemes for controlling the balance between false positives and false negatives are proposed.

1.4.2.2 One-class learning

A one-class learning is an alternative to discriminative approaches that deal with imbalanced datasets. In the one-class learning, a model is built using only target class instances. The model is then learned to recognize these instances, which can be under certain conditions superior to discriminative approaches [92]. Two one-class learning algorithms were studied in literature, particularly the SVM [93, 94] and auto-encoders [92, 94]. An experimental comparison of these two methods can be found in [94]. Usefulness of the one-class learning on extremely unbalanced datasets composed of high dimensional noisy features is showed in [95].

1.4.2.3 Cost-sensitive learning

A cost-sensitive learning is another commonly used way in the context of imbalanced datasets. A classification model is extended with a cost model in the form of a cost matrix. Given the cost matrix as shown in Figure 1.2 in section 1.1 we can define *conditional risk* for making decision α_i about instance x as

$$R(\alpha_i|x) = \sum_j \lambda_{ij} P(j|x)$$

where $P(j|x)$ is a posterior probability of class j being true class of instance x. The goal in a cost-sensitive classification is to minimize the cost of misclassification. This means that the optimal prediction for an instance x is the class i that minimize a conditional risk. Note that the optimal decision can differ from the most probable class [96].

A method which makes classifier cost sensitive, the **MetaCost**, is proposed in [97]. The MetaCost learns an internal cost-sensitive model, then estimates class probabilities and re-labels training instances with their minimum expected cost classes. A new model is built using the relabelled dataset.

The **AdaCost** [98] method based on Adaboost [99] has been made a cost-sensitive by an over-weighting instances from the minority class, which are misclassified. Empirical experiments have shown, that the AdaCost has lower cumulative misclassification costs in comparison with the AdaBoost.

1.4.3 Ensemble learning methods

Ensemble methods are methods, which use a combination of methods with the aim to achieve better results. Two most known ensemble methods are *bagging* and *boosting*. The bagging (Bootstrap aggregating) proposed in [100] initially generates B bootstrap sets of the original dataset and then builds a classification or regression model using each bootstrap set. Predicted values of these models are combined to predict the final result. In classification tasks it works as follows. Each model has one vote to predict a class, the bagged classifier counts the votes and assigns the class with the most votes. For regression tasks, the predicted value is computed as the average of values predicted by each model.

The boosting, firstly described in [101], is based on the idea a powerful model is created using a set of weak models. The method is quite similar to the bagging. Like the bagging the boosting uses voting for a classification task or averaging for a regression task to predict the output value. However, the boosting is an iterative method. In each iteration a newly built model is influenced by the performance of those built previously. By assigning greater weights to the instances that were misclassified in previous iterations the model pays more attention on these instances.

Another in comparison with bagging and boosting less widely used method is *stacking* proposed in [102]. In the stacking method the original dataset is splitted into two disjoint sets, a training set and a validation set. Several base models are learned on the training set and then applied to the validation set. Using predictions from the validation set as inputs and correct values as the outputs, a higher level model is build. In comparison with the bagging and boosting, the stacking can be used to combine different types of models.

Ensemble methods such the bagging, boosting and stacking often outperform another methods. Therefore, they have been widely studied in recent years and lot of approaches have been proposed. The earlier mentioned **Adaboost** [99] and **AdaCost** [98] are other methods that use the boosting are **RareBoost** [103] or **SMOTEBoost** [104]. A method combining the bagging and stacking to identify the best combination of classifiers is used in [105]. Three agents (Naive Bayes, C4.5, 5-NN) are combined in the approach proposed in [86]. There are many other methods utilizing the mentioned approaches.

CHAPTER 2

Experiments

Experimental part is divided into three parts where each part corresponds to one particular task related to training set construction, described in previous chapter. Experiments are performed in the following way. For each experiment are chosen appropriate evaluation measures (Section 2.1). Experiments are designed to assess the behavior of the algorithms on different datasets and for various classification models (Section 2.2 and 2.3). Results are then compared in relation with a particular dataset and conrete classification model (Section 2.4).

2.1 Evaluation measures

For all tasks used for training set construction described in the previous chapter is the main goal to improve the performance of a model by constructing quality training set. Many measures for evaluation have been described in the previous chapter in section 1.1.1.1. For our experiments are mainly used two performance measures, classification accuracy as a commonly used measure for classification tasks and f-measure as a measure for imbalanced datasets, where we are focused on correct classification of instances of positive class and classification accuracy does not give enough information about correctness of positive instances classification.

However, each task (data splitting, instance selection and class balancing) has another quality measures that are appropriate to their purposes. Description of measures used for assessment of these methods follows in next sections.

2.1.1 Data splitting

Measures to asses data splitting methods are capability of a splitting method to select instances for building quality model as was described earlier and the second important measure of method quality is the reliability of performance prediction (to choose suitable instances for evaluation). We split an original dataset in order to evaluate the performance of learned model on unseen data. We need a method with a good estimate of future performance. Reliability of performance prediction of a method is measured as a difference in performance estimate by a method and the real performance on unseen data. Main measures used for data splitting methods comparison and assessment are:

- Learned model performance.
- Reliability of a model assessment.

2.1.2 Instance selection

For comparison and assessment of instance selection methods we use two measures. First is classification performance of a model learned on subset produced by these methods and the second measure is the **compression rate**. Compression rate is defined as

$$compression\ rate = \frac{|R^*|}{|R|}$$

where $|R|$ is the size of an original set and $|R^*|$ is the size of a set of selected instances.

Another measures that are related to compression rate or rather to amount of instances in the training set are **learning time** and **response time**. Learning time is a time consumed in learning phase to build a model. Response time is a time consumed by a model to give an appropriate response for an instance. This times strongly depend on the learning algorithm and the model itself. Usually when learning time is not relevant for building a model response time is crucial and vice versa. Main measures used for instance selection methods comparison and assessment are:

- Learned model performance.
- Compression rate.
- Learning and response time of a model.

2.1.3 Class balancing

The purpose of class balancing methods is to increase the performance of a model. Only measure for comparison and assessment of these methods is the performance of a model on balanced set produced by these methods. Main measure used for class balancing methods comparison and assessment is:

- Learned model performance

2.2 Design of experiments

At first, before main experiments, all datasets have been preprocessed. Attributes have been normalized into 0–1 scale and class labels have been remapped into integers. Because of absence of two independent samples, original dataset has been splitted into two subsets in order to have the same effect as having two independent samples. Splitting is performed randomly (Figure 2.2). Dataset have been splitted in a ration 7:3. The bigger sample (S_1) is used as an input for tested methods in each experiment and the smaller sample (S_2) is utilized for evaluation of the methods. Dataset has been splitted 10 – 100 times in dependence on the time complexity of the experiment.

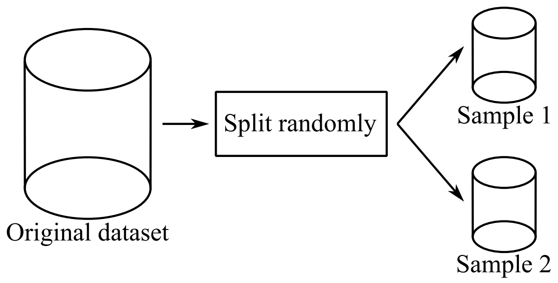


Figure 2.1: Experiment design

Results of each experiment are compared on six classification models:

- 1-nn classifier,
- 5-nn classifier,
- 10-nn classifier,
- 20-nn classifier,
- neural network classifier
  (500 cycles, learning rate=0.3, momentum=0.2, error=$1 \cdot 10^{-5}$),
- naive bayes classifier
  (minimum kernel bandwidth =0.1, number of kernels=10).

2.2.1 Data splitting

Sample S_1 is splitted by each splitting method into training and testing set. Training set is used for building a models. Performance of each model is evaluated on the testing set and on the sample S_2, where results on S_2 are taken as the real performance on unseen data. Difference between the real performance and the performance achieved on testing set is the reliability of data splitting method. This process is repeated 10 – 100 times for each splitting algorithm and for each sample S_1. The experiment is shown schematically in Figure 2.2.1. Following data splitting methods are compared:

- Null method – original dataset is used
- Random sampling
- Stratified sampling
- NN – Nearest Neighbour splitting
- CADEX – Kennard-Stone's algorithm
- Cross-validation – 10 fold cross-validation
- Bootstrap – .632 bootstrap

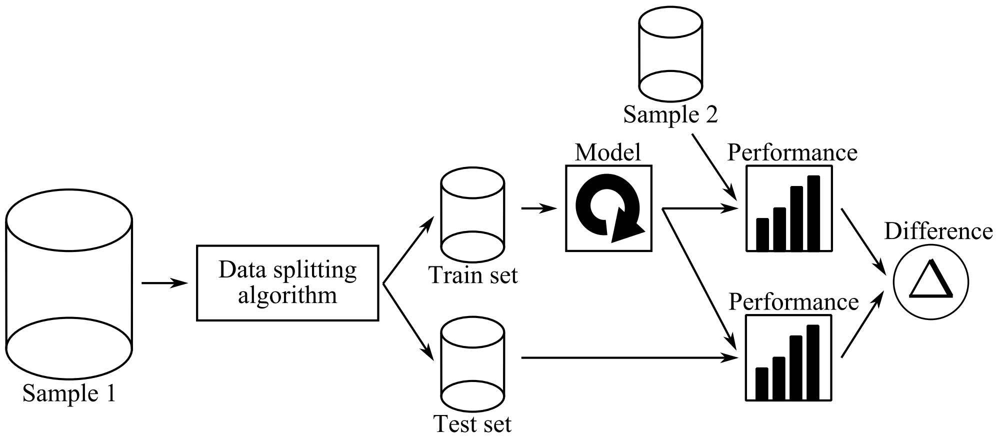


Figure 2.2: Design of experiment: data splitting

2.2.2 Instance selection

Each instance selection algorithm is applied on sample S_1. Between the original set and the reduced set is computed compression rate. Reduced set is used for building a models. Performance of learned models is evaluated on the sample S_2 and compared with the performance on the original set. This process is repeated 10 – 100 times for each instance selection algorithm and for each sample S_1. The experiment is shown schematically in Figure 2.2.2.

During the learning phase of a model building is measured the learning time. The values are averaged over all iterations. In the same way works measuring of the response time of a model. Response time is measured during evaluation of a model on the entire subset and values are also averaged over all iterations. In this case absolute values are not so meaningful as a differences. Following instance selection methods are compared:

- Null method – original dataset is used
- ENN – Edited Nearest Neighbour
- CNN – Condensed Nearest Neighbour
- DROP2 – Decremental Reduction Optimization Procedure 2
- DROP3 – Decremental Reduction Optimization Procedure 3
- RNN – Reduced Nearest Neighbour

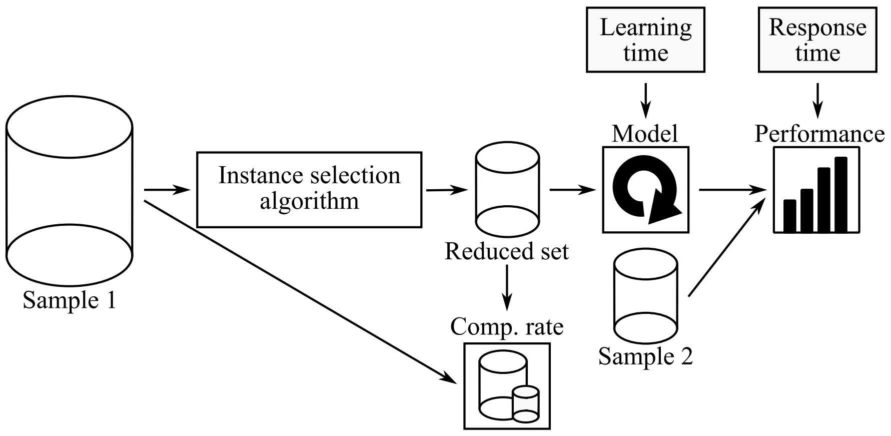


Figure 2.3: Design of experiment: instance selection

2.2.3 Class balancing

Class balancing methods are applied on sample S_1. Balanced set produced by each method is used for building a models. Built models are evaluated on sample S_2 and results are compared with results on the original dataset. This process is repeated 10 – 100 times for each class balancing algorithm and for each sample S_1. The experiment is shown schematically in Figure 2.2.3. Following class balancing methods are compared:

- Null method – original dataset is used
- Random under-sampling
- Random over-sampling
- Mixed balancing – combination of under-sampling and over-sampling
- SMOTE – Synthetic Minority Over-sampling Technique
- SMOTE + ENN

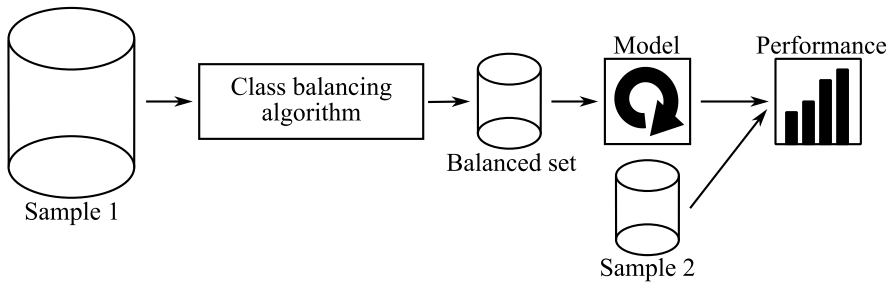


Figure 2.4: Design of experiment: class balancing

2.3 Experimental data

All dataset used for benchmarking are from the *UCI Machine Learning Repository* [106]. Datasets have been chosen by their characteristics to be appropriate for our experimental purposes. Each dataset have its main characteristics which are crucial for assessment of the results of each experiment. Benchmarking datasets and their characteristics are in Table 2.3, detailed description of each dataset is in Appendix A. Description of terminology used for dataset's characteristics is following:

normal – dataset is well balanced, without noise, instances are in two classes,

imbalanced – dataset contains significantly more instances of one class(es) than other class(es),

multi-class – dataset contains instances in more than two classes,

noisy – dataset contains significant part of noise instances.

All datasets are used in experiments with data splitting methods and instance selection methods and only imbalanced datasets are used in the the experiment with class balancing methods.

Dataset #	Name	Characteristic	Instances	Attributes	Classes
Dataset A.1	Haberman's Survival	normal	306	3	2
Dataset A.2	Blood Transfusion	imbalanced	748	4	2
Dataset A.3	Wine	multi-class	178	13	3
Dataset A.4	Pima Indians Diabetes	imbalanced	768	8	2
Dataset A.5	Mammographic Mass	normal	961	5	2
Dataset A.6	Ecoli	multi-class, imbalanced	332	7	6
Dataset A.7	Glass Identification	imbalanced	214	9	2
Dataset A.8	Heart Disease	noisy	270	13	2

Table 2.1: Benchmarking datasets

2.4 Results

In this section are presented results of all experiments. Presentation of results is divided into three parts corresponding to the purpose of each group of training set construction methods mentioned above. Individual results of each method on various datasets using different classifiers are discussed and all results are presented by graphs and figures. Graphs where is notable behaviour or interesting trend are in the text. All other graphical representation of results are given in appendix C. Evaluation of the results is more focused on trends and differences between methods that are more interesting and meaningful than absolute values of results.

2.4.1 Data Splitting

This section presents results of data splitting methods for six classification models (1-nn, 5-nn, 10-nn, 20-nn, neural network and naive bayes) on various datasets. Results on each dataset are discussed separately and at the end of the section is a summary of the results.

First dataset is *Haberman's Survival Data Set* (Dataset A.1). Dataset A.1 is two-class, well balanced set containing 306 instances. Dataset A.1 has been splitted by different splitting algorithms into the training and testing set. Classification models have been built on the training set produced by each splitting algorithm. Figure 2.5 shows the performance of the classifiers achieved on independent sample S_2. Each sub-figure corresponds to a single classification algorithm and each box-plot to a single splitting algorithm. Reliability of the performance prediction for each splitting algorithm is shown in Figure 2.6. Like in the previous figure each sub-figure corresponds to a single classification algorithm. Box-plots here show the differences in the predicted and real performance (performance estimate on S_2). The vertical dashed line represents the zero difference.

Classification performance of models is shown in Figure 2.5. No method significantly outperforms other methods. In the classification performance CADEX has had better results in more generalised models (10-nn,20-nn) and conversely the zero method (using entire dataset) has had better results on more sensitive models. Using all available data for learning a model even evaluation of the prediction performance gives optimistic estimate for all classifiers as has been expected. Good performance prediction with relatively small variance has had CADEX for k-nn classifiers. Other methods have had results with higher variance and with inconsistency to be optimistic or pessimistic. The worst estimate have had CADEX for neural network. The nearest neighbour splitting method has had good results with a small variance for all classifiers.

The *Mammographic Mass Data Set* (Dataset A.5) is the biggest dataset used for benchmarking. Dataset A.5 is two-class, well balanced dataset containing 961 instances. Classification performance of models learned on subsets of Dataset A.5 produced by different splitting algorithms is shown in Figure C.5. Performance prediction reliability of these splitting algorithms on dataset A.5 is shown in Figure C.6.

On this dataset gives nearest neighbour rule good results. It has had good estimate of the performance and has had smaller variance than other methods such as random or stratified sampling. The largest variance in the performance prediction has had cross-validation, but with good estimate in average. CADEX has had the estimate with small variance but with unacceptable pessimistic bias. CADEX estimate has differed over 10% against the real values. Poor results of CADEX on this dataset are surprising.

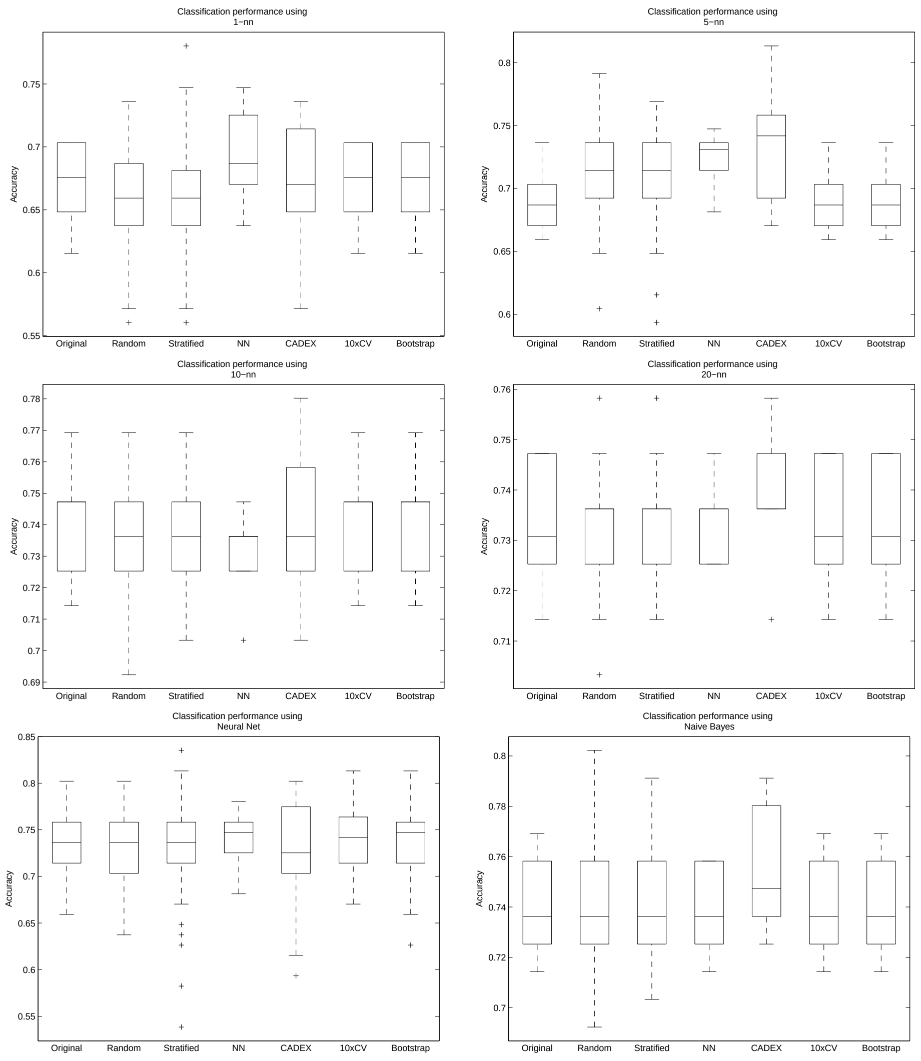


Figure 2.5: Model performance (Dataset A.1)

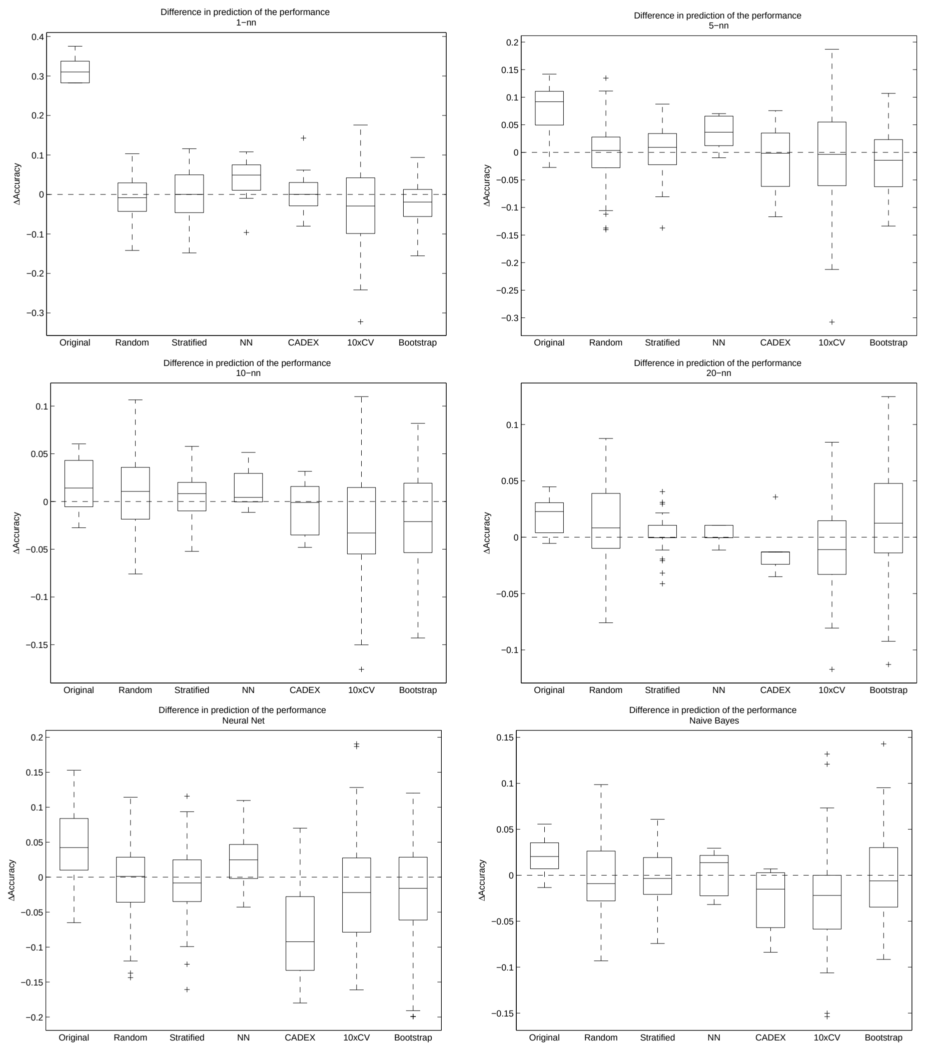


Figure 2.6: Difference in the prediction of model performance (Dataset A.1)

More interesting situation is with imbalanced dataset. The *Blood Transfusion Service Center Data Set* (Dataset A.2) is also two-class but highly imbalanced. Dataset A.2 contains 748 instances, from which 76 percent are negative. Appropriate performance measure for this dataset is f-measure, because we are focused mainly on correct detection of positive instances. F-measure achieved on independent sample S_2 for each classifier is shown in Figure 2.7. Prediction performance reliability for Dataset A.2 is shown in Figure 2.8.

In all sub-figures corresponding to classifiers in Figure 2.7 we can see that classifiers learned from subset produced by CADEX outperform others in classification performance. Significant improvement in comparison with other methods is obvious in Figure 2.4.1, 2.4.1 and 2.4.1. Significance of improvement has been confirmed by paired t-test at the five percent level of confidence. CADEX outperforms other methods also in reliability of the performance prediction where has had the most accurate estimate with relatively small variance in comparison with others. Good estimate has had also nearest neighbour splitting method, but with higher variance than CADEX. Methods based on a random factor have had high variance in estimate of performance. Best estimate of these methods on dataset A.2 has had Bootstrap.

Another imbalanced dataset is the *Pima Indians Diabetes Data Set* (Dataset A.4). Dataset A.4 is two-class dataset with 768 instances of which 268 are positive. Because we are already focused on positive instances, we use f-measure as the performance measure. Classification performance of models learned on subsets produced by data splitting methods is shown in Figure C.3. Performance prediction reliability of splitting methods for this dataset is shown in Figure C.4.

As on the previous imbalanced dataset, classifiers learned on subset obtained using CADEX has had better performance than same classifiers learned on subsets produced by other methods. In Figure C.3 is obvious increase in performance for all classifiers when CADEX has been used. In comparison with commonly used cross-validation, CADEX has had better estimate of the model performance with smaller variance. Good results on this dataset have had also random sampling and stratified sampling. Their estimates are more accurate with lower variance in comparison with other methods. Nearest neighbour method has had more optimistic estimate of performance but no so optimistic as the estimate on the original dataset. Bootstrap has had more often pessimistic estimate than optimistic.

The Glass identification Data Set (Dataset A.7) is also imbalanced and contains 214 instances in two classes. Classification performance of models learned on subsets of Dataset A.7 produced by different splitting algorithms is shown in Figure C.7. Performance prediction reliability of these splitting algorithms on this dataset is shown in Figure C.8.

Models with the best classification performance on Dataset A.7 are given by methods using all available data for learning. Cross-validation and bootstrap have had most accurate estimate of the performance, but still with large variability such as on previous datasets. All other methods decreased classification performance. Probably, the reason of decreased performance is that the dataset has not redundant or similar instances from the same class. It means that removing any instance decreases the capability of the model to classify similar instances correctly. This corresponds to the fact that the best results have been achieved by k-nn classifier with $k = 1$ and performance has been decreased with increasing k.

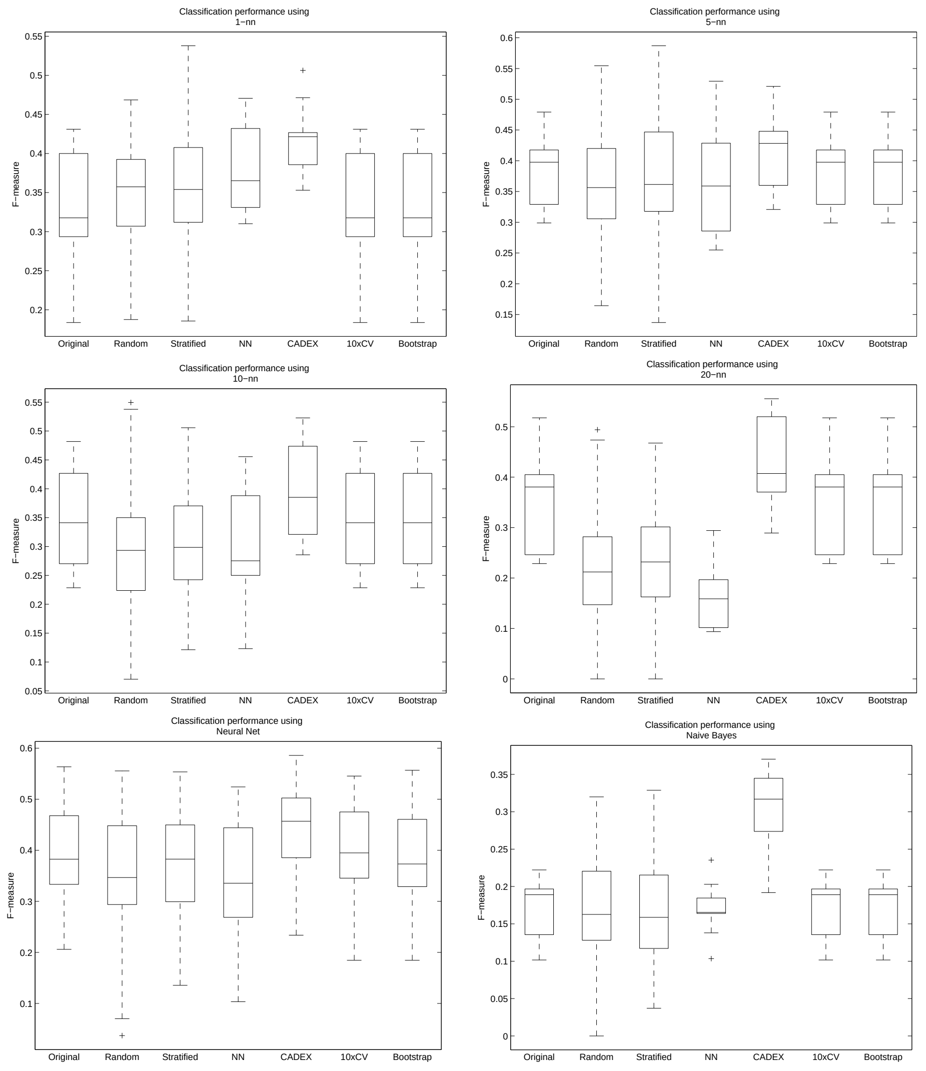


Figure 2.7: Model performance (Dataset A.2)

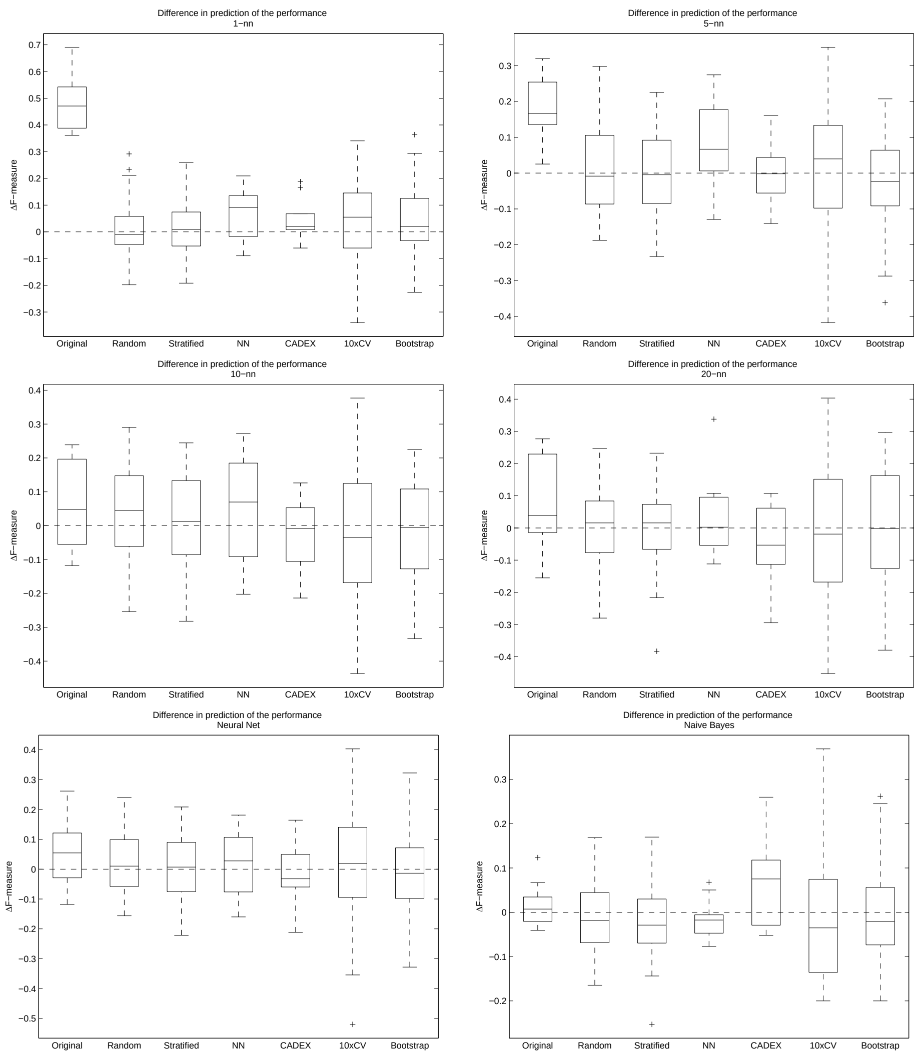


Figure 2.8: Difference in the prediction of model performance (Dataset A.2)

Although each multi-class classification task can be decomposed into more two-class classification tasks (by repeating classification for each class), we present results in situation with multi-class classification. The *Wine Data Set* (Dataset A.3) is a multi-class imbalanced dataset. Dataset A.3 contains 178 instances in 3 classes. Classification performance is shown in Figure C.1. Performance prediction reliability for Dataset A.3 is shown in Figure C.2.

Best results on this multi-class dataset have had stratified sampling and random sampling followed by bootstrap. Good results with a small pessimism has had CADEX. The nearest neighbour splitting has had too optimistic estimate of the performance. Cross-validation has had large variance.

The *Ecoli* (Dataset A.6) is the second benchmarking multi-class dataset. Dataset A.6 conatains 332 instances in 6 classes. Classification performance of models learned on subsets of Dataset A.6 produced by different splitting algorithms is shown in Figure 2.9. Performance prediction reliability of these splitting algorithms on Dataset A.6 is shown in Figure 2.10.

For Dataset A.6 has had best results cross-validation. This method outperforms others in the performance of learned models even in the performance prediction reliability. Random and stratified sampling have had also good estimate of a model performance, but models have been worse learned in comparison with methods using all available data (cross-validation, bootstrap). CADEX and nearest neighbour splitting method have had the worst results on this dataset. CADEX is on this dataset strongly pessimistic and contrariwise nearest neighbour rule is too optimistic.

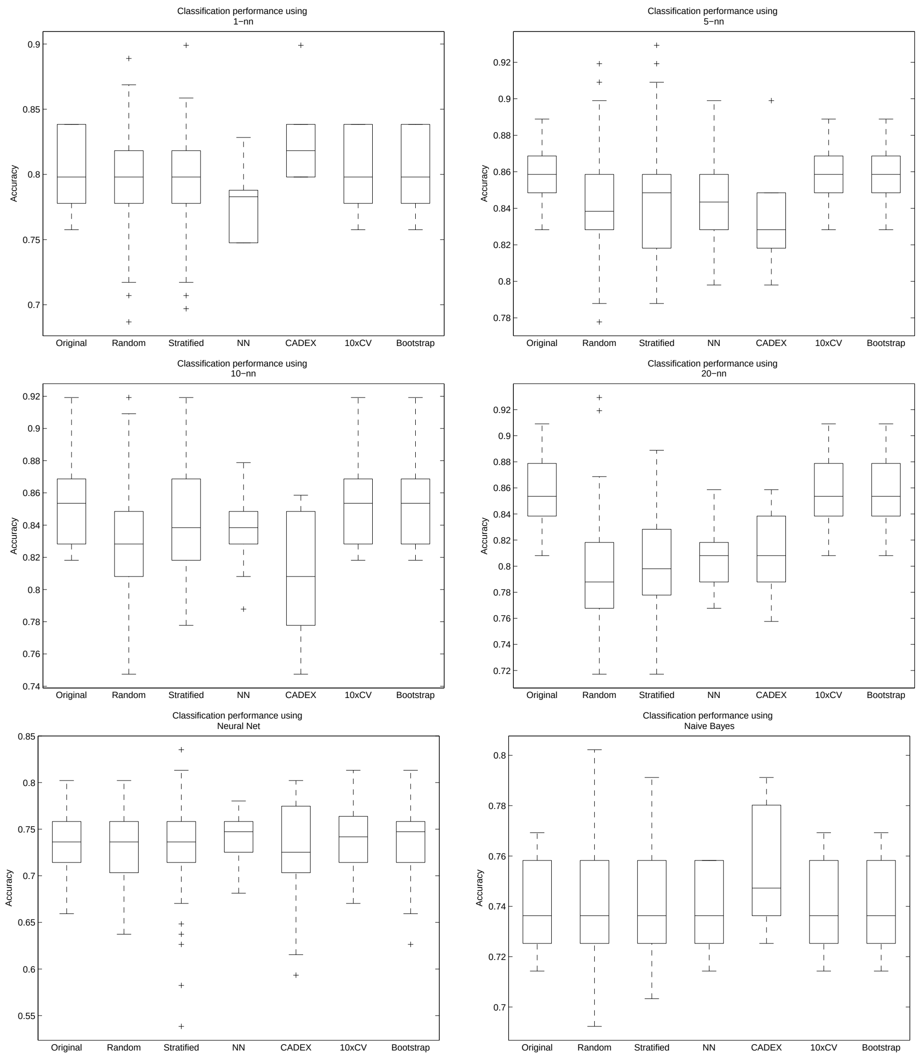


Figure 2.9: Model performance (Dataset A.6)

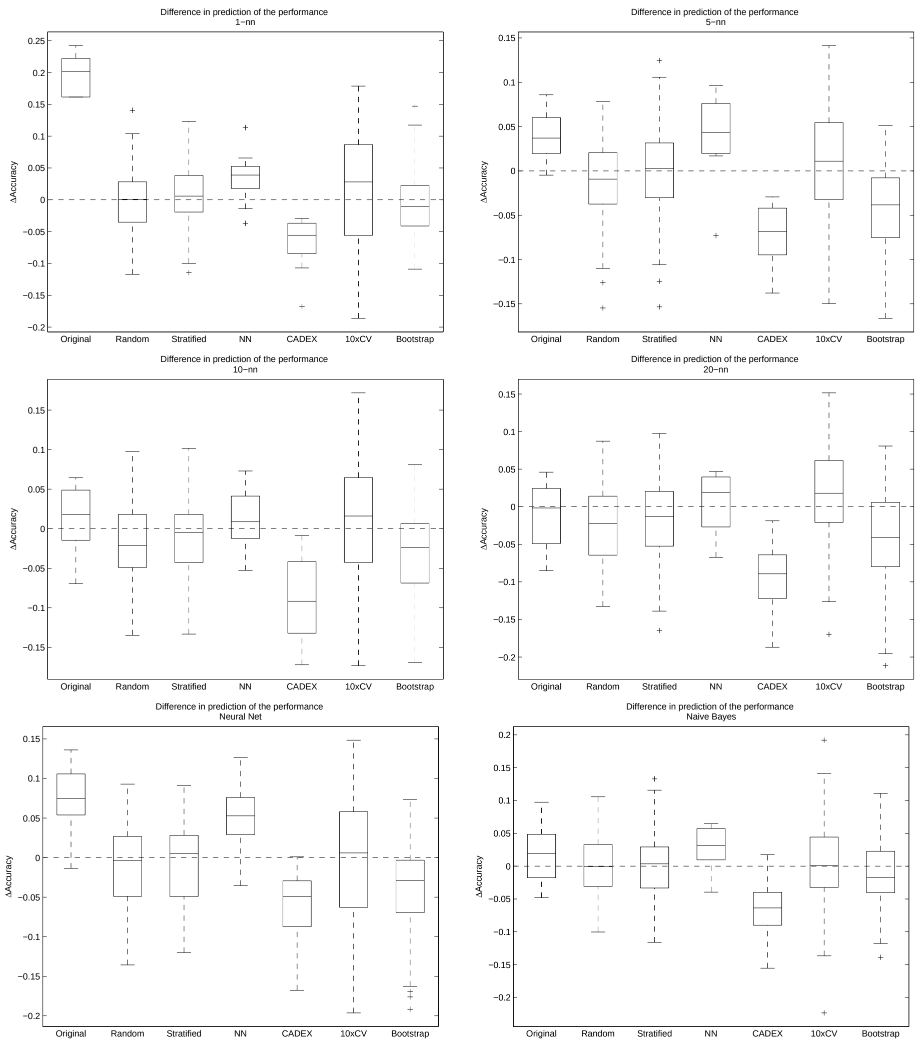


Figure 2.10: Difference in the prediction of model performance (Dataset A.6)

The *Heart Disease Data Set* (Dataset A.8) is two-class dataset with 10% of noise instances. Classification performance is shown in Figure 2.11. Performance prediction reliability for Dataset A.8 is shown in Figure 2.12.

All models using various methods have had on this dataset similar performance. In the performance prediction reliability has had CADEX on this dataset strongly pessimistic estimate. This can point to CADEX's high sensitivity to noise. Random sampling and stratified sampling perform on this dataset well and also cross-validation and bootstrapping give good estimate, but with a larger variance.

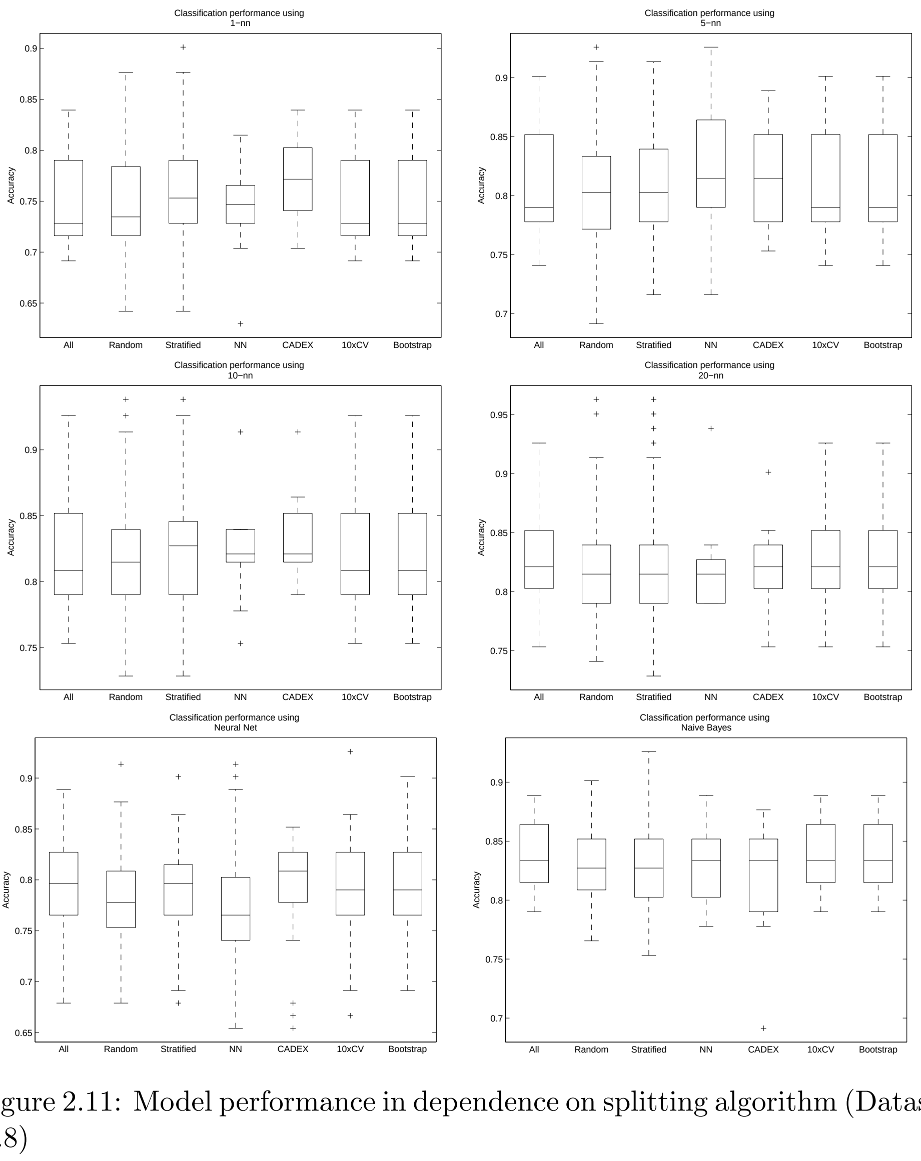


Figure 2.11: Model performance in dependence on splitting algorithm (Dataset A.8)

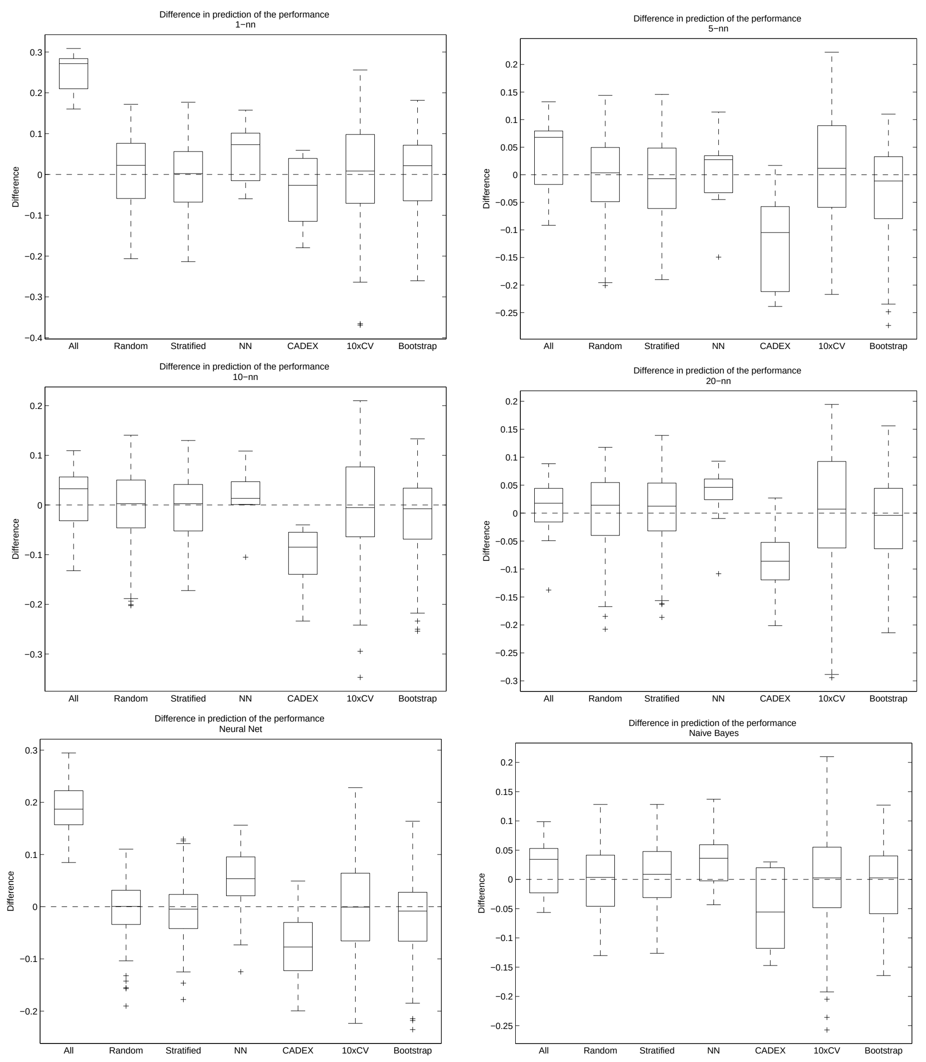


Figure 2.12: Difference in the prediction of model performance (Dataset A.8)

2.4.1.1 Summary

No method outperforms other methods on all or most of the datasets. Usually if a method gives better results with one classifier than other methods, it gives better results also with other classifiers. CADEX has had significantly better results than other methods on two of three imbalanced datasets, but on other datasets has had unacceptable pessimistic estimate of the performance. The cross-validation have had large variance in its estimate in comparison with other methods. The bootstrap has had similar results as the cross-validation but usually with a smaller variance and with a small pessimistic bias. Advantage of the cross-validation and bootstrap is that they use all available data for learning, this gives better results on small datasets. In stability of the performance prediction over all datasets has had the best results stratified sampling with no significant difference against random sampling. Both have had good estimate with acceptable variance in comparison with other methods. However, stratified sampling usually gives better learned model.

2.4.2 Instance selection

In this section are presented results of instance selection methods for six classification models (1-nn, 5-nn, 10-nn, 20-nn, neural network and naive bayes) on various datasets. As in the previous section, results on each dataset are discussed separately and at the end of the section is a summary of the results.

Dataset A.1 contains 306 instances and is well balanced. Figure 2.13 presents results of instance selection algorithms on Dataset A.1. The horizontal axis shows the compression of the training set and the vertical axis shows the performance achieved by a classifier on the compressed training set. The dashed horizontal line corresponds to the performance of a classifier on whole training set. Comparison of methods in the compression rate is shown in Figure 2.4.2. Best results in instance selection have been achieved on Dataset A.1 by DROP. DROP2 has had, with more than three times smaller set, a small decrease in classification performance. DROP3 outperforms DROP2 in the compression rate and in some cases even in the performance of a model. ENN that preserves nearly entire dataset increases the preformance of a model only in two cases. Results of DROP3 have been correlated with ENN because DROP3 uses ENN as a noise filter before it starts DROP2. CNN and RNN have had worse compression rate than DROP with higher decrease in classification performance.

Results of instance selection methods on Dataset A.2, which is two-class and imbalanced, are shown in Figure C.9. On this dataset ENN improves performance of all classifiers. DROP has had the best compression rate and DROP3 outperforms DROP2 in the performance of learned model. DROP did not work with k-nn, where k is great, because in the dataset was not enough instances. Poor results has had on this dataset RNN, which did not reduce the number of instance. CNN has reduced the amount of instance by half with a small decrease in performance.

Dataset A.3 contains in comparison with previous two-class datasets three classes. Results of instance selection methods on Dataset A.3 are shown in Figure C.10. As on the previous dataset, results for k-nn classifier are poor for great k, because in the dataset are not enough instance. This can be seen in Figures C.2, C.2 and C.2. This dataset probably does not contain any noise instances because ENN has had no effect. This is the reason why DROP2 and DROP3 have had the same results on this dataset. The best results has had DROP for neural net classifier where decrease performance by 1.3% and reduces the dataset to one fifth. CNN and RNN have had better results on this dataset than in previous cases, but still are worse than DROP.

Dataset A.6 is the second multi-class dataset with 332 instances in six classes. Results of instance selection methods on Dataset A.6 are shown in Figure C.11. Best results on Dataset A.6 has been achieved by DROP.

Dataset A.4 is two-class imbalanced dataset. Results of instance selection methods on Dataset A.4 are shown in Figure 2.15. On Dataset A.4 have had

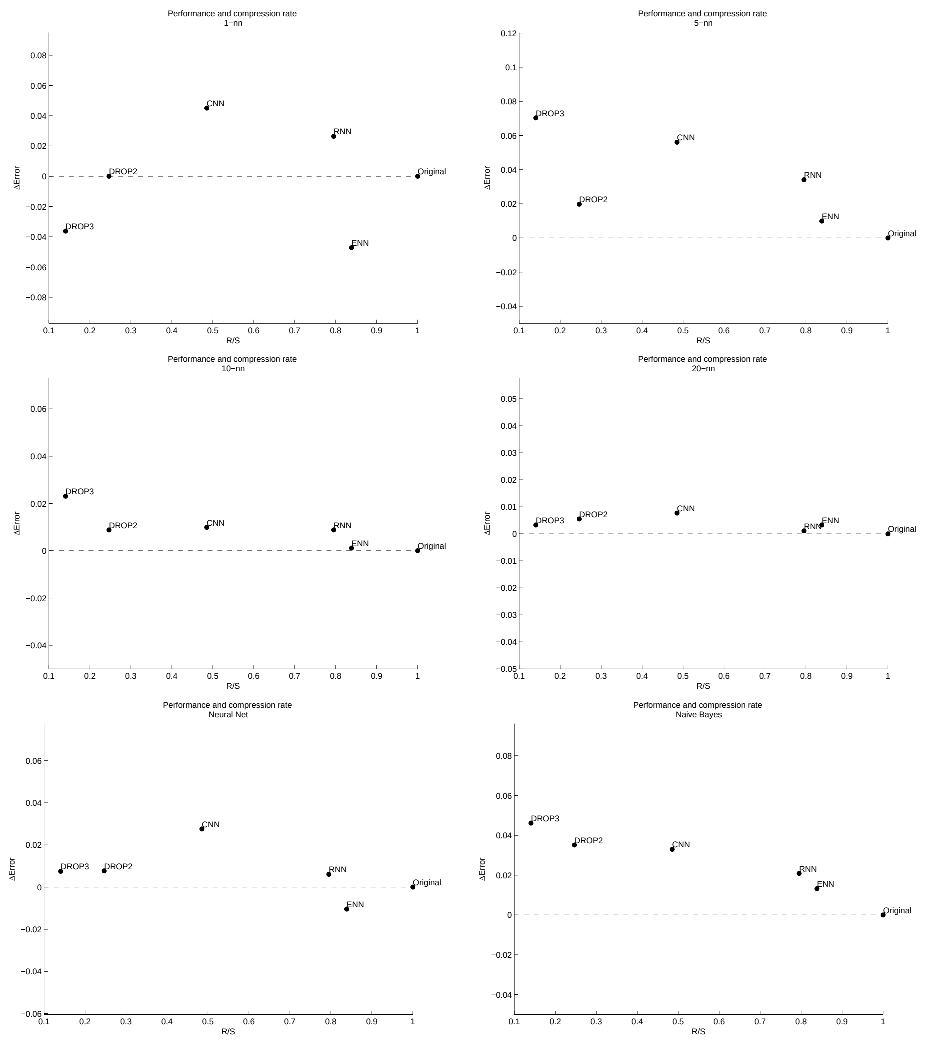


Figure 2.13: Classifier performance and method compression rate (Dataset A.1)

all instance selection methods similar results, but absolutely best results has had DROP. Only ENN increased the classification performance of models.

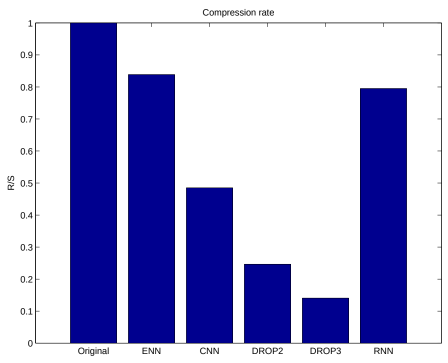


Figure 2.14: Training set compression (Dataset A.1)

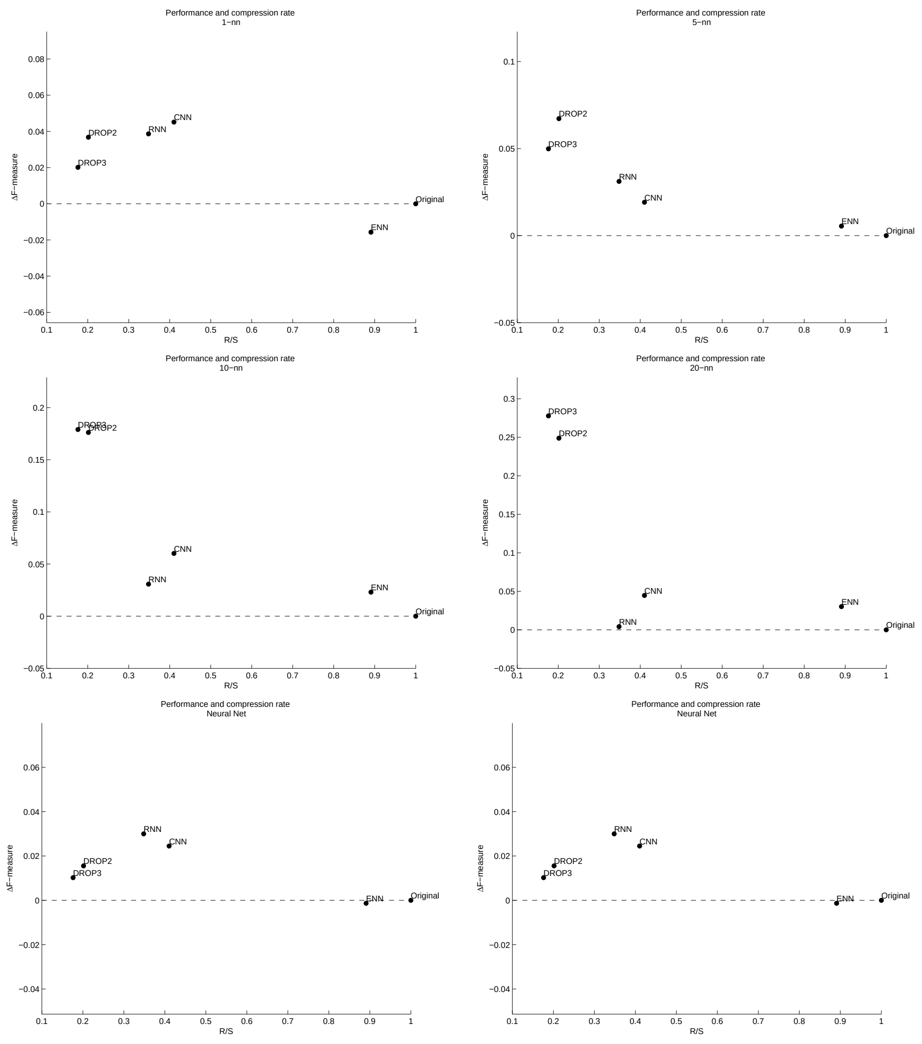


Figure 2.15: Model performance and method compression rate (Dataset A.4)

Dataset A.5 is the biggest dataset used for benchmarking. It contains 961 instances in two well balanced classes. Results of instance selection methods on Dataset A.5 are shown in Figure 2.16. On Dataset A.5 DROP2 has reduced amount of instances to one tenth with less than 2% decrease of accuracy for all classifiers. CNN has reduced dataset to one third and gives better or similar classification accuracy as DROP (except 1-nn classifier).

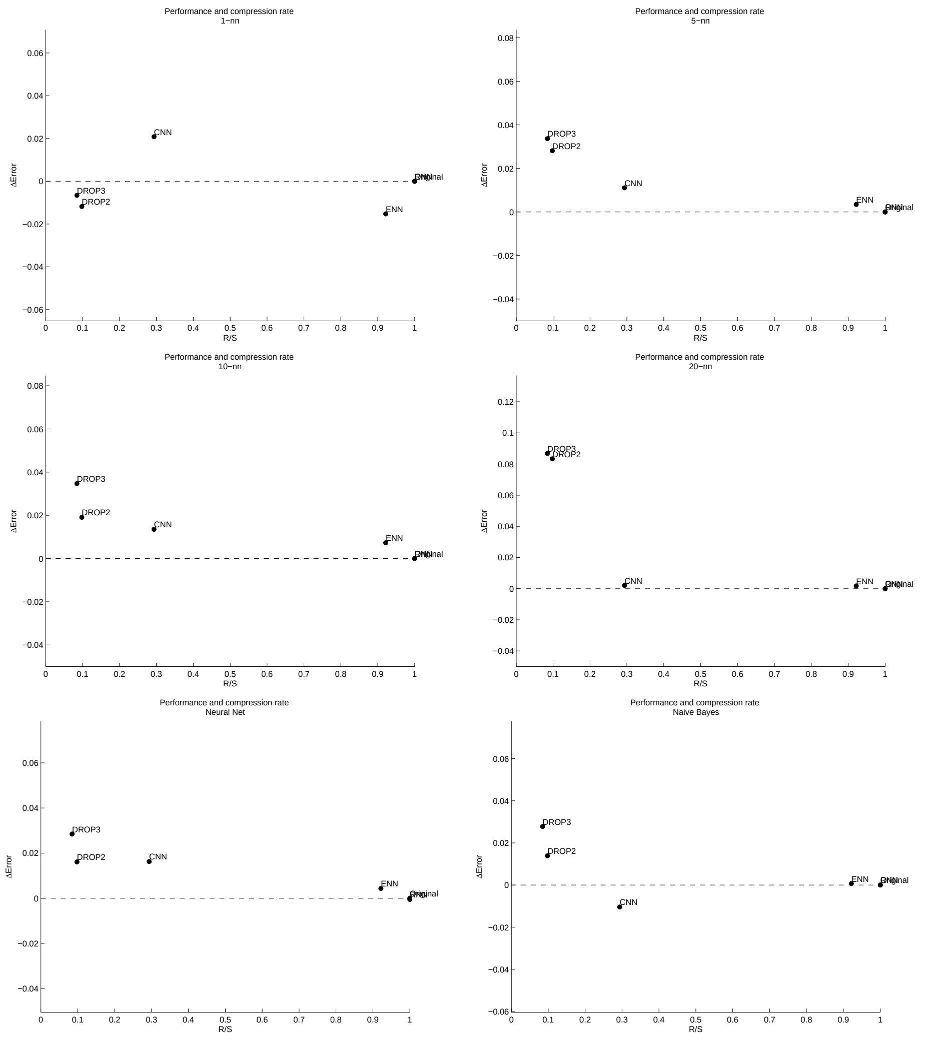


Figure 2.16: Model performance and method compression rate (Dataset A.5)

Dataset A.8 contains 10% of noise instances. Results of instance selection methods on Dataset A.8 are shown in Figure 2.17. Classification performance has not increased using ENN on this dataset as expected. Classification performance for naive bayes classifier even has decreased (also for k-nn classifier for greater k, but this is because in the dataset ware not enough instance). Nevertheless, increase in classification performance has been greater than 3% for neural network classifier and k-nn with less k.

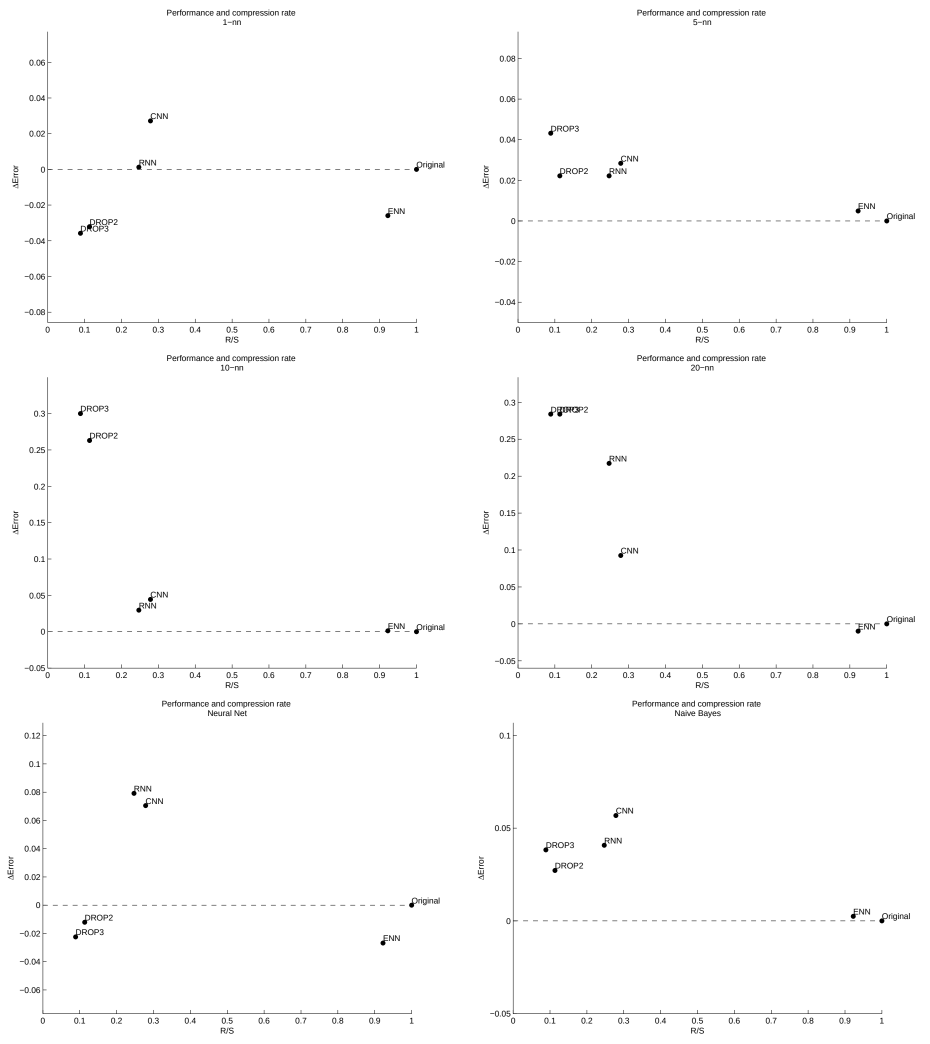


Figure 2.17: Model performance and method compression rate (Dataset A.8)

2.4.2.1 Learning and response time

The main reason why we reduce amount of instances is to speed up the learning phase of a model or response time of a model. In dependence of an algorithm used for learning and the model itself, number of instances affects learning and response time of a model. Figure 2.4.2 shows the compression rate of each instance selection method. Figure 2.18 shows the learning time for four different classifiers, where each subfigure corresponds to one classifier and each bar corresponds to one instance selection method. Figure 2.19 shows the response time in the same way.

In Figure 2.4.2.1 is obvious speed up in learning phase of neural network. Time has been reduced more than five times for DROP3, which reduced the training set to less than one fifth. Similar is it for support vector machine in Figure 2.4.2.1. With k-nn classifier is situation different, learning time is negligible here, because learning phase is only in assigning training set to the classifier. Amount of instances affects k-nn classifier in classification phase because classifier have to find k nearest neighbours through whole training set for each not classified instance. Response time of k-nn classifier is in Figure 2.4.2.1. Response time has been reduced three times with DROP3, which reduce training set to one fifth. Contrary, learned neural network has constant response time as can be seen in Figure 2.4.2.1.

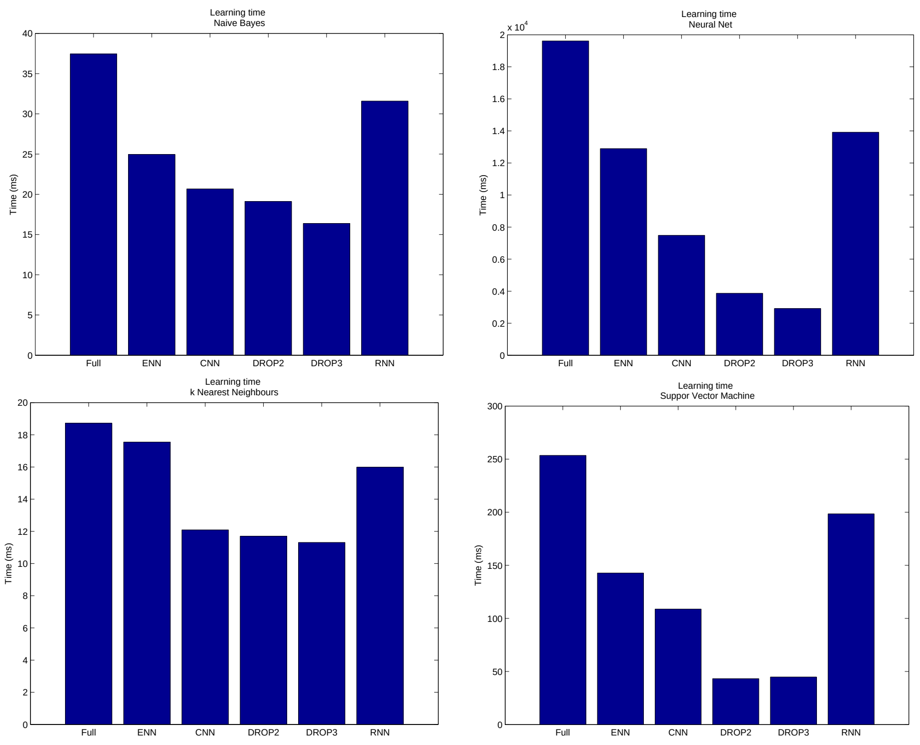


Figure 2.18: Learning time (Dataset A.1))

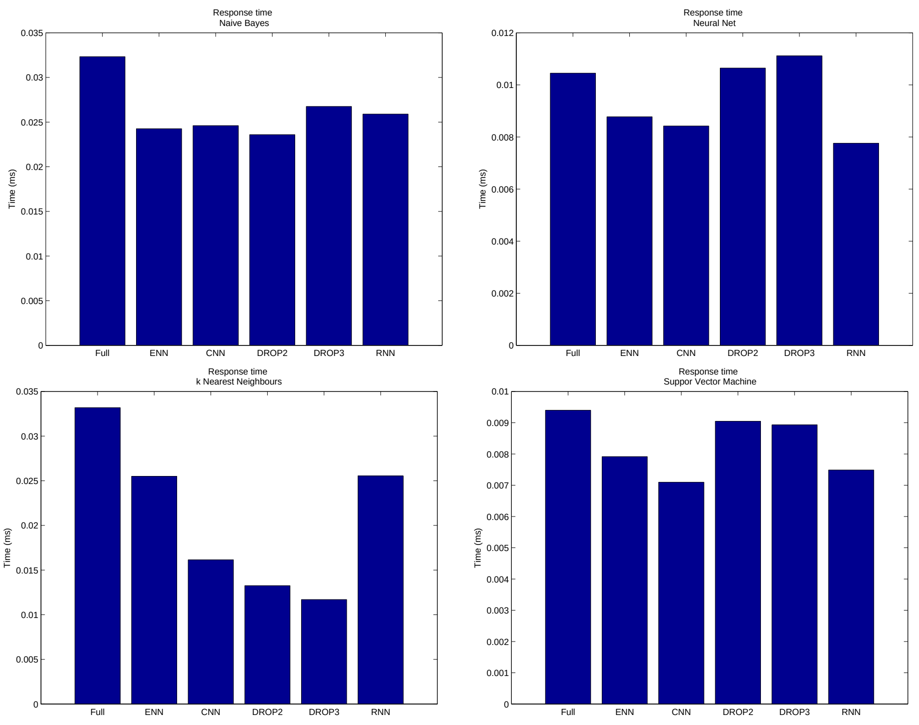


Figure 2.19: Response time (Dataset A.1)

2.4.2.2 Summary

The Decremental Reduction Optimization Procedure (DROP) outperforms other methods on all datasets and in both used measures – model performance on produced subset and the compression rate. DROP has reduced amount of instances to less than one fifth with a small decrease in the classification performance on most of the datasets. ENN on most of the datasets increased the classification performance of models, it confirms that ENN works well as the noise filter.

In Figure 2.18 is obvious strong relation between amount of instances and learning time of some models (especially for Neural Network,SVM) and in Figure 2.19 can be seen the relation between amount of instances in the training set and response time of other models (*k*-NN). The amount of instances in the training set affects the learning and response time of the models. Instance selection methods could rapidly reduce the learning time and the response time of models.

2.4.3 Class balancing

This section presents results of class balancing methods for six classification models (1-nn, 5-nn, 10-nn, 20-nn, neural network and naive bayes) on various datasets. As in the previous section, results on each dataset are discussed separately and at the end of the section is a summary of the results.

Dataset A.2 contains 76 percent instances of the negative class. We are focused on the positive instances therefore we use f-measure as a performance measure. In Figure 2.20 is shown the classification performance of different models. As in previous sections each subfigure corresponds to one classifier and each box-plot to results of particular class balancing method. Class balancing methods increased classification performance on this dataset as shows Figure 2.20. All methods have had similar result, except under-sampling of majority class that has had a bit worse results than others. Increase of performance is greater for k-nn classifiers than for neural network, that has had similar results as k-nn even without class balancing. Nevertheless, over-sampling decreases variance in classification performance for all models.

Dataset A.6 is in comparison with the previous multi-class. Results on Dataset A.6 for different classifiers and class balancing methods are shown in Figure C.12. Figure C.12 shows that class-balancing methods did not increase classification performance. Moreover, under-sampling strongly decreased classification performance of all models.

Another imbalanced dataset is Dataset A.7. Results of class balancing methods on Dataset A.7 for each classifier are shown in Figure C.13. All over-sampling methods including mixed sampling increased classification performance, but no method significantly overcomes other. Increase is greater for k-nn classifiers. Increase for neural network classifier and naive bayes classifier is not significant.

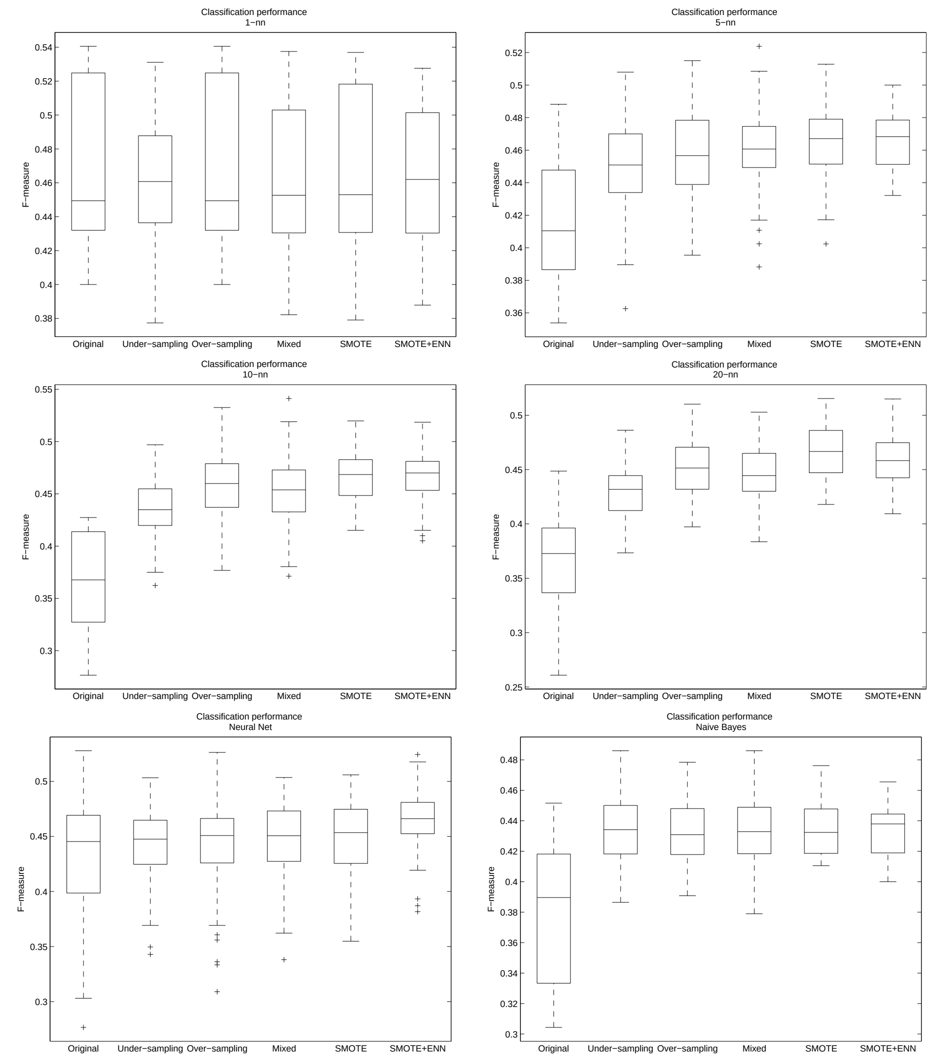


Figure 2.20: Model performance (Dataset A.2)

Dataset A.4 is two-class dataset with 768 instances of which 268 instances are positive. Results for different classifiers and class balancing methods are shown in Figure 2.21. On this dataset all class balancing methods significantly decreased classification performance. Measure of the performance is f-measure. Class balancing methods increased sensitivity of all models as is shown in Figure 2.22, but at the same time strongly decreased precision. It means that model correctly classify positive instance (true positives) but also has a lot of negative instances classified as positive (false positives). In final the performance has been decreased.

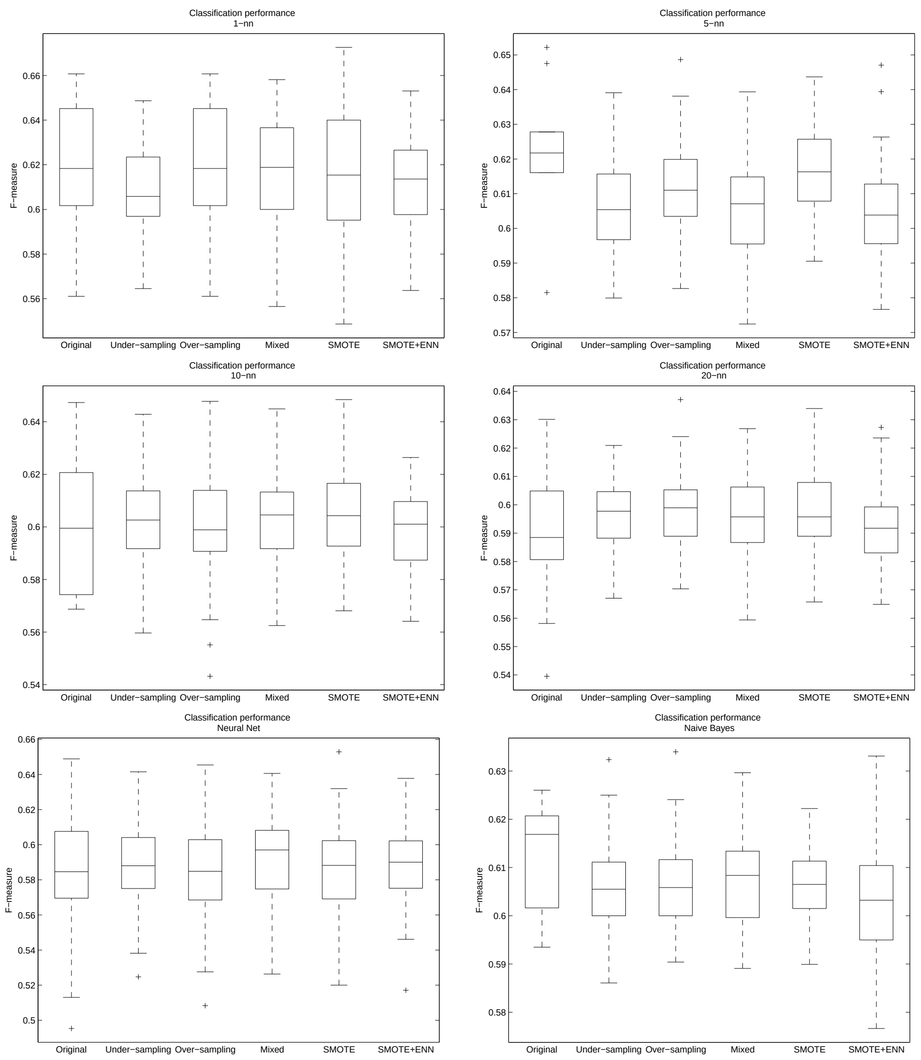


Figure 2.21: Model performance – f-measure (Dataset A.4)

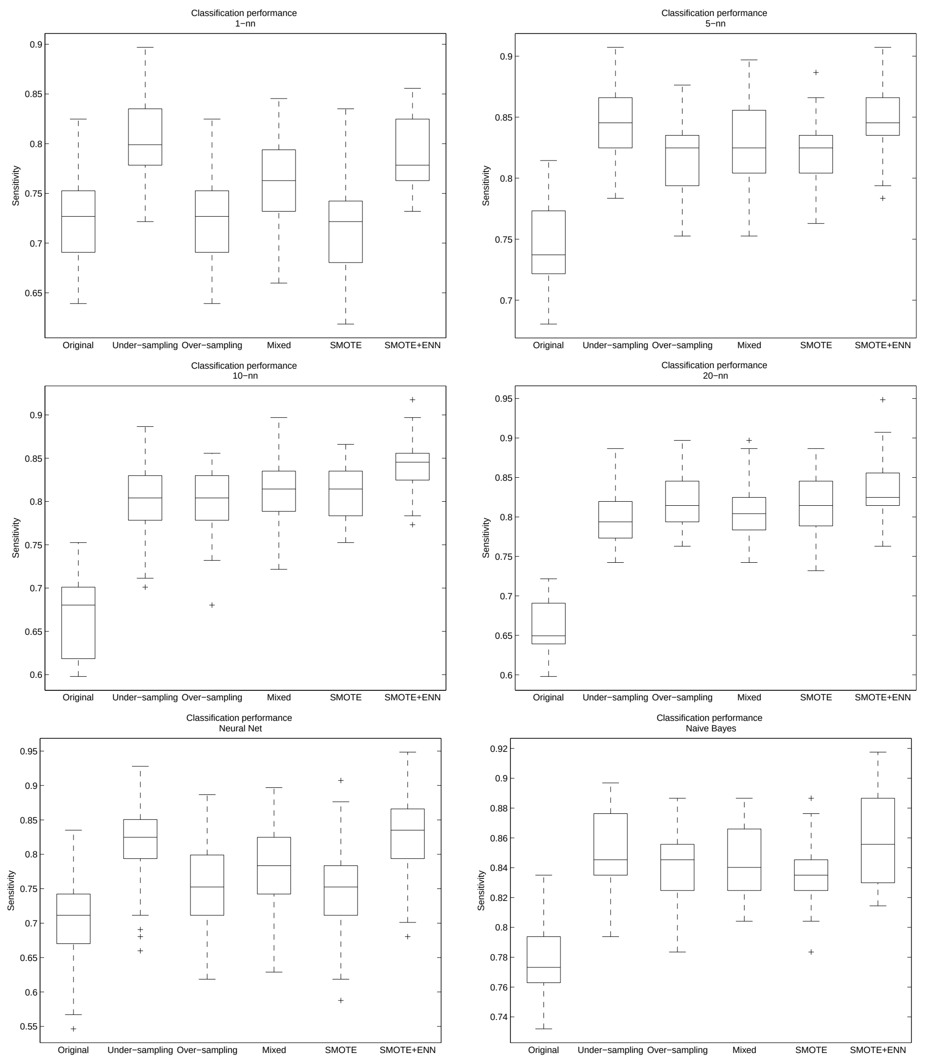


Figure 2.22: Model performance – sensitivity (Dataset A.4)

2.4.3.1 Summary

Class balancing methods can increase the performance of classification models on imbalanced datasets. These methods significantly increases sensitivity of a model on the minority class. With increasing sensitivity is usually decreased precision of positive class classification. It means that is a lot of false positive classifications and overall performance can decrease. No method has had significantly better results than other. Nevertheless, under-sampling of the majority class gives the worst results on all benchmarking datasets.

Conclusion

Several methods for data splitting, instance selection and class balancing, published in literature, were reviewed. All of these methods are very important in processes of construction of training and testing sets. Data splitting methods allow to split a dataset into more subsets in the case of absence of an independent set for model validation or prediction performance assessment. Instance selection methods reduce a training set by removing instances useless for estimating parameters of a model, which can speed up the learning phase and response time, especially for lazy learners. Class balancing algorithms solve the problem of inequality in class distributions in order to increase the performance of learned models.

In the experimental part, several representatives of data splitting, instance selection and class balancing methods were tested and compared. From data splitting methods have had the most stable results over all benchmarking datasets simple random and stratified sampling. More complex methods as CADEX or NN can improve the classification performance on particular datasets, but unfortunately they are often strongly biased. They can not be recommended as a general approach. Instance selection methods can significantly reduce a training set and still reach high performance on unseen data. Amount of instances in a training set has a strong impact on the time consumed by learning a model or on the response time of a model, especially for k-nn. Class balancing methods increase sensitivity of a model to a minority class. However, with increasing sensitivity can be decreased the precision and overall performance may be worse than before class balancing. No class balancing method has had significantly better results than others.

All compared methods have had different results on various datasets. This indicates that methods are strongly domain dependent. Moreover, results of methods are dependent on specific classifiers. Unfortunately, universal approach does not exists.

Bibliography

[1] Borovicka, T.; Jirina Jr., M.; Kordik, P.; etc.: Selecting Representative Data Sets. *Submitted for publication*, 2012.

[2] Duda, R.; Hart, P.: *Pattern classification and scene analysis*. Wiley, 1996.

[3] Brighton, H.; Mellish, C.: Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, volume 6, no. 2, 2002: pp. 153–172.

[4] Pan, F.; Wang, W.; Tung, A.; etc.: Finding representative set from massive data. In *Data Mining, Fifth IEEE International Conference on*, IEEE, 2005, pp. 8–pp.

[5] Olvera-López, J.; Carrasco-Ochoa, J.; Martínez-Trinidad, J.; etc.: A review of instance selection methods. *Artificial Intelligence Review*, volume 34, no. 2, 2010: pp. 133–143.

[6] Hawkins, D.; etc.: The problem of overfitting. *Journal of chemical information and computer sciences*, volume 44, no. 1, 2004: pp. 1–12.

[7] Snee, R.: Validation of regression models: methods and examples. *Technometrics*, 1977: pp. 415–428.

[8] Han, J.; Kamber, M.: *Data mining: concepts and techniques*. The Morgan Kaufmann series in data management systems, Elsevier, 2006, ISBN 9781558609013. Available at WWW: <`http://books.google.cz/books?id=AfL0t-YzOrEC`>

[9] Witten, I.; Frank, E.; Hall, M.: *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

[10] Cohen, G.; Hilario, M.; Sax, H.; etc.: Data imbalance in surveillance of nosocomial infections. *Medical Data Analysis*, 2003: pp. 109–117.

[11] Geman, S.; Bienenstock, E.; Doursat, R.: Neural networks and the bias/variance dilemma. *Neural computation*, volume 4, no. 1, 1992: pp. 1–58.

[12] Dietterich, T.: Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, volume 10, no. 7, 1998: pp. 1895–1923.

[13] Everitt, B.: *The analysis of contingency tables*. Chapman & Hall/CRC, 1992.

[14] Snedecor, G.; Cochran, W.: Statistical methods 6th ed. *IOWA State University press, USA. 450pp*, 1967.

[15] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, volume 7, 2006: pp. 1–30.

[16] Anderson, T.; Darling, D.: A test of goodness of fit. *Journal of the American Statistical Association*, 1954: pp. 765–769.

[17] Kruskal, J.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, volume 29, no. 1, 1964: pp. 1–27.

[18] Stephens, M.: EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 1974: pp. 730–737.

[19] Bentler, P.; Bonett, D.: Significance tests and goodness of fit in the analysis of covariance structures. *Psychological bulletin*, volume 88, no. 3, 1980: p. 588.

[20] Thas, O.: *Comparing distributions*. Springer, 2010.

[21] Bickel, P.: A distribution free version of the Smirnov two sample test in the p-variate case. *The Annals of Mathematical Statistics*, volume 40, no. 1, 1969: pp. 1–23.

[22] Fasano, G.; Franceschini, A.: A multidimensional version of the Kolmogorov-Smirnov test. *Monthly Notices of the Royal Astronomical Society*, volume 225, 1987: pp. 155–170.

[23] Justel, A.; Peña, D.; Zamar, R.: A multivariate Kolmogorov-Smirnov test of goodness of fit. *Statistics & Probability Letters*, volume 35, no. 3, 1997: pp. 251–259.

[24] Rencher, A.: Methods of multivariate analysis. 2002.

[25] Stone, M.: Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1974: pp. 111–147.

[26] Molinaro, A.; Simon, R.; Pfeiffer, R.: Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, volume 21, no. 15, 2005: pp. 3301–3307.

[27] Burman, P.: A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, volume 76, no. 3, 1989: pp. 503–514.

[28] McLachlan, G.; Wiley, J.: *Discriminant analysis and statistical pattern recognition*. Wiley Online Library, 1992.

[29] Geisser, S.: *Predictive inference: An introduction*, volume 55. Chapman & Hall/CRC, 1993.

[30] Tibshirani, R.; Efron, B.: An introduction to the bootstrap. *Monographs on Statistics and Applied Probability*, volume 57, 1993: pp. 1–436.

[31] Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International joint Conference on artificial intelligence*, volume 14, LAWRENCE ERLBAUM ASSOCIATES LTD, 1995, pp. 1137–1145.

[32] Jain, A.; Dubes, R.; Chen, C.: Bootstrap techniques for error estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, , no. 5, 1987: pp. 628–633.

[33] Breiman, L.; Spector, P.: Submodel selection and evaluation in regression. the x-random case. *International Statistical Review/Revue Internationale de Statistique*, 1992: pp. 291–319.

[34] Bailey, T.; Elkan, C.: Estimating the accuracy of learned concepts.". In *Proc. International Joint Conference on Artificial Intelligence*, Citeseer, 1993.

[35] Bickel, P.; Freedman, D.: Some asymptotic theory for the bootstrap. *The Annals of Statistics*, volume 9, no. 6, 1981: pp. 1196–1217.

[36] Andrews, D.: Inconsistency of the bootstrap when a parameter is on the boundary of the parameter space. *Econometrica*, volume 68, no. 2, 2000: pp. 399–405.

[37] Kennard, R.; Stone, L.: Computer aided design of experiments. *Technometrics*, 1969: pp. 137–148.

[38] de Groot, P.; Postma, G.; Melssen, W.; etc.: Selecting a representative training set for the classification of demolition waste using remote NIR sensing. *Analytica Chimica Acta*, volume 392, no. 1, 1999: pp. 67 – 75, ISSN 0003-2670, doi:10.1016/S0003-2670(99)00193-2. Available at WWW: <http://www.sciencedirect.com/science/article/pii/S0003267099001932>

[39] Daszykowski, M.; Walczak, B.; Massart, D.: Representative subset selection. *Analytica Chimica Acta*, volume 468, no. 1, 2002: pp. 91–103.

[40] Thomas H. Cormen, R. L. R. C. S., Charles E. Leiserson: *Introduction to Algorithms*. London, England: The MIT Press, 2009.

[41] Jankowski, N.; Grochowski, M.: Comparison of instances selection algorithms I. Algorithms survey. *Artificial Intelligence and Soft Computing-ICAISC 2004*, 2004: pp. 598–603.

[42] Liu, H.; Motoda, H.: On issues of instance selection. *Data Mining and Knowledge Discovery*, volume 6, no. 2, 2002: pp. 115–130.

[43] Jankowski, N.; Grochowski, M.: Comparison of instances selection algorithms I. Algorithms survey. *Artificial Intelligence and Soft Computing-ICAISC 2004*, 2004: pp. 598–603.

[44] Cover, T.; Hart, P.: Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, volume 13, no. 1, 1967: pp. 21–27.

[45] Grochowski, M.; Jankowski, N.: Comparison of instance selection algorithms II. Results and comments. *Artificial Intelligence and Soft Computing-ICAISC 2004*, 2004: pp. 580–585.

[46] Gates, G.: The reduced nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, volume 18, no. 3, 1972: pp. 431–433.

[47] Ritter, G.; Woodruff, H.; Lowry, S.; etc.: An algorithm for a selective nearest neighbor decision rule (Corresp.). *Information Theory, IEEE Transactions on*, volume 21, no. 6, 1975: pp. 665–669.

[48] Chou, C.; Kuo, B.; Chang, F.: The generalized condensed nearest neighbor rule as a data reduction method. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, IEEE, 2006, pp. 556–559.

[49] Wilson, D.: Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, volume 2, no. 3, 1972: pp. 408–421.

[50] Tomek, I.: An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, , no. 6, 1976: pp. 448–452.

[51] Devijver, P.; Kittler, J.: On the edited nearest neighbor rule. In *Proc. 5th Int. Conf. on Pattern Recognition*, 1980, pp. 72–80.

[52] Vázquez, F.; Sánchez, J.; Pla, F.: A stochastic approach to Wilson's editing algorithm. *Pattern Recognition and Image Analysis*, 2005: pp. 35–42.

[53] Aha, D.; Kibler, D.; Albert, M.: Instance-based learning algorithms. *Machine learning*, volume 6, no. 1, 1991: pp. 37–66.

[54] Wilson, D.; Martinez, T.: Reduction techniques for instance-based learning algorithms. *Machine learning*, volume 38, no. 3, 2000: pp. 257–286.

[55] Zhao, K.; Zhou, S.; Guan, J.; etc.: C-pruner: An improved instance pruning algorithm. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 1, IEEE, 2003, pp. 94–99.

[56] Vapnik, V.: *The nature of statistical learning theory*. Springer Verlag, 2000.

[57] Li, Y.; Hu, Z.; Cai, Y.; etc.: Support vector based prototype selection method for nearest neighbor rules. *Advances in Natural Computation*, 2005: pp. 408–408.

[58] Srisawat, A.; Phienthrakul, T.; Kijsirikul, B.: SV-kNNC: an algorithm for improving the efficiency of k-nearest neighbor. *PRICAI 2006: Trends in Artificial Intelligence*, 2006: pp. 975–979.

[59] García, S.; Cano, J.; Herrera, F.: A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, volume 41, no. 8, 2008: pp. 2693–2709.

[60] Kuncheva, L.: Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, volume 16, no. 8, 1995: pp. 809–814.

[61] Kuncheva, L.; Bezdek, J.: Nearest prototype classification: Clustering, genetic algorithms, or random search? *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, volume 28, no. 1, 1998: pp. 160–164.

[62] Bezdek, J.; Kuncheva, L.: Nearest prototype classifier designs: An experimental study. *International Journal of Intelligent Systems*, volume 16, no. 12, 2001: pp. 1445–1473.

[63] Cano, J.; Herrera, F.; Lozano, M.: Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *Evolutionary Computation, IEEE Transactions on*, volume 7, no. 6, 2003: pp. 561–575.

[64] Glover, F.; McMillan, C.: The general employee scheduling problem. An integration of MS and AI. *Computers & operations research*, volume 13, no. 5, 1986: pp. 563–573.

[65] Cerverón, V.; Ferri, F.: Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, volume 31, no. 3, 2001: pp. 408–413.

[66] Zhang, H.; Sun, G.: Optimal reference subset selection for nearest neighbor classification by tabu search. *Pattern Recognition*, volume 35, no. 7, 2002: pp. 1481–1490.

[67] Riquelme, J.; Aguilar-Ruiz, J.; Toro, M.: Finding representative patterns with ordered projections. *Pattern Recognition*, volume 36, no. 4, 2003: pp. 1009–1018.

[68] Raicharoen, T.; Lursinsap, C.: A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm. *Pattern recognition letters*, volume 26, no. 10, 2005: pp. 1554–1567.

[69] Narayan, B.; Murthy, C.; Pal, S.: Maxdiff kd-trees for data condensation. *Pattern recognition letters*, volume 27, no. 3, 2006: pp. 187–200.

[70] Friedman, J.; Bentley, J.; Finkel, R.: An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, volume 3, no. 3, 1977: pp. 209–226.

[71] Caises, Y.; González, A.; Leyva, E.; etc.: SCIS: combining instance selection methods to increase their effectiveness over a wide range of domains. In *Proceedings of the 10th international conference on Intelligent data engineering and automated learning*, Springer-Verlag, 2009, pp. 17–24.

[72] Veenman, C.; Reinders, M.: The Nearest Sub-class Classifier: a Compromise between the Nearest Mean and Nearest Neighbor Classifier. *Transactions on PAMI*, volume 27, no. 9, 2005: pp. 1417–1429.

[73] Veenman, C.; Reinders, M.; Backer, E.: A maximum variance cluster algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, volume 24, no. 9, 2002: pp. 1273–1280.

[74] Arturo Olvera-López, J.; Ariel Carrasco-Ochoa, J.; Francisco Martínez-Trinidad, J.: Object selection based on clustering and border objects. *Computer Recognition Systems 2*, 2007: pp. 27–34.

[75] Paredes, R.; Vidal, E.: Weighting prototypes-a new editing approach. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, IEEE, 2000, pp. 25–28.

[76] Olvera-López, J.; Carrasco-Ochoa, J.; Martínez-Trinidad, J.: Prototype selection via prototype relevance. *Progress in Pattern Recognition, Image Analysis and Applications*, 2008: pp. 153–160.

[77] Kotsiantis, S.; Kanellopoulos, D.; Pintelas, P.: Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, volume 30, no. 1, 2006: pp. 25–36.

[78] Guo, X.; Yin, Y.; Dong, C.; etc.: On the class imbalance problem. In *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, volume 4, IEEE, 2008, pp. 192–201.

[79] Japkowicz, N.; Stephen, S.: The class imbalance problem: A systematic study. *Intelligent data analysis*, volume 6, no. 5, 2002: pp. 429–449.

[80] Laurikkala, J.: Improving identification of difficult small classes by balancing class distribution. *Artificial Intelligence in Medicine*, 2001: pp. 63–66.

[81] Kubat, M.; Matwin, S.: Addressing the curse of imbalanced training sets: one-sided selection. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, MORGAN KAUFMANN PUBLISHERS, INC., 1997, pp. 179–186.

[82] Chawla, N.; Bowyer, K.; Hall, L.; etc.: SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, volume 16, 2002: pp. 321–357.

[83] Han, H.; Wang, W.; Mao, B.: Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. *Advances in Intelligent Computing*, 2005: pp. 878–887.

[84] Weiss, G.: *The effect of small disjuncts and class distribution on decision tree learning*. Dissertation thesis, Rutgers, The State University of New Jersey, 2003.

[85] Weiss, G.; Provost, F.: Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Intell. Res. (JAIR)*, volume 19, 2003: pp. 315–354.

[86] Kotsiantis, S.; Pintelas, P.: Mixture of expert agents for handling imbalanced data sets. *Annals of Mathematics, Computing & Teleinformatics*, volume 1, no. 1, 2003: pp. 46–55.

[87] Batista, G.; Prati, R.; Monard, M.: A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, volume 6, no. 1, 2004: pp. 20–29.

[88] Drummond, C.; Holte, R.; etc.: C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Datasets II*, Citeseer, 2003.

[89] Barandela, R.; Sánchez, J.; Garcia, V.; etc.: Strategies for learning in class imbalance problems. *Pattern Recognition*, volume 36, no. 3, 2003: pp. 849–851.

[90] Wu, G.; Chang, E.: Class-boundary alignment for imbalanced dataset learning. In *ICML 2003 workshop on learning from imbalanced data sets II*, 2003, pp. 49–56.

[91] Veropoulos, K.; Campbell, C.; Cristianini, N.: Controlling the sensitivity of support vector machines. In *Proceedings of the international joint conference on artificial intelligence*, volume 1999, Citeseer, 1999, pp. 55–60.

[92] Japkowicz, N.: Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, volume 42, no. 1, 2001: pp. 97–122.

[93] Schölkopf, B.; Platt, J.; Shawe-Taylor, J.; etc.: Estimating the support of a high-dimensional distribution. *Neural computation*, volume 13, no. 7, 2001: pp. 1443–1471.

[94] Manevitz, L.; Yousef, M.: One-class SVMs for document classification. *The Journal of Machine Learning Research*, volume 2, 2002: pp. 139–154.

[95] Raskutti, B.; Kowalczyk, A.: Extreme re-balancing for SVMs: a case study. *ACM Sigkdd Explorations Newsletter*, volume 6, no. 1, 2004: pp. 60–69.

[96] Elkan, C.: The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, volume 17, LAWRENCE ERLBAUM ASSOCIATES LTD, 2001, pp. 973–978.

[97] Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 1999, pp. 155–164.

[98] Fan, W.; Stolfo, S.; Zhang, J.; etc.: AdaCost: misclassification cost-sensitive boosting. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, Citeseer, 1999, pp. 97–105.

[99] Freund, Y.; Schapire, R.: A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, Springer, 1995, pp. 23–37.

[100] Breiman, L.: Bagging predictors. *Machine learning*, volume 24, no. 2, 1996: pp. 123–140.

[101] Schapire, R.: The strength of weak learnability. *Machine learning*, volume 5, no. 2, 1990: pp. 197–227.

[102] Wolpert, D.: Stacked generalization*. *Neural networks*, volume 5, no. 2, 1992: pp. 241–259.

[103] Joshi, M.; Kumar, V.; Agarwal, R.: Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, IEEE, 2001, pp. 257–264.

[104] Chawla, N.; Lazarevic, A.; Hall, L.; etc.: SMOTEBoost: Improving prediction of the minority class in boosting. *Knowledge Discovery in Databases: PKDD 2003*, 2003: pp. 107–119.

[105] Phua, C.; Alahakoon, D.; Lee, V.: Minority report in fraud detection: classification of skewed data. *ACM SIGKDD Explorations Newsletter*, volume 6, no. 1, 2004: pp. 50–59.

[106] Frank, A.; Asuncion, A.: .

List of Abbreviations

$Acc(X)$	Classification accuracy obtained using X as a training set. , page 2
R	Training set , page 2
S	Sample set , page 2
S^*	Representative set , page 2
T	Testing set , page 2
V	Validation set , page 2
CNN	Condensed Nearest Neighbour, page 19
DROP1-5	Decremental Reduction Optimization Procedure, page 21
EA	Evolutionary Algorithms, page 21
ENN	Edited Nearest Neighbour, page 20
FN	False Negatives, page 5
FP	False Positives, page 5
GCM	Generalized-Modified Chang algorithm, page 22
GCNN	Generalized Condensed Nearest Neighbour, page 20
IB1-3	Instance Based Methods, page 21
ICF	Iterative Case Filtering, page 21
LOOCV	Leave-one-out cross-validation, page 14
MSE	Mean Squared Error, page 8
NCL	Neighbourhood Cleaning Rule , page 23
NN	Nearest Neighbour, page 18
NSB	Nearest Sub-class Classifier method, page 22

OSC	Object Selection by Clustering, page 22
OSPC	Over-sampling Algorithm Based on Preliminary Classification , page 25
OSS	One-side Sampling , page 23
POC-NN	Pair Opposite Class-Nearest Neighbour, page 22
POP	Pattern by Ordered Projections, page 21
PSR	Prototype Selection by Relevance , page 22
RENN	Repeated Edited Nearest Neighbour, page 20
RNN	Reduced Nearest Neighbour, page 19
SMOTE	Synthetic Minority Over-sampling Technique , page 24
SNN	Selective Nearest Neighbour, page 20
SVM	Support Vector Machine, page 21
TN	True Negatives, page 5
TP	True Positives, page 5
TS	Tabu Search, page 21
WP	Weighting prototype, page 22

APPENDIX A

Description of Datasets

A.1 Haberman's Survival Data Set

`http://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival`

A.1.1 Data Set Characteristics

Data Set Characteristics: Multivariate Number of Instances: 306 Attribute Characteristics: Integer Number of Attributes: 3 Date Donated 1999-03-04 Associated Tasks: Classification Missing Values?: No

A.1.2 Data Set Information:

The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

A.1.3 Attribute Information:

1. Age of patient at time of operation (numerical)
2. Patient's year of operation (year - 1900, numerical)
3. Number of positive axillary nodes detected (numerical)
4. Survival status (class attribute)
   - 1 = the patient survived 5 years or longer
   - 2 = the patient died within 5 year

A.2 Blood Transfusion Service Center Data Set

`http://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center`

Whether he/she donated blood in March 2007 binary 1=yes 0=no Output 0 1 1 (24

A.2.1 Data Set Information

To demonstrate the RFMTC marketing model (a modified version of RFM), this study adopted the donor database of Blood Transfusion Service Center in Hsin-Chu City in Taiwan. The center passes their blood transfusion service bus to one university in Hsin-Chu City to gather blood donated about every three months. To build a FRMTC model, we selected 748 donors at random from the donor database. These 748 donor data, each one included R (Recency - months since last donation), F (Frequency - total number of donation), M (Monetary - total blood donated in c.c.), T (Time - months since first donation), and a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

A.2.2 Attribute Information

1. R (Recency - months since last donation),
2. F (Frequency - total number of donation),
3. M (Monetary - total blood donated in c.c.),
4. T (Time - months since first donation), and
5. a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

A.3 Wine Data Set

`http://archive.ics.uci.edu/ml/datasets/Wine`
Imbalanced Instances. 178 Features. 13 Classes. 3

A.3.1 Data Set Information

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

A.3.2 Attribute Information:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

A.4 Pima Indians Diabetes Data Set

768 instances 268 positive instance `http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`

A.4.1 Data Set Information

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm; see the paper for details.

A.4.2 Attribute Information

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index ($weightinkg/(heightinm)^2$)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

A.5 Mammographic Mass Data Set

`http://archive.ics.uci.edu/ml/datasets/Mammographic+Mass`
Class Distribution: benign: 516; malignant: 445

A.5.1 Data Set Information

Mammography is the most effective method for breast cancer screening available today. However, the low positive predictive value of breast biopsy resulting from mammogram interpretation leads to approximately 70% unnecessary biopsies with benign outcomes. To reduce the high number of unnecessary breast biopsies, several computer-aided diagnosis (CAD) systems have been proposed in the last years.These systems help physicians in their decision to perform a breast biopsy on a suspicious lesion seen in a mammogram or to perform a short term follow-up examination instead. This data set can be used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BI-RADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the Institute of Radiology of the University Erlangen-Nuremberg between 2003 and 2006. Each instance has an associated BI-RADS assessment ranging from 1 (definitely benign) to 5 (highly suggestive of malignancy) assigned in a double-review process by physicians. Assuming that all cases with BI-RADS assessments greater or equal a given value (varying from 1 to 5), are malignant and the other cases benign, sensitivities and associated specificities can be calculated. These can be an indication of how well a CAD system performs compared to the radiologists.

A.5.2 Attribute Information

1. BI-RADS assessment: 1 to 5 (ordinal, non-predictive!)
2. Age: patient's age in years (integer)
3. Shape: mass shape: round=1 oval=2 lobular=3 irregular=4 (nominal)
4. Margin: mass margin: circumscribed=1 microlobulated=2 obscured=3 ill-defined=4 spiculated=5 (nominal)
5. Density: mass density high=1 iso=2 low=3 fat-containing=4 (ordinal)
6. Severity: benign=0 or malignant=1 (binominal, goal field!)

A.6 Ecoli Data Set

`http://archive.ics.uci.edu/ml/datasets/Ecoli`

A.6.1 Data Set Information:

Data giving characteristics of each ORF (potential gene) in the E. coli genome. Sequence, homology (similarity to other genes) and structural information, and function (if known) are provided.

A.6.2 Attribute Information

1. mcg: McGeoch's method for signal sequence recognition.
2. gvh: von Heijne's method for signal sequence recognition.
3. lip: von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.
4. chg: Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.
5. aac: score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.
6. alm1: score of the ALOM membrane spanning region prediction program.
7. alm2: score of ALOM program after excluding putative cleavable signal regions from the sequence.

A.7 Glass Identification Data Set

`http://archive.ics.uci.edu/ml/datasets/Glass+Identification`

A.7.1 Data Set Information

The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

A.7.2 Attribute Information

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass

A.8 Heart Disease Data Set

`http://archive.ics.uci.edu/ml/datasets/Heart+Disease`

A.8.1 Data Set Information

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The ”goal” field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

This dataset is a heart disease database similar to a database already present in the repository (Heart Disease databases) but in a slightly different form. The task is to detect the absence (1) or presence (2) of heart disease.

A.8.2 Attribute Information

Only 14 attributes used:

1. #3 (age)
2. #4 (sex)
3. #9 (cp)
4. #10 (trestbps)
5. #12 (chol)
6. #16 (fbs)
7. #19 (restecg)
8. #32 (thalach)
9. #38 (exang)
10. #40 (oldpeak)
11. #41 (slope)
12. #44 (ca)
13. #51 (thal)
14. #58 (num) (the predicted attribute)

APPENDIX B

Description of Implementation

B.1 RapidMiner

For experimental purposes was used RapidMiner. RapidMiner is the world's-leading open-source system for data mining. RapidMiner is a modular system that allows to developers and researchers to implement their own solutions based on it. Modular concept of RapidMiner is based on individual operators with given purpose, that are connected in complex chains in order to design complete data minning processes. RapidMiner allows to be extended by user-developed operators that are deployed as plug-ins. All source codes , built operators, built RapidMiner and RapidMiner processes are included in attached CD.

B.1.1 Operators

Several plug-ins were developed for experimental purposes. List of operators is bellow. Implemented operators are highlighted in bold.
One operator was developed in order to data pre-processing:

- **Label remapping**

For experiments with data splitting methods, two operators were developed and four built-in operators were used:

- **CADEX**
- **Nearest neighbour split**
- Random sampling

- Stratified sampling
- Cross-validation
- Bootstrapping

Four operators were developed for experiments with instance selection methods:

- **ENN** – Edited Nearest Neighbour
- **CNN** – Condensed Nearest Neighbour
- **RNN** – Reduces Nearest Neighbour
- **DROP2-3** – Decremental Reduction Optimization Procedure

For experiments with class-balancing methods were developed four operators:

- **Random under-sampling**
- **Random over-sampling**
- **Mixed sampling**
- **SMOTE** – Synthetic Minority Over-sampling Technique

B.1.2 Processes

Several RapidMiner processes were designed in order to perform proposed experiments.

Data splitting (`splitting.*.rmp`)
Processes used to split input dataset into subsets by chosen data splitting methods. Process is repeated by a given number of iteration.

Instance selection (`resampling.rmp`)
Process applies implemented instance selection algorithms on a given sets and produces reduced sets.

Class balancing (`balancing.rmp`)
Process applies implemented class balancing algorithms on a given sets and produces balanced sets.

Performance (`performance.*.rmp`)
Process used for evaluation of methods, it measures a performance of several models on given sets.

Execution time (`execution_time.rmp`)
Process measures learning and response time.

B.2 Matlab

Matlab was used for processing of a results and their graphical representation. All Matlab scripts are included in attached CD. Matlab scripts and their function are described here:

Data splitting (`Splitting.m`)
: Script used for processing of results of data splitting methods. Script also generates images for results by given parameters.

Instance selection (`Resampling.m`)
: Script used for processing of results of instance selection methods. Script also generates images for results by given parameters.

Class balancing (`Balancing.m`)
: Script used for processing of results of class balancing. Script also generates images for results by given parameters.

Paired t-test (`Paired_ttest.m`)
: Script used to verify significance of improvement.

APPENDIX C

Experimental Results

C.1 Data splitting

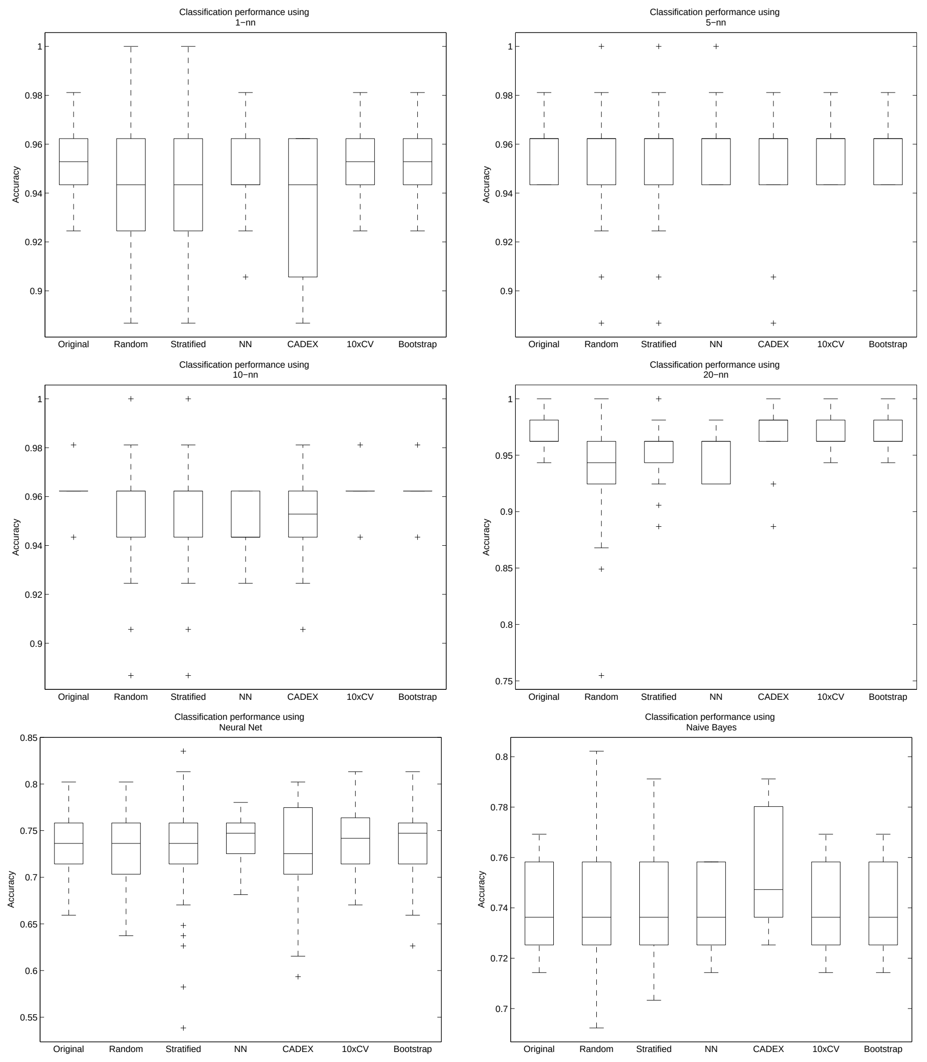


Figure C.1: Model performance (Dataset A.3)

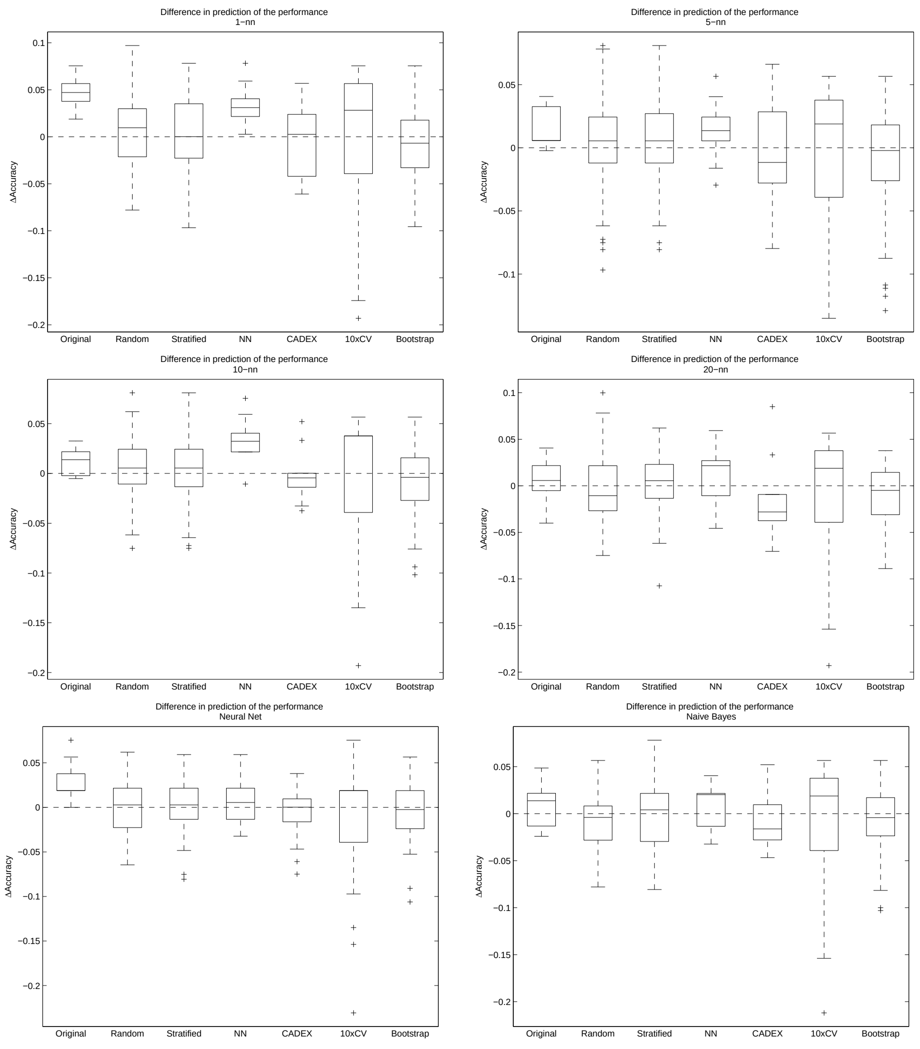


Figure C.2: Difference in the prediction of model performance (Dataset A.3)

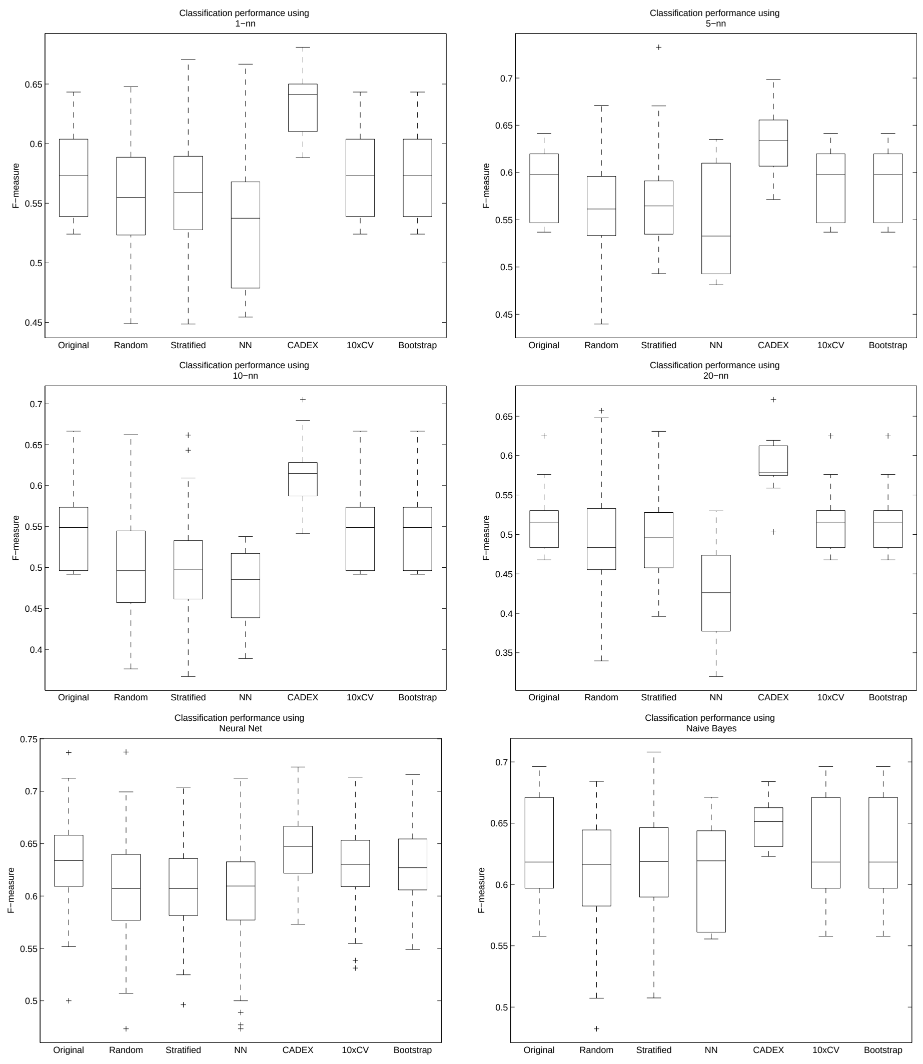


Figure C.3: Model performance in dependence on splitting algorithm (Dataset A.4)

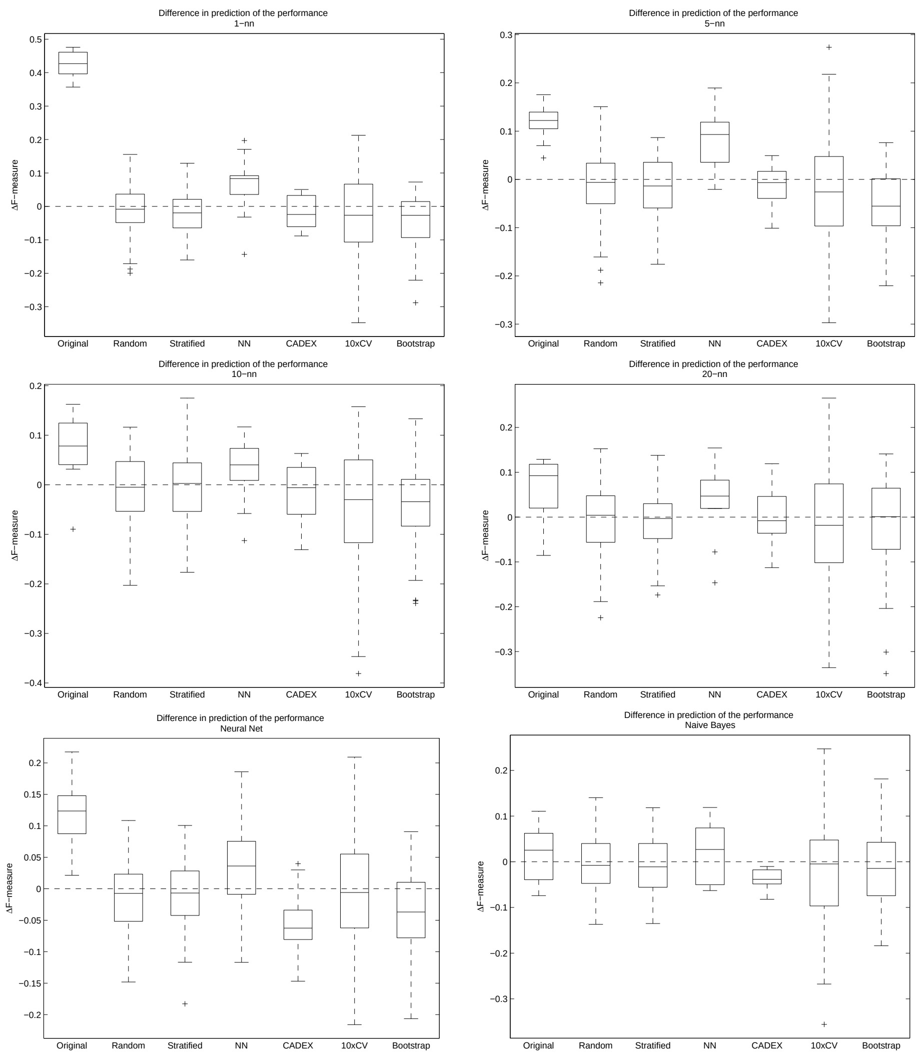


Figure C.4: Classifier performance (Dataset A.4)

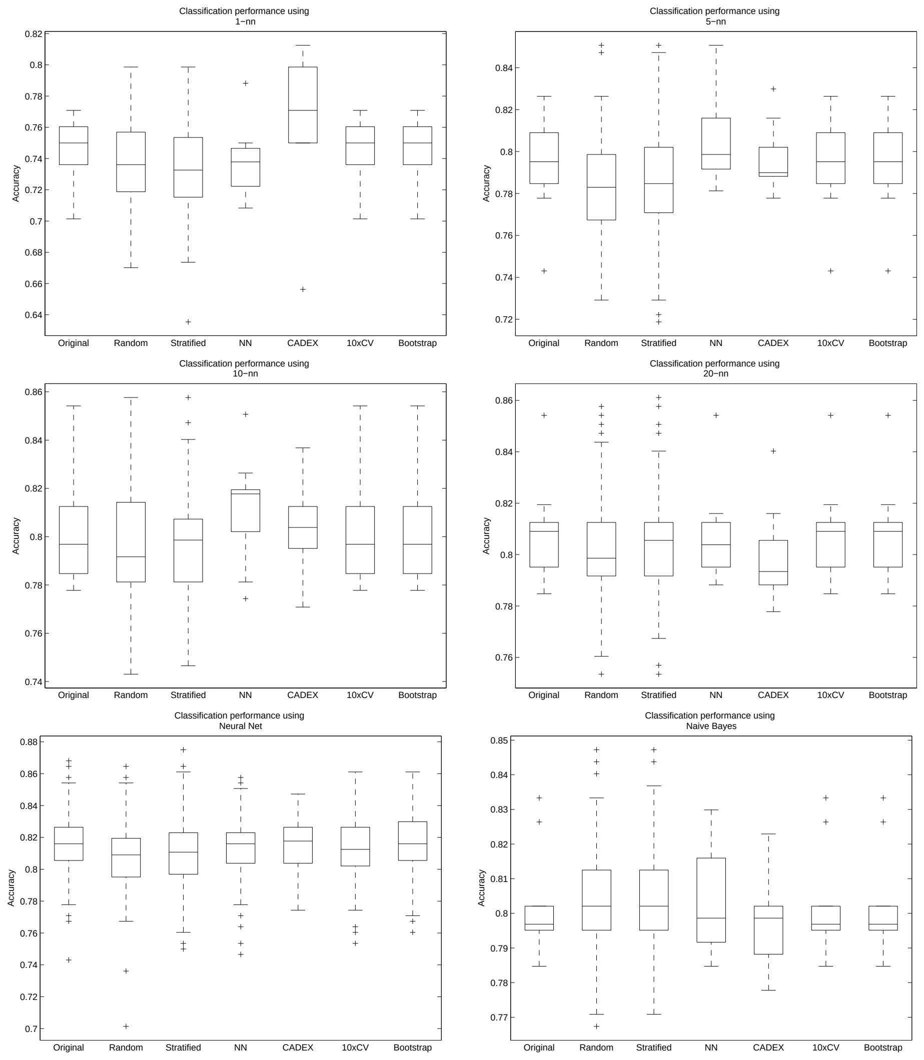


Figure C.5: Model performance (Dataset A.5)

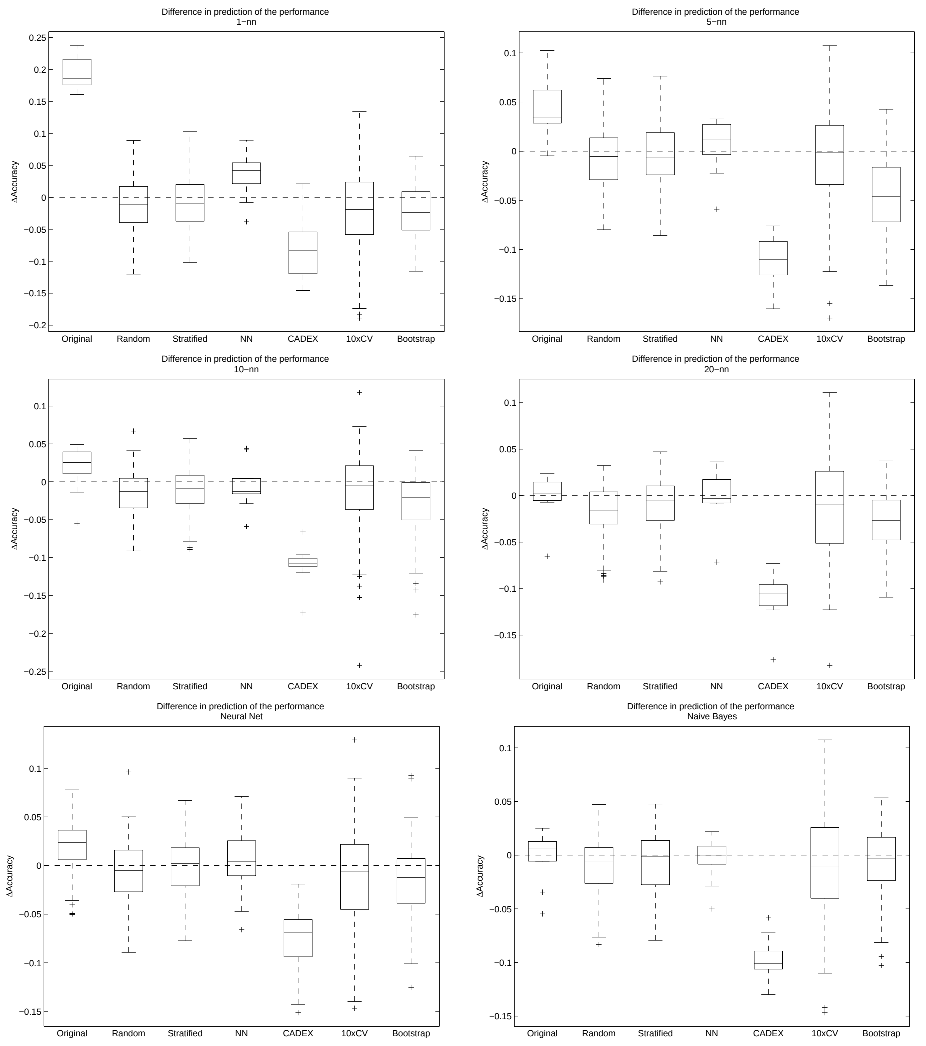


Figure C.6: Difference in the prediction of model performance (Dataset A.5)

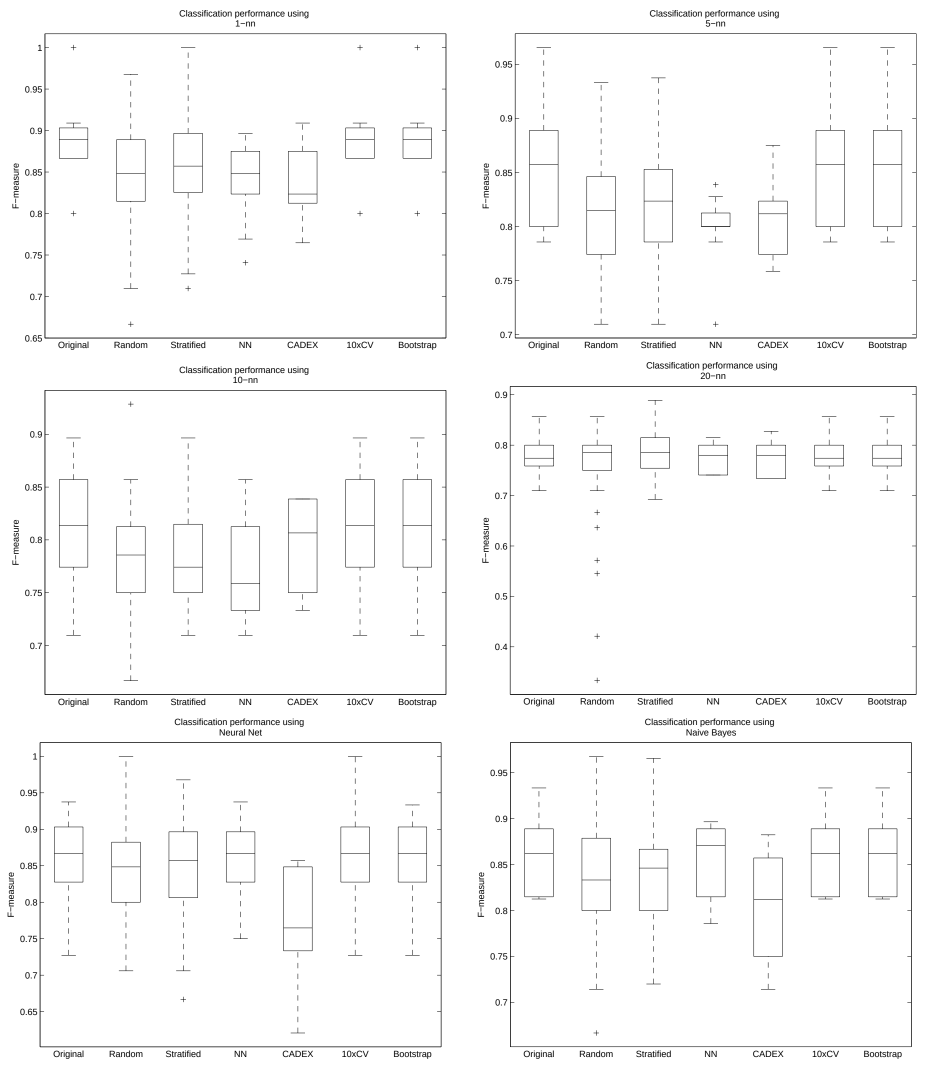


Figure C.7: Model performance in dependence on splitting algorithm (Dataset A.7)

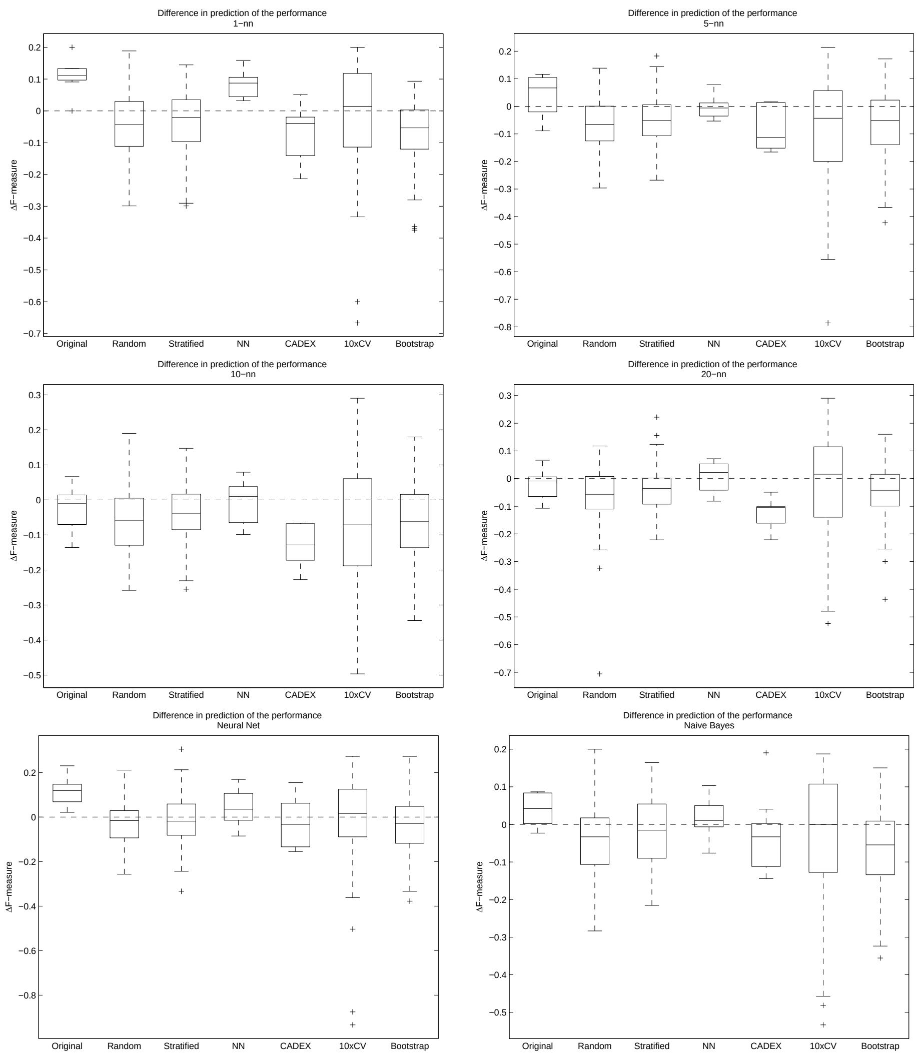


Figure C.8: Difference in the prediction of model performance (Dataset A.7)

C.2 Instance selection

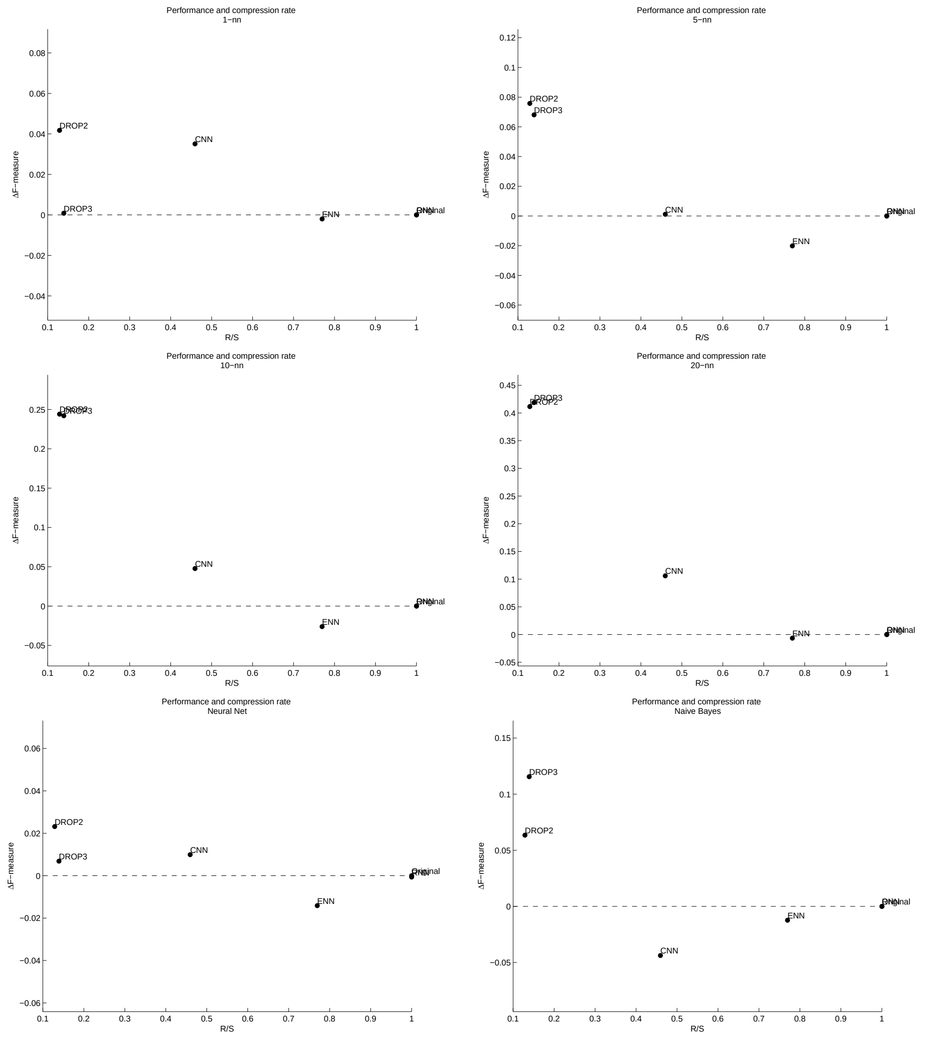


Figure C.9: Model performance and method compression rate (Dataset A.2)

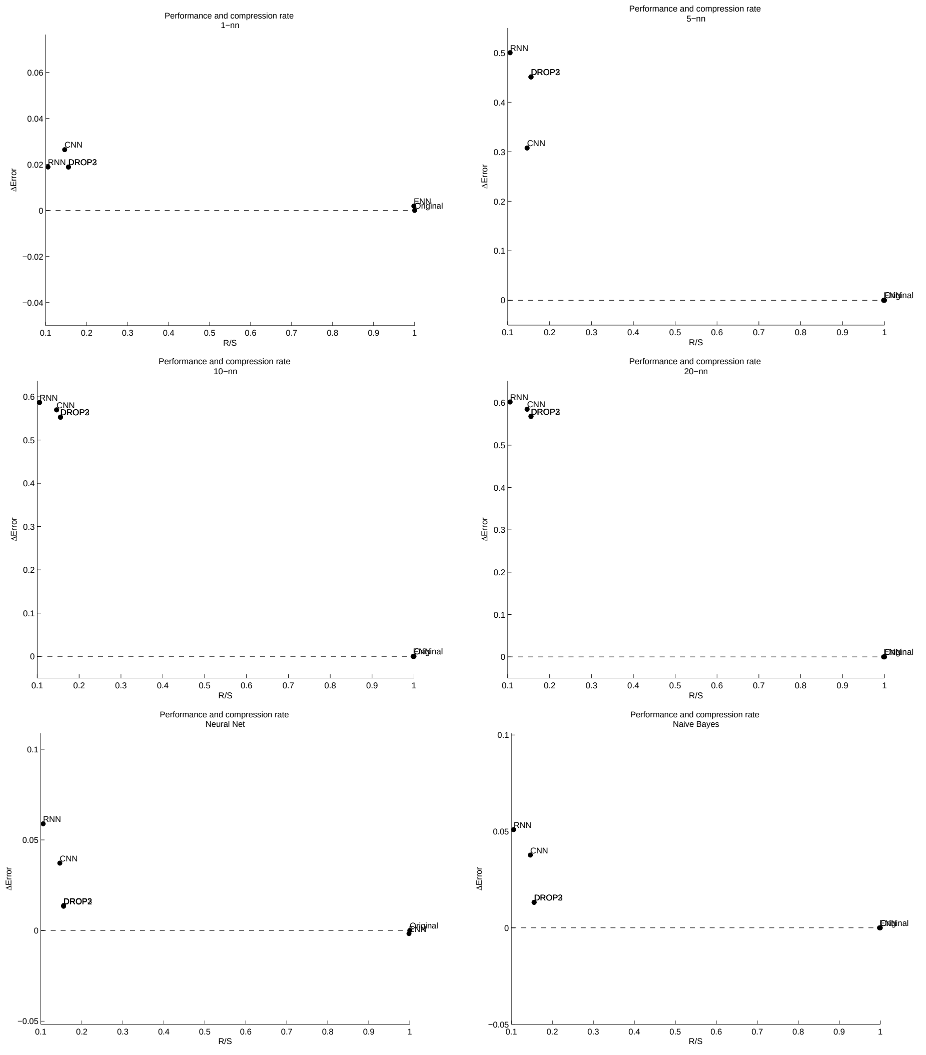


Figure C.10: Model performance and method compression rate (Dataset A.3)

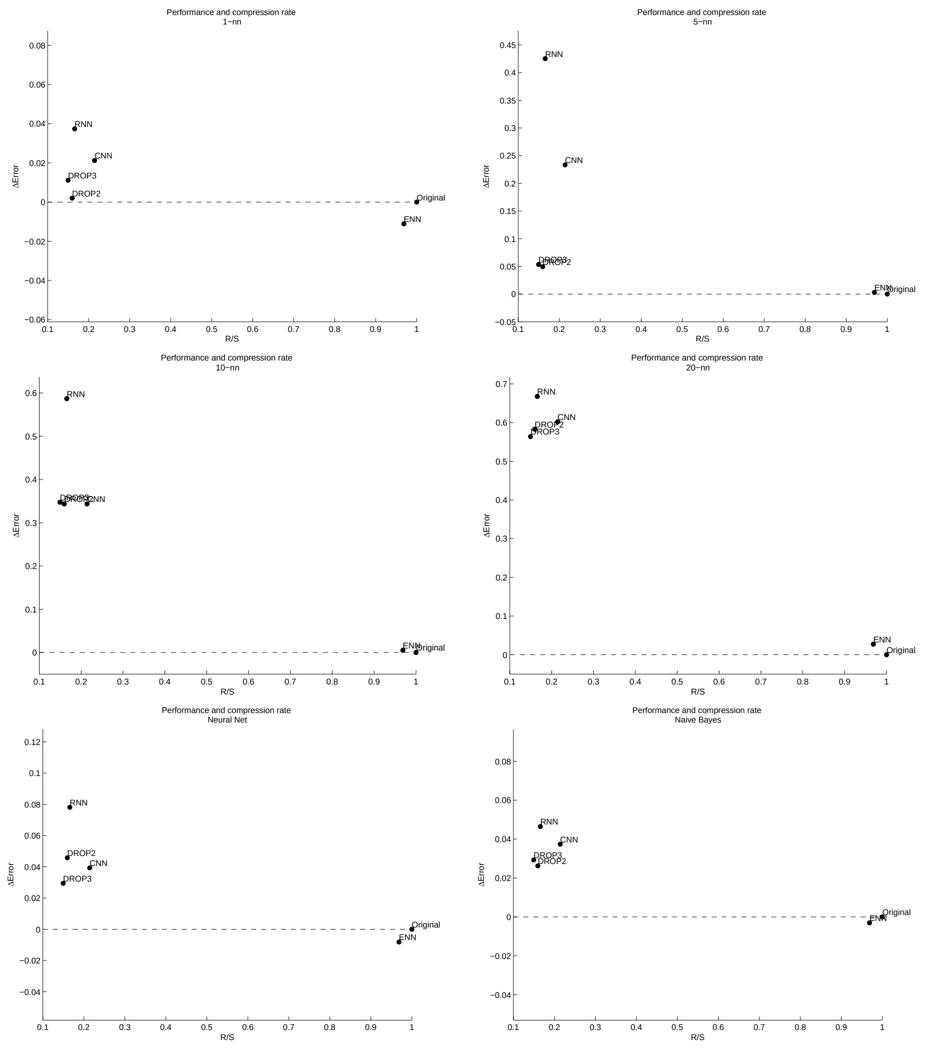


Figure C.11: Model performance and method compression rate (Dataset A.6)

C.3 Class balancing

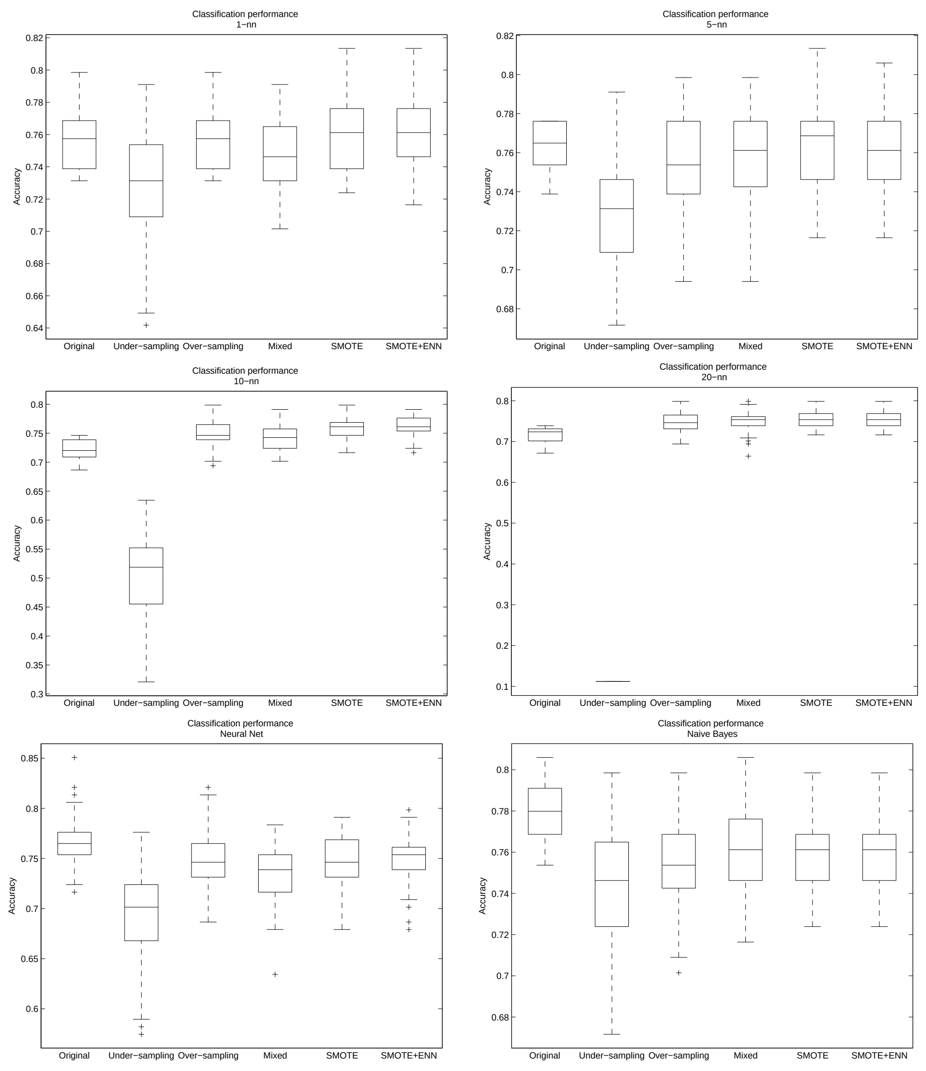


Figure C.12: Model performance (Dataset A.6)

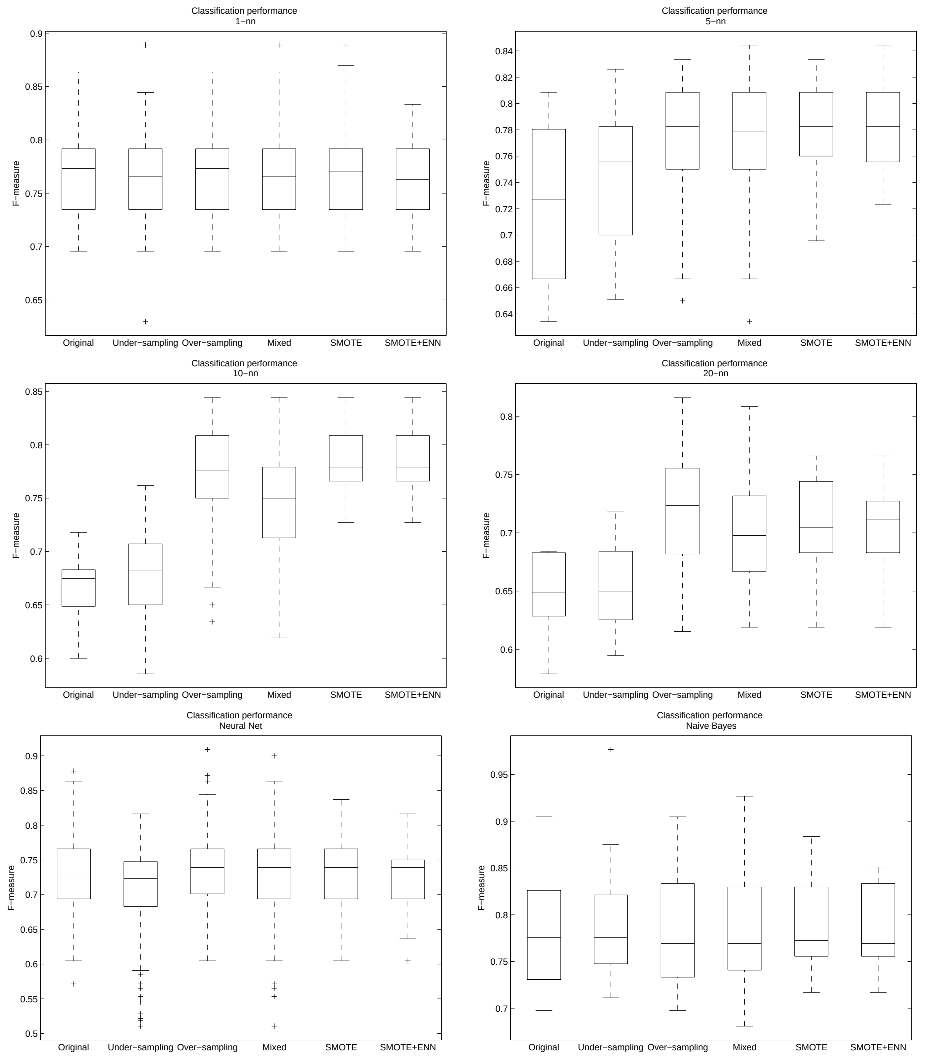


Figure C.13: Model performance (Dataset A.7)

APPENDIX D

Content of CD

```
readme.txt ........................ the file with CD content description
data ........................................... the data files directory
    datasets .................... the directory of datasets of experiments
    results ...................... the directory of results of experiments
exe ........................... the directory with executable programs
    rapidminer ................. the directory with compiled RapidMiner
        scripts ............... the directory with RapidMiner run scripts
            rapidminer.bat ........ the RapidMiner executable (Windows)
            rapidminer ............... the RapidMiner executable (UNIX)
src ....................................... the directory of source codes
    rapidminer ................. the directory with RapidMiner sources
        RapidMiner_Vega ............ the directory with program sources
        RapidMiner_Extensions ....... the directory with plug-ins sources
        rapidminer_processes .......... the directory with processes files
            *.rmp ........................ the RapidMiner processes files
    matlab .......................... the directory with matlab scripts
        *.m ..................................... the matlab scripts
    thesis ............. the directory of LaTeX source codes of the thesis
        figures ............................ the thesis figures directory
        *.tex ................... the LaTeX source code files of the thesis
text ........................................ the thesis text directory
    thesis.pdf ..................... the Master's thesis in PDF format
```