Academic Year 2014/2015

Warsaw University of Technology
Faculty of Electronics and Information Technology
Electrical and Computer Engineering

MASTER OF SCIENCE THESIS

Krzysztof Zieliński

Classification of Sound Events with Robust Soft Learning Vector Quantization

Supervisor
Piotr Salata, MSc

Evaluation: .....

.....

Signature of the Head
of Examination Committee

Electrical and Computer Engineering

Date of Birth: 1989.06.28

Starting Date of Studies: 2012.10.01

Curriculum Vitae

.....
Signature of the Student

Master of Science Examination

Examination was held on: .....20__

With the result: .....

Final Result of the Studies: .....

Suggestions and Remarks of the B.Sc. Examination Committee: .....

.....

.....

SUMMARY

The purpose of this study was to compare three currently popular Classification algorithms on real-world sound events audio samples for both multi and two-class classification cases. All three algorithms: k-Nearest Neighbour (kNN), Robust Soft Learning Vector Quantization (RSLVQ) and Support Vector Machine (SVM) had to be benchmarked in the same test framework. To provide reliable results Nested k-Fold Stratified Cross-Validation was implemented. The choice of performance metrics was crucial on account of largely imbalanced classes. For One-vs.-Rest multiclass extension the performance of RSLVQ turned out to be almost as good as SVM. However, prediction time for RSLVQ was significantly smaller then for SVM what makes RSLVQ a good candidate for real-time classification solutions based on sound events.

Keywords:

RSLVQ, One-vs.-Rest, Multiclass, Sound Events

Klasyfikacja Zdarzeń Dźwiękowych przy użyciu Robust Soft Learning Vector Quantization

Celem tej pracy było porównanie trzech aktualnie popularnych algorytmów klasyfikacji na rzeczywistych próbkach audio zdarzeń dźwiękowych dla multi i dwuklasowej odmiany klasyfikacji. Wszystkie trzy algorytmy: k-Nearest Neighbour (kNN), Robust Soft Learning Vector Quantization (RSLVQ) i Support Vector Machine (SVM) musiały zostać przetestowane w ramach tego samego frameworku. Zaimplementowano Nested k-Fold Stratified Cross-Validation, co powinno zapewnić bardziej wiarygodne wyniki. Wybór sposobu obliczania wydajności algorytmu był istotny w związku z dużą dysproporcją liczności klas. Wydajność algorytmu RSLVQ dla One-vs.-Rest multiklasyfikacji okazała się prawie tak samo wysoka, jak wydajność algorytmu SVM. Niemniej jednak czas predykcji dla RSLVQ był znacząco krótszy od wyniku dla SVM, co czyni RSLVQ dobrym kandydatem dla wszelkich rozwiązań klasyfikacji zdarzeń dźwiękowych w czasie rzeczywistym.

Słowa kluczowe:

RSLVQ, One-vs.-Rest, Multiclass, Sound Events

Table of Contents

List of Figures

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

LVQ	Learning Vector Quantization
RSLVQ	Robust Soft Learning Vector Quantization
SVM	Support Vector Machine
kNN	k-Nearest Neighbour
NPC	Nearest Prototype Classification
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
ROC	Receiver Operating Characteristic
MCC	Matthew Correlation Coefficient
OSR	Overall Success Rate
ICSI	Individual Classification Success Index
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
HAL	Heuristically programmed ALgorithmic computer
RBF	Radial Basis Function
PLVQ	Probabilistic Learning Vector Quantization
SV	Support Vector
DAGSVM	Directed Acyclic Graph Support Vector Machine

1. INTRODUCTION

Hands-free usage or, more general, sound-triggered events are nowadays in enormous demand. With each update of software and each hardware generation capabilities of respective devices or implementations are always bigger. Automation of everyday tasks is here the main motivation, followed by safety, security and, unfortunately, espionage concerns.

1.1. Motivation

On the verge of biggest technology companies bringing Speech Recognition systems into general use [*Microsoft's 'Cortana', Apple Launches iPhone 4S*] there is increasing demand on improving the quality of service provided by them. Nonetheless speech recognition systems are already seen in fighter aircrafts [E*urofighter Typhoon*] or in-car solutions; with the first milestone in 1952 by Audrey (Automatic Digit Recognizer) [*Juang, Rabiner 2005*]. All this partially inspired by such fictional contraptions like “HAL” - an intelligent computer from an acclaimed Stanley Kubrick's movie “2001: A Space Odyssey”. But there is also one a little overlapping field that grabs attention – Sound Event Recognition and Classification.

Simultaneously with emulating the central nervous system of living things by Artificial Neural Networks, researchers have been conducting studies on human ear and its artificial imitator. Mostly, inspirations come from the human cochlea and operate on frequency domain properties [*Wang, Brown 2006*]). All this processing is done in order to better facilitate feature extraction methods.

Robust Soft Learning Vector Quantization (RSLVQ) [*Seo, Obermayer 2003*] is an extension of Learning Vector Quantization (LVQ), which has been proven to generate better results. The original version of LVQ algorithm is already almost 20 years old [*Kohonen 1986*]. LVQ alogithms group is a part of wider class of learning algorithms – Nearest Prototype Classification (NPC). To this family belongs also well-known k-Nearest Neighbour algorithm (kNN) proposed already in 1951 [*Fix, Hodges 1951*]. The basic LVQ version during its prototypes update relies only on two currently closest prototypes for considered training example. This heuristic can introduce not optimal results on given training data set (the order of training examples and their initialization are crucial in terms of the end result).

The main goal of the thesis is to compare performance of classification for Robust Soft Learning Vector Quantization (RSLVQ) with two popular classification algorithms: Support Vector Machine (SVM) and k-Nearest Neighbours (kNN). Not only classification performance should be considered. Another performance factors like memory or time consumption are relevant. Appropriate pre-processing of input learning data is to be expected given the nature of aforementioned algorithms and the data itself.

Since not all considered algorithms operate by default on the same branch of classification (either binary classification or multi-class classification) the comparison should be carried out for each type of classification. The case of extending binary-only classifiers for multiclass data is not particularly an obvious task. Coming across conflicting classifications from binary classifiers is to be expected (one has to introduce some confidence value estimation [*Tax, Duin 2002*]). Therefore in this paper only synthetic mean values of class-performances is calculated; estimation of real value of performance extends to another specific problem and can be presented in another paper.

1.2. Scope

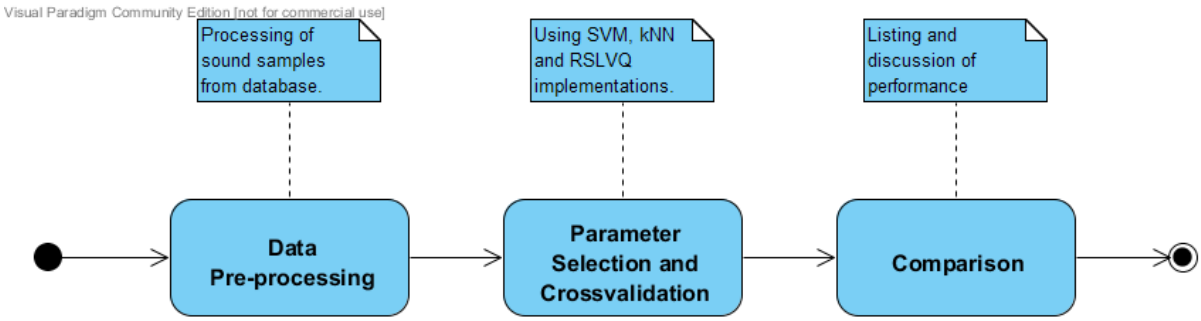

Figure 1.1: Milestone stages of algorithms comparison.

Required sound processing is provided by the Two!EARS library [*May, Decorsière, Kim, Kohlrausch 2014*]. Sound samples used in this work come from the 2013 IEEE Audio and Acoustic Signal Processing Challenge [*IEEE AASP Challenge*]. Data processed in such a way is applied to different algorithms within a benchmark framework focusing on reliability of achieved performances. Acquired various results are compared and discussed in terms of future work [Figure 1.1].

1.3. Outline

The second and the third chapter cover mostly theoretical part. The second chapter will briefly present Machine Learning algorithms and principles considered in this paper (with focus on different performance measures, what is crucial in this work). Followed by fundamentals of required sound processing and feature extraction in the next chapter. Then, related works will be discussed with the concept of benchmark described (the fourth chapter). At the end, benchmark results will summarize and attempt to indicate better performing algorithms and their learning parameters in the case of classification of sound events. The drawn conclusions cannot be easily transferred onto the area of e.g. image recognition since the performance of specific algorithms is to some degree dependant on the given training data.

2. MACHINE LEARNING FUNDAMENTALS

There are three main Machine Learning approaches [*Bishop 1995*]:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning.

This work focuses only on Supervised Learning (colloquially referred to as "learning with a teacher") for classification task. To each available observation vector $\mathbf{x}^{(\alpha)}$, there is a certain label value $y^{(\alpha)}$ attached (for example label stating if received e-mail is spam or not). Given all training pairs:

$\{(\mathbf{x}^{(\alpha)}, y^{(\alpha)}), \alpha = 1, \ldots, p\}$,

where p is the total number of observations, after the learning process it is possible to predict the label value of a previously unseen vector. The predicted label value should more or less correspond with label values of training set.

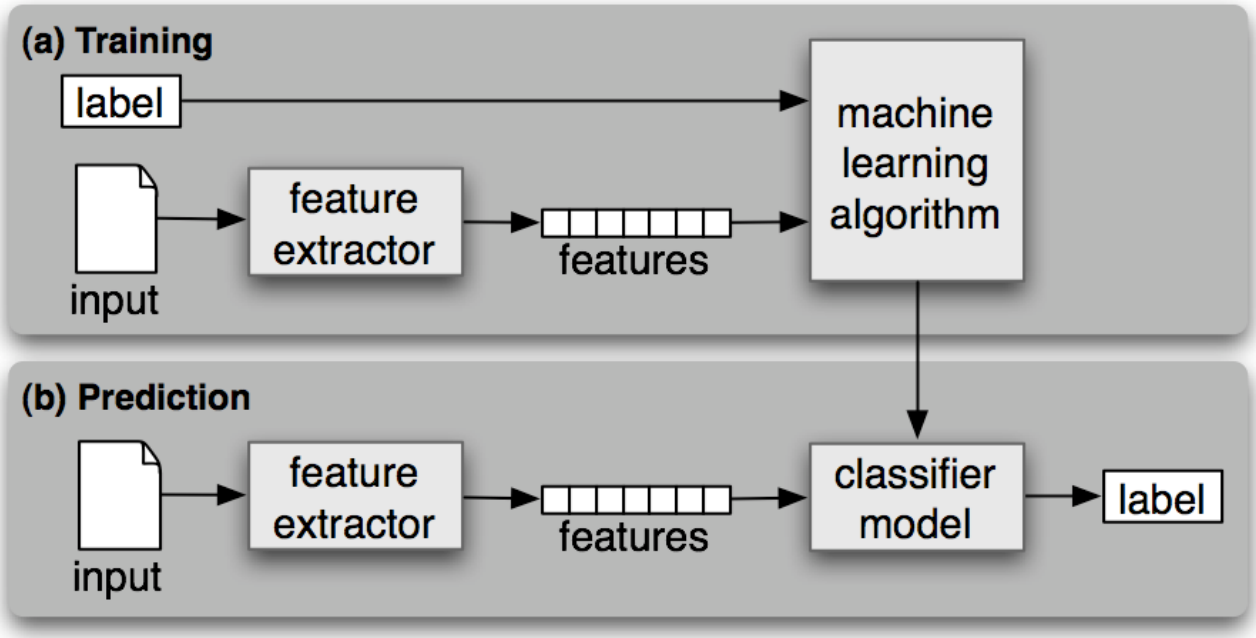

Figure 2.1: Training and Prediction part of typical Machine Learning algorithm usage [*Joe McCarthy*].

The figure 2.1 depicts the general outlook on stages occurring in Machine Learning for classification. During the training part an algorithm uses pairs of training data examples together with labels to find a best fitting model of classifier. Training data is properly processed (through feature extraction described in the Chapter 1.) to adapt it to training data required by the algorithm. The prediction part of an algorithm extracts the feature on the same principle as during the training part. Then the classifier attempts to find the best fitting class representation (label) for the given test example. It maps an instance to a label using internal 'knowledge' (data structures).

2.1. Classification Performance Measurement

In more general case, specifically – the regression, all the predicted values are not grouped into labels (called also classes). Calculating various difference measures between predicted and expected values gives coarse overlook on the quality of the learning part of the used algorithm. As mentioned previously, the data chosen for training part has crucial influence on the error rates (results should be of course calculated on the unseen data to avoid overfitting). Over the years numerous performance measure techniques for classification have been proposed. In many cases they are practically equivalent [*Labatut, Cherifi*]. However, one applying a certain performance measure has to be aware of specific properties of the chosen technique (especially on the margins of the applicability range and untypical usage) and also the classification context as well as the objective of classification. This chapter provides a short overview of popular performance measure techniques with brief justification on measures chosen to implement in the later work.

		Actual class	
		Condition positive	Condition negative
Predicted class	Prediction positive	**TP** True positive	**FP** False positive
	Prediction negative	**FN** False negative	**TN** True negative

Figure 2.2: Confusion Matrix for binary case.

The whole picture of respective (per class) error rates is given by the confusion matrix. Figure 2.2 shows the confusion matrix for binary case. Numbers in all cells add up to the total number of samples (of classified data during test stage). **TP** and **TN** values shows the correct prediction of test data sample, whereas **FP** and **FN** values indicate the false predictions. There are a few popular performance measure techniques used in most cases of classification algorithms implementations. These techniques are presented and briefly discussed in the next sub-chapters.

		Actual class		
		Class A	Class B	Class C
Predicted class	Class A	TP_A	e_{BA}	e_{CA}
	Class B	e_{AB}	TP_B	e_{CB}
	Class C	e_{AC}	e_{BC}	TP_C

Figure 2.3: Confusion Matrix for multiclass classification case.

For multiclass classification case the confusion matrix grows in size depending on the number of classes, as shown in Figure 2.3. Furthermore, the false negative and false positive values from binary classification are replaced with respective error rates per class (for given class prediction). For example the value $\mathbf{e}_{AB}$ indicates how many times the B class were predicted when the actual class of the considered sample was A. Others prediction rates (besides $\mathbf{TP}_X$) are calculated using the corresponding error rates $\mathbf{e}_{XY}$. In the same away as with the binary case, the sum of all the numbers in the confusion matrix cells gives the total number of test samples. To use the formulas from binary classification in multi-classification case, the most general approach is to use means of calculated 'per class' values from confusion matrix. For some cases one would also want to incorporate the variance or the minimum of those values. Several performance measure techniques designed for multiclass classifiers have

been proposed [*Labatut, Cherifi*] (like Individual Classification Success Index (ICSI) that gives the ability to control the share of classes performances in overall performance). But yet, the paper discusses also on the binary classification in the one vs. rest approach. Hence, one performance measure is adapted to the both cases (on account of generalization and simplicity in regards to performance measurement).

Estimated performance of the classifier could be quite inaccurate when encountering an imbalanced dataset. Some training datasets can have significantly more instances for one class. The problem can occur in both binary as well as in multiclass classification. Some performance metrics are biased when working with class imbalances. Since the dataset used in this work is heavily imbalanced, the proper performance measure has to be chosen to avoid too optimistic (and thereby unrealistic) performance estimations. Sometimes, just by the nature of the input data, the imbalanced training dataset is expected.

Most machine learning algorithms for classification will use more or less direct the confusion matrix. The typical derivations from the confusion matrix used to calculate performance measures are:

- $sensitivity = recall = \frac{TP}{TP + FN}$

  Sensitivity, also called true positive rate or recall rate, indicates the probability of a positive prediction outcome, given the actual condition is positive. Bigger sensitivity values are preferred. However, maximizing only the sensitivity will not always produce better classifier. For example, a cancer test designed to predict always positive value would have 100% sensitivity but would be completely unusable. This unwanted tendency to predict more positive values results from not taking false positive values into account. To be precise, in binary case recall is called sensitivity (however, often these performance measures are used interchangeably).

- $specificity = \frac{TN}{FP + TN}$

  Specificity (sometimes called true negative rate) shows share of correctly classified negative instances in all negative predictions. For instance, all healthy people are classified negative for cancer test. Specificity measure tries to counterweight the sensitivity measure to achieve more balanced classifier.

- $precision = \frac{TP}{TP + FP}$

  Precision, known also as positive predictive value, is the ratio of the number of true positives to all the instances classified as positive. In case of Machine Learning the concept of precision may differ from other branches of technology. The maximized value of precision would indicate that all labels predicted as positive are in fact from predicted class.

Coming back to sensitivity and specificity, one should not only rely solely on one of them. Some balance between those two performance measures is needed, depending on situation and desired ratio of those measures. One approach is to achieve the acceptable level of sensitivity and try to maximize the specificity. Depending on the situation, one would prefer higher sensitivity over specificity and vice versa (metal detectors go off more often than it is needed; high sensitivity is desirable to avoid smuggling).

One of the ways to compensate imbalances of classes is to inflate (mostly by copying) the least populated labels in order to achieve uniform distribution of labels. However, this method affects the actual distribution of classes in real-world example. If the classes are naturally imbalanced then it would be apparent to have a little bias into chosing one specific class.

2.1.1. Accuracy

Accuracy is the coarse overall correctness of the model. It is the fraction of all correctly classified instances to all the instances in the test data:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad .$$

With this performance measure one cannot control the share of true values (correctly classified: true positive and true negative values) in the results. Therefore counterweighting sensitivity and specificity is not possible in this case. One can imagine also a situation where classes are highly imbalanced. If spam is scarce in one's mail inbox, the classifier stating all the time 'no spam' classification would have quite high accuracy value.

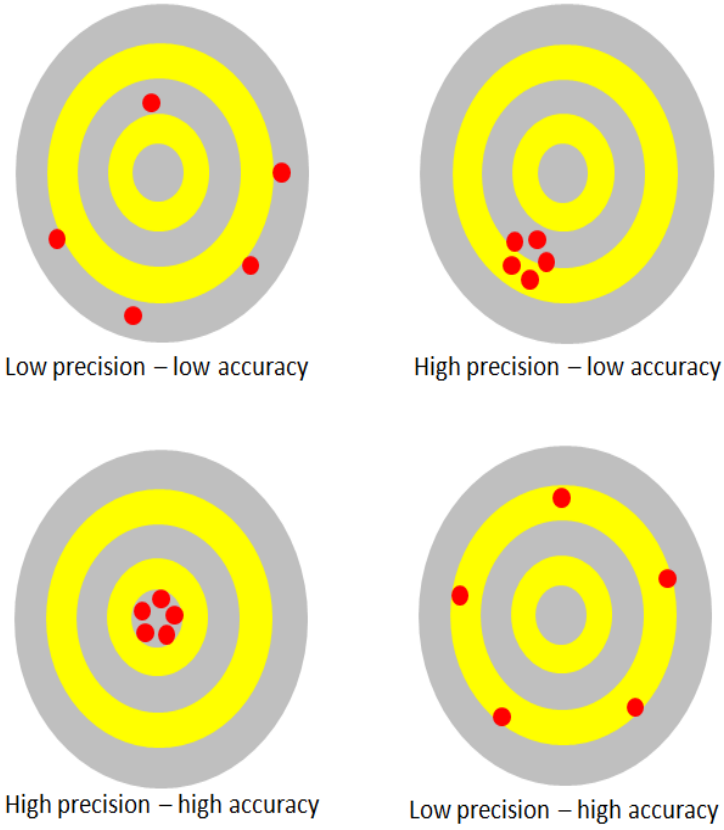

Figure 2.4: Example models of different accuracy and precision [*Keith Gibbs*].

In everyday speech accuracy and precision are used as synonyms. However, the model's prediction can have high accuracy but low precision and vice versa (as shown in the Figure 2.4).

2.1.2. Balanced Accuracy

Balanced Accuracy is a useful performance metric designed to perform well also on imbalanced datasets (where basic accuracy fails and favours the more numerous class). Balanced Accuracy is the arithmetic mean of sensitivity and specificity (what is the same as average accuracy per-class):

$$balanced\,accuracy = \frac{sensitivity + specificity}{2} \quad .$$

This is also equal to the arithmetic mean of recall for each class. One of the drawbacks of Balanced Accuracy is the inability to control the share of sensitivity and specificity values in the final result what can lead to unreliable classifiers in extreme cases.

2.1.2.1. Fair Balanced Accuracy

To avoid one of the measures to have most share in the final balanced accuracy value, Fair Balanced Accuracy is introduced (for Two!Ears project purposes [*Trowitzsch*]):

$$fair\ balanced\ accuracy = 1 - \sqrt{\frac{(1-sensitivity)^2 + (1-specificity)^2}{2}} \quad .$$

Fair Balanced Accuracy tries to evenly distribute its value into sensitivity and specificity measures. Therefore square function provokes the classifier into avoiding low values of any of those metrics. The substraction and square root serve only the purpose of maintaining the same endpoints of the applicability interval as in balanced accuracy which is $[0,1]$.

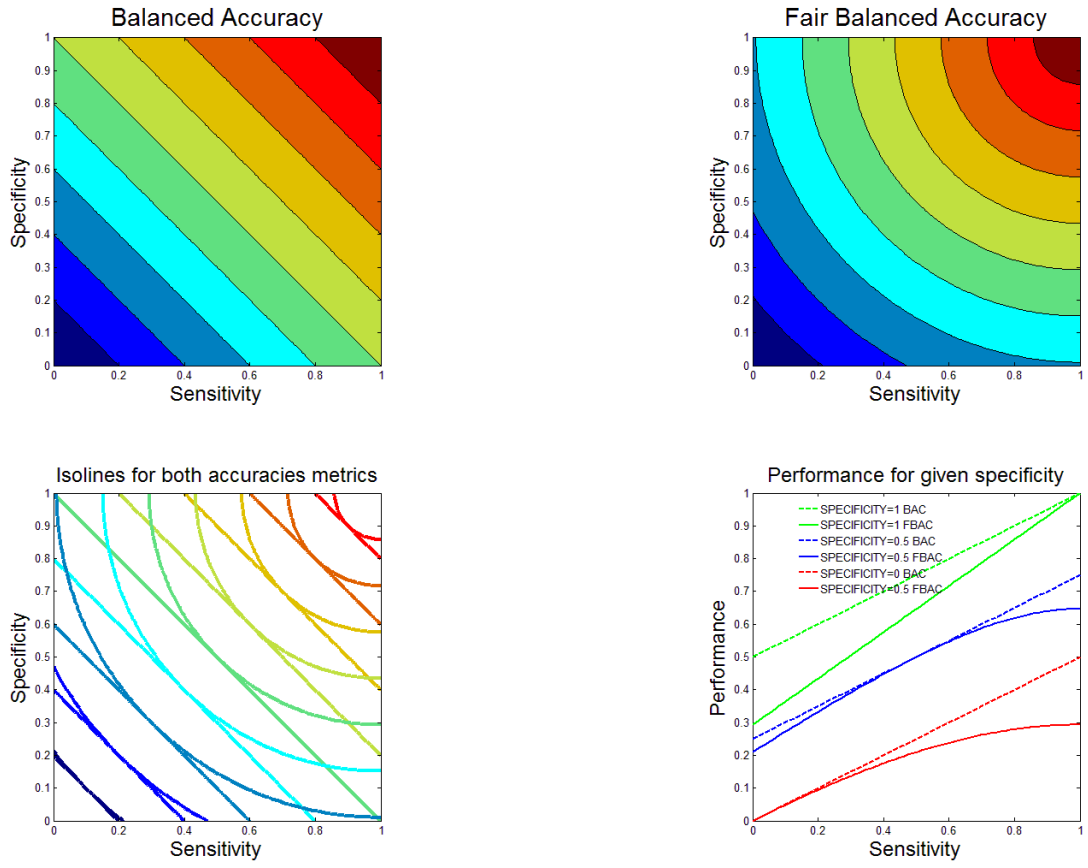

Figure 2.5: Comparison of both Balanced Accuracies metrics.

Since the initial tests of the algorithms for the given data have shown that specificity was strongly preferred in many cases, the usage of basic Balanced Accuracy performance

measure was abandoned in favour of Fair Balance Accuracy. The Figure 2.5 shows the comparison between two discussed Balanced Accuracies metrics. Fair Balanced Accuracy favours more equivalent values of sensitivity and specificity. The Isolines of the same colours show the same value of accuracy for both measures. In terms of boundary cases, these performance metrics can differ quite much. For Balanced Accuracy of 0.5 (for specificity = 1 and sensitivity = 0), Fair Balanced Accuracy returns 0.3. Therefore, a classifier with the hypothetical Balanced Accuracy performance of 0.7 (what would be for example sensitivity of 0.7 and 0 value of specificity) would only have Fair Balanced Performance of about 0.27.

2.1.3. F-measure

F-measure is also known under the name F-score. It uses test's precision and recall values as arguments. The general formula has one positive real parameter β:

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

The β Parameter makes it possible to weight the contribution of precision and recall in the final value of F-score. Commonly used F-measure is F1-score, where the parameter β equals 1. This value of β gives the interpretation of the F1-score as the harmonic mean of precision and recall:

$$F_1 = \frac{1}{\frac{1}{2}\left(\frac{1}{recall} + \frac{1}{precision}\right)} = 2\frac{precision \cdot recall}{precision + recall}$$

In order to achieve high value of F1-score, both recall and precision should be large. However, given the fact that neither precision nor recall take true negative rate into account, it may not be preferable to use this performance measure to calculate the performance of a binary classifier. Moreover, according to the article 'Predictive Coding Performance and the Silly F1 Score', relying on F1-score in all cases is like comparing apple to oranges:

The point is that the appropriate trade-off between low recall with high precision and high recall with lower precision depends on the economics of the case, so it cannot be captured by a statistic with fixed (arbitrary) weight like the F1 score [Bill Dimm].

The F1-score is strictly arbitrary; without previously set requirement on the trade-off between precision and recall one should use another performance measurement or test

considered system thoroughly with different β parameters. The whole approach of testing the learning system in this case could be proposal for another thesis. Therefore, given the fact of parametric nature of F-measure, F1-score is not included into calculations of algorithms performances (to avoid adding another layer of parametrization to the problem). Generally, the most reasonable case of measuring the performance is to compare the precision at the same level of recall [*Bill Dimm*].

2.1.4. Matthews Correlation Coefficient

Matthews Correlation Coefficient is one of the performance measure that tries to include the fact that not all the labels in train data are balanced (the same number of samples for each class during training stage). MCC is the correlation coefficient between predicted and actual labels. Directly from confusion matrix Matthews Correlation Coefficient can be calculated as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad .$$

In case when any of the part sums in denominator equals 0, the Matthews Correlation Coefficient is set to 0 (what indicates the random prediction). The minimal value of MCC is -1, contrary to typical ranges of performance measures – from 0 to 1.

Matthews Correlation Coefficient is noticeably more computationally demanding in comparison of other performance measures like different variations of accuracy. Therefore, this work uses accuracy-related performance measures. Like many other performance measures Matthews Correlation Coefficient relates to the binary case of classification (however there are extensions for multi-class classification [*Gorodkin 2004*]).

2.1.5. Overall Success Rate

The most popular and widely used multiclass classification performance measure method, due to its simplicity, is Overall Success Rate. The interpretation is straight away the mean of all correctly classified instances. Numerically this is the trace of the confusion matrix:

$$OSR = \frac{1}{q}\sum_{i=1}^{q} TP_{c_i} \quad ,$$

where q is the number of classes.

However, calculating performance only on true positive values is prone to favour a class with the bigger number of instances. In extreme cases the best classifier would always return the label of the largest class. Therefore, in this paper only the balanced performance measure are considered (due to the very imbalanced data set).

2.1.6. Receiver Operating Characteristic

The Receiver Operating Characteristic (ROC) curve is a standard technique for summarizing classifier performance over a range of tradeoffs between true positive and false positive error rates. The Area Under the Curve (AUC) is an accepted performance metric for a ROC curve. [*Chawla 2005*]

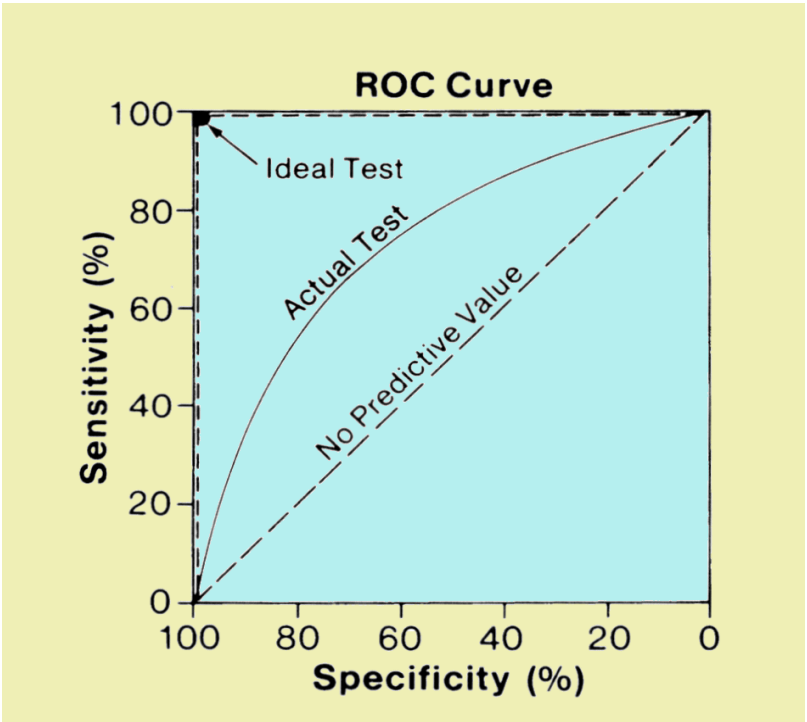

Figure 2.6: Sample plots of hypothetical ROC Curves given the number of true positives (sensitivity) and the number of false negatives (inversed specificity) [*Sprawls*].

Figure 2.6 shows the hypothetical ROC curves. The ideal curve intersects the point (100,100) what gives the maximum performance (for accuracy the maximum value equals 1). In practice the Actual Test curve is more stepped, without smoothness; there is no reason to perform thousands of tests to compare a few classifiers. For acquiring somehow general performance of a given classifier and ability to compare classifiers the Area Under the Curve from the ROC Curve is calculated. Depending on the sensitivity and specificity preference, another factors could be taken into account. For example the area until the first intersection of classifiers or in the specific range.

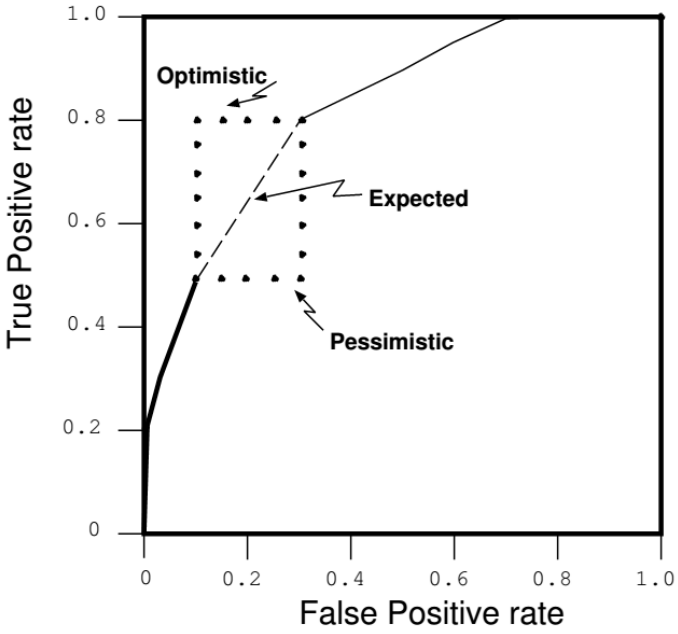

Figure 2.7: Under- and over-estimation of the ROC curvature [*Fawett 2004*].

Trying to assess ROC Curves for multiclass classification could be cumbersome. However, there are some attempts in generalizing the AUC metric for multiclass classification [*Hand, Till 2001*]. For native multiclass classifiers the problem of under-sampling would occur [*Chawla 2005*]. Namely, the ability to plot sensitivity value for the given specificity on the given class ROC Curve is limited (it is not so easy to acquire desired sensitivity or specificity). Hence, the actual area under a line connecting two points is questionable (comparison between possible ranges of ROC Curves between given tested points in Figure 2.7). Usually either the optimistic or pessimistic curve is chosen to have at least the knoweledge whether the AUC value is under- or over-estimated.

2.2. Cross-Validation and Error Estimation

Cross-Validation is important method to choose the most accurate (and most of all the most reliable) classifier from a given set of classifiers. Reliability is fulfilled mostly after proving that the classifier does not induce overfitting (by ensuring that testing is executed on the previously unseen data). It has been proven that the best type of Cross-Validation for model selection is 10-fold Stratified Cross-Validation (for a real-world datasets) [*Kohavi 1995*]. Interestingly, leave-one-out Cross-Validation is not the best choice since it has high variance (being, however, almost unbiased) [*Elfron 1983*]. Because of not sufficient computational power, in this work 5-fold Stratified Cross-Validation is implemented. 5-fold Cross-Validation does not produce significantly worse results as 10-fold Cross-Validation (or even the same, assuming the algorithm is stable for a given dataset) [*Kohavi 1995*].

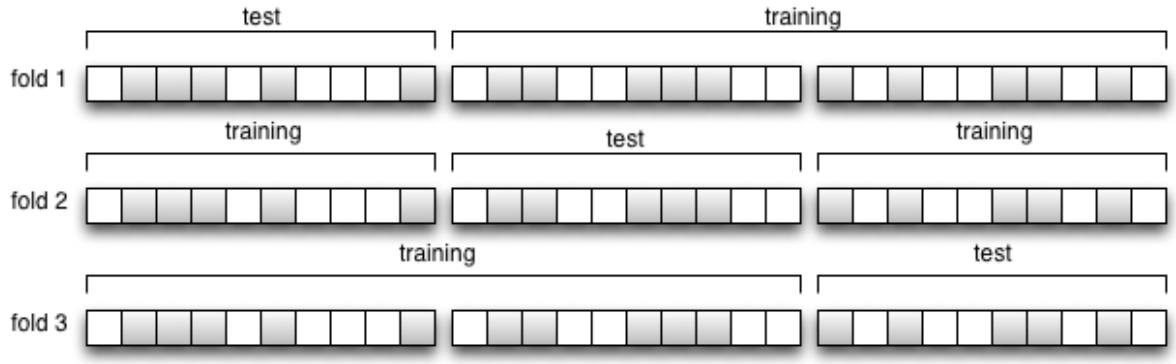

Figure 2.8: Visualisation of 3-fold Cross-Validation. Instances of different colours indicate different classes (labels) [*Martin*].

Cross-Validation is carried out by splitting dataset into smaller folds (as shown in Figure 2.8). The training part is done on all-but-one folds for every possible fold being taken out from training. The trained models are afterwards tested on the unseen data left for testing. Performing error measures on each fold one can compute averaged generalized performance for the given circumstances (for k-fold Cross-Validation):

$$E^G = \frac{1}{k}\sum_{i=1}^{k} e_i^G \quad .$$

Some approaches take also the variances into account, however, it has been widely accepted to perform averaging of per-fold performance measure to be a good strategy (given large enough datasets and folds count) [*Kohavi 1995*].

2.2.1. Stratified Cross-Validation

The basic technique to split the dataset in folds is to random assign training data into folds. Although it seems reasonable, random deviations of such an assignment can produce unreliable performance results of classifiers. Especially for a relatively small set of data (or the smallest class being scarcely represented) the stratification of data is recommended. Stratified folds are such folds that have approximately the same distribution of labels as the whole original dataset. The proportions between classes are preserved. For classification the Stratified Cross-Validation is in most cases a good strategy. *Stratification reduces the variance slightly, and thus seems to be uniformly better than cross-validation, both for bias and variance [Kohavi 1995]*.

2.2.2. Nested Cross-Validation

After model selection one would like to perform model assessment to ensure themselves that the chosen strategy of model selection is not biased [*Krstajic, Buturovic, Leahy, Thomas 2014*]. Similarly to Cross-Validation for model selection, the outer loop of validations is provided. All cross-validated model selections are then cross-validated to assess more reliable and averaged model performance for a given classifier. At the end, the classifier is trained on the whole dataset for future predictions.

2.3. Adapting two-class classifiers for multiclass classifications

Many, if not the most, of usages of the machine learning is a two-class classification (binary case). Moreover, many classification algorithms work only in binary classification (like Support Vector Machine used in this work). Using multiclass classifiers for two-class case is straight away. However, adapting two-class classifier for multiclass classification requires some effort. Namely, a few binary classifiers have to be somehow combined to achieve multiclass classification. There are two main approaches to achieve that: One vs. Rest and One vs. One. The first approach (nomen omen) is to train the classifier for each class where 'opponent'-class is all the left classes merged into one rest-class. The second approach

is to train the classifier for each combination of pair from given classes.

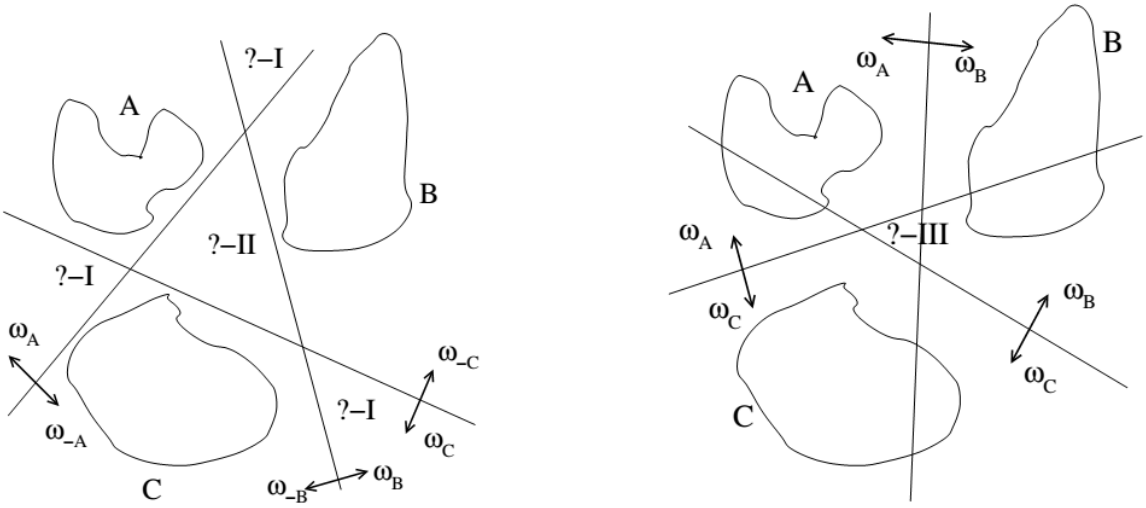

Figure 2.9: One vs. Rest (left) and One vs. One (right) multiclass classification approaches. Areas marked with question marks show unsettled classification result area. [*Tax, Duin 2002*].

In both cases one problem emerges. For the situation shown in Figure 2.9 for both classification approaches unresolved classification areas appear. Classifiers dividing the plane into two entail the existence of the area for which all the classifiers return the 'rest-class' value. There are several techniques of attempting to classify those unsettled instances (like probability estimation or random assignment) [*Tax, Duin 2002*]. This problem itself is complex; therefore in this work the final classification is not executed. Per-class performances are calculated and only obtained mean and variance values of performance metrics are considered. One has to be aware that this approach is optimistic but more important is to focus on the mechanics of used algorithms and ability to compare individual performances.

The adaptation technique for multiclass classification in this work was One vs. Rest. In comparison to One vs. One this approach gives similar results in terms of performance, where One vs. One has significantly bigger complexity $O(numberOfClasses^2)$ [*Chih-Wei Hsu, Chih-Jen Lin 2002*]. The One vs. Rest classification was chosen for all the algorithms in theirs binary adaptations.

2.4. Learning Rate

The learning rate parameter controls the speed of convergence by adjusting consecutively the prototypes update (or the weights values in case of neural network). Too small learning rate will slow down the learning process. In some cases this will make an algorithm practically unusable (due to the long time of the training part). On the other hand too large learning rate will cause the supposed convergence point to be further away from the actual point of the maximum performance (because of the large prototypes updates involving big step size on every iteration). This behaviour is called 'Apparent Convergence'. Prototypes updates in each step of iteration are given in basic case by

$$\theta(t+1)=\theta(t)+\alpha\cdot(x-\theta(t)) \quad ,$$

where α is the learning rate and x is a training data point (or generally a vector) from the training dataset.

In some cases not small enough value of fixed learning rate can lead model to oscillate between two states and going slowly towards the local minimum [*Bishop 1995*]. The adaptive learning rate mostly overcomes the issue with selection of learning rate value. Starting with relatively large learning rate involving big prototype values update steps, subsequent learning rate values are multiplied by annealing rate parameter. Updating the learning rate value after each step of prototypes update decreases the learning rate to allow a better model fit:

$$\alpha(t+1)=\rho\cdot\alpha(t) \quad .$$

The ρ value can also be parameterized depending on error rates obtained after prototypes update in each iteration step. This method provides some awareness of the 'distance' from the convergence point and allows to adjust step size respectively.

2.5. Adopted Algorithms

Two quite common algorithms were chosen to compare with the Robust Soft Learning Vector Quantization algorithm: K-Nearest Neighbours as the simplest one and Support Vector Machine as more sophisticated approach in Machine Learning for classification. One cannot clearly state that one of those algorithms is always better; the decision depends on the requirement and specific aspects of data. In the following subchapters these three algorithms are briefly presented and discussed.

2.5.1. k-Nearest Neighbours

K-Nearest Neighbours is one of the most basic algorithms from Nearest Prototype Classifiers algorithms family. Stupendously fast to train since all the training samples become prototypes as they are. Also relatively fast in testing part when consciously coded.

During training part kNN algorithm saves the whole training vector (or matrix) as the set of prototypes. For substantially big dataset this is the least optimal solution (given also the fact, that for Machine Learning purposes data used in training stages should be as big as possible, given no redundancy). There is a linear relationship between size of the training dataset and memory requirements for the acquired model.

The test part for kNN considers all the data instances gathered in training part. Distances between a sample from test data and all the train samples are calculated and arranged in the ascending order. The first k occurrences of samples from the same class indicate the winner-class. The distance between vectors is commonly based on the Euclidean distance. However, there are other measures like Chebyshev or Manhattan Distance. Also in more advanced implementations the distance itself can be weighted (for example inversed squared distance) [*Mathworks*]. To lower computational complexity the distance calculations can be adapted. For example, when calculating the Euclidean distances, square roots are neglected (since one has to only decide the relation of two considered distances, not to output their values).

The distance and weight measure choice is totally heuristic; one can only draw conclusions from values or prior knowledge about the data. When the number of dimensions is large, the kNNs possibility to properly classify test data lowers. For many dimensions multiple points are equidistant to the given vector (in terms of Euclidean distance all dimensions are equally important).

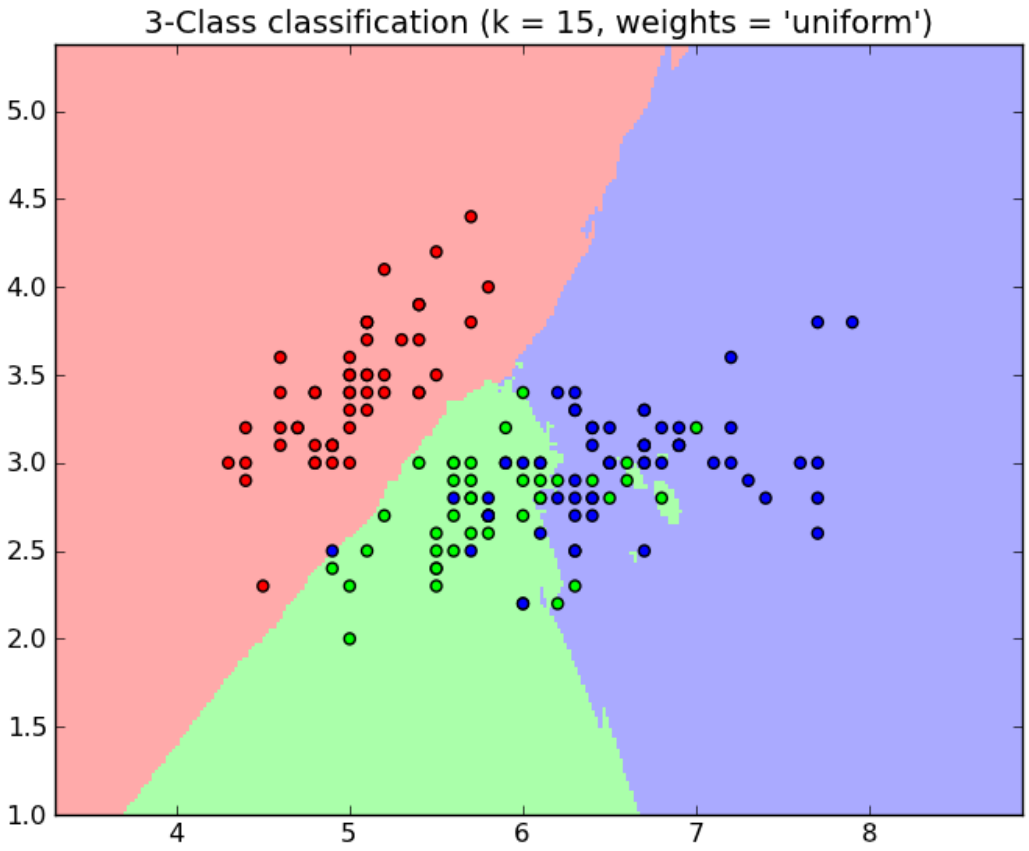

Figure 2.10: Sample classification boundaries for kNN evaluation [*scikit-learn*].

Figure 2.10 shows the one of the issues of kNN classifier ("stepped", not smooth classification boundaries). In the most basic form, kNN does not actually attempt to compile the most fitting model. The training data is the model itself and even small deviation from the actual distribution is not practically suppressed. The bigger value of the k parameter makes the decision boundary smoother. However, for the imbalanced classes the approach of increasing k value can even eliminate the class from the possible output values in extreme cases. Anyway, k-Nearest Neighbour Classifier still ranks in *Top 10 Algorithms In Data Mining [Wu 2007]*.

2.5.2. Support Vector Machine

Support Vector Machine (or Support Vector Networks [*Cortes, Vapnik 1995*]) is more refined and sophisticated in mathematical background as k-Nearest Neighbour. Fundamentally performing linear classification (with extensions to perform regression or non-linear classification using so-called Kernel Trick [*Guyon, Boser, Vapnik 1993*]).

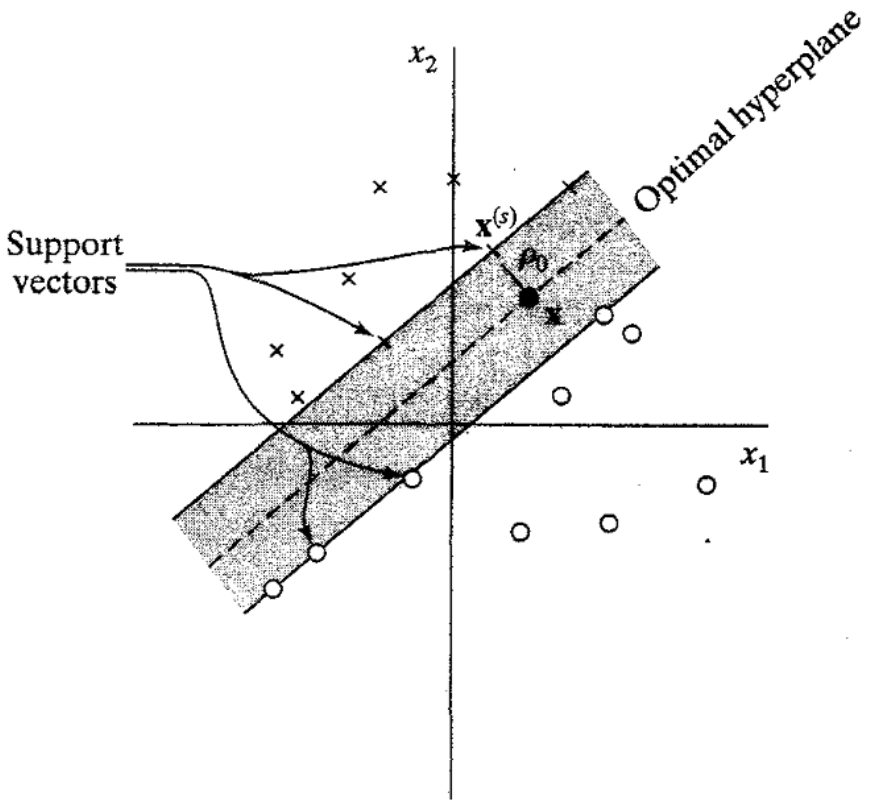

Figure 2.11: An example of the optimal hyperplane separating two sets [*Haykin 1999*].

Briefly, the objective of SVM is to seperate two classes on the feature space with the widest gap possible (as shown in Figure 2.11). A few vectors from the training dataset are selected as "Support Vectors" (enough to define the Optimal Hyperplane given the chosen parameters). Bearing in mind that the training data always have some noise, some vectors can find themselves inside the Hyperplane. This happens for the Soft-margin variant of SVM when computing the Optimal Hyperplane [*Cortes, Vapnik 1995*].

To achieve a wider margin separating classes one has to minimize the function of the norm of the weights vector:

$$f_0: \ \frac{1}{2}\|\underline{w}\|^2 \ .$$

And holding on to the classification rule the following set of equations has to apply:

$$f_\alpha: \ y^{(\alpha)}(\underline{w}\cdot\underline{x}^{(\alpha)}-b)\geq 1, \alpha=1,\ldots,p \ ,$$

where p is the number of training instances. This pair of equations is called Primal Optimization Problem. Applying the Lagrangian gives:

$$L=\frac{1}{2}\|\underline{w}\|^2-\sum_{\alpha=1}^{p}\lambda_\alpha[\,y^{(\alpha)}(\underline{w}\cdot\underline{x}^{(\alpha)}-b)-1] \ ,$$

where λ_α are called dual variables. Solving the Langrangian (implying Kuhn-Tucker conditions) in search for a saddle point results in the classification rule [*Haykin 1999*]:

$$y=sign[\sum_{SV}\lambda_\alpha\cdot y^{(\alpha)}(\underline{x}^{(\alpha)})^T\underline{x}+b] \ .$$

Only a few training vector are qualified as Support Vectors (SV) - fulfil the constraints. For the rest of the data (almost all) $\lambda_\alpha=0$. Therefore, the dual solution is scarce (easier to solve and represent). One of the most important improvement of the SVM is the Soft-margin option [*Cortes, Vapnik 1995*]. Soft-margin allows some vector to be inside the hyperplane margin. Then, the constraints are changed with regards to the regularization parameter C for misclassified training data points. The primal problem is then represented by following equations:

$$f_0: \ \frac{1}{2}\|\underline{w}\|^2+C\sum_{i=1}^{p}\xi_i \ ,$$

where $\xi_i\geq 0$ represents Slack Variable of misclassification rate,

$$f_\alpha: \ y^{(\alpha)}(\underline{w}\cdot\underline{x}^{(\alpha)}-b)\geq 1-\xi_i, \alpha=1,\ldots,p \ .$$

Exploiting the fact that vectors multiplication returns a scalar one can customize the obtained result to fit to the the non-linear function called Kernel (like gaussian or sigmoid). This operation (Kernel Trick [*Guyon, Boser, Vapnik 1993*]) allows to fit complex curves instead of a straight line margin. Next to linear kernel the most popular is the Radial Basis Function Kernel (RBF, Gaussian function):

$$K(x_i, x_j) = e^{-\gamma \|\underline{x}_i - \underline{x}_j\|^2} \quad .$$

In this case hypersphere is created to separate the classes. The proper choice of values of parameters C and γ is critical for the final performance of SVM algorithm for given data.

2.5.3. Robust Soft Learning Vector Quantization

The main drawback (or an advantage in some cases) of k-Nearest Neighbour algorithm is virtually no "learning" part, only gathering the information. The Learning Vector Quantization approach tries to derive appropriate prototypes that generalize the provided data. Since the introduction of the LVQ algorithm many improvements have been proposed. Window rule was introduced to counteract divergence of prototypes (updated driving them consecutively further apart) [*Kohonen 1997*]. A Data point that is close enough to the decision boundary between prototypes will update their coordinates. Any point outside this window does not affect them.

RSLVQ chooses in contrast a "soft" approach. The assumption is that dataset is sampled from a mixture of normal distributions (rendered by each prototype). When the prediction part is straight away – the label of the closest prototype is returned, the focus is here on the training part, namely, the rules for prototypes updates are more refined ("soft" so to say). Briefly, the prototypes update rule goes as follows [*Seo, Obermayer 2003*]:

$$\theta_l(t+1) = \theta_l(t) + \alpha(t) \begin{cases} (P_y(l|x) - P(l|x))(x - \theta_l), & c_l = y \\ -P(l|x)(x - \theta_l), & c_l \neq y \end{cases} ,$$

where y is the label of the data point x .

Abovementioned conditional probabilities result from the mixture of prototypes normal distribution and the distance from the given point:

$$P_y(l|x) = \frac{\exp\left(\frac{-(x-\theta_l)}{2\sigma^2}\right)}{\sum_{j:c_j=y} \exp\left(\frac{-(x-\theta_j)}{2\sigma^2}\right)},$$

$$P(l|x) = \frac{\exp\left(\frac{-(x-\theta_l)}{2\sigma^2}\right)}{\sum_{j=1}^{M} \exp\left(\frac{-(x-\theta_j)}{2\sigma^2}\right)}$$

where M is the number of prototypes.

Most important parameters are the number of prototypes and α (defining the speed and the precision of updates by adapting updates step size). The number of prototypes tends to adjust the complexity of the model (classification boundary). However, for quite obvious datasets distribution many prototypes could end up with almost the same coordinates, what should be expected. The value of "softness" parameter σ is in most cases chosen at the beginning of the training process and kept constant. However, there could be cases, where values of σ per-prototype would lead to better performance results [*Schneider, Biehl, Hammer 2009*]. The fact of prediction based only on comparison between 1-prototype-long distances (not the first k distances like in kNN) gives a hope for short times of the prediction part.

3. SOUND PROCESSING FUNDAMENTALS

The phrase "garbage in, garbage out" applies notably for results given by providing unprocessed data into Machine Learning algorithms [*Quinion*]. Not only do these algorithms have no knowledge about the nature of provided data, but also sometimes they just simply do not operate well in applied value ranges (naive learning rates converging too fast). Redundant and irrelevant data does have influence on training process. As with Neural Networks, the same biologically-motivated approach is chosen by researchers in regards to processing sound data. This perceptual approach attempts to resemble human abilities (provided by ear canals or the cochlea). Audio processing part was provided by Two!EARS working package [*May, Decorsière, Kim, Kohlrausch 2014*].

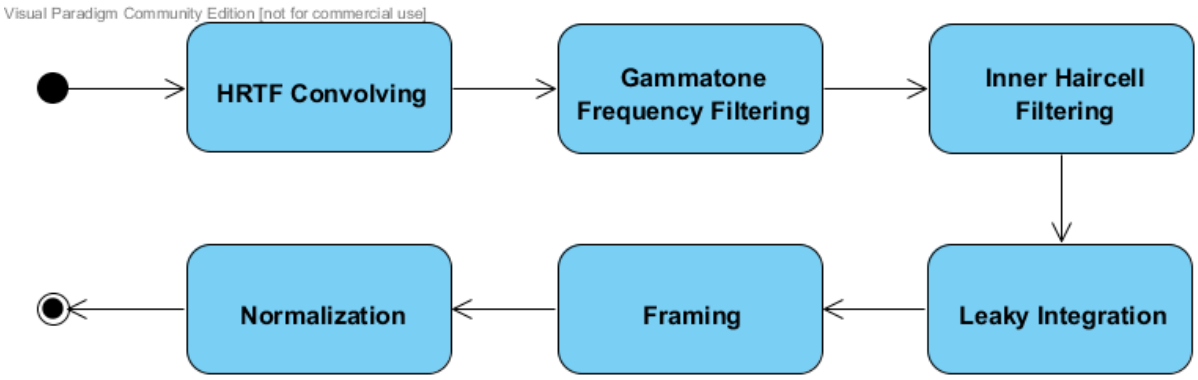

Figure 3.1: Provided sound pre-processing [*Trowitzsch*].

Figure 3.1 presents important steps in sound-data pre-processing utilized in this work:

- Simulating the natural phenomena of reflections and diffractions induced by the human body (especially the head) – applying Head Related Transfer Function [*Ciba 2011*]. The signal coming to each ear is almost always different (depends on the position of the source of sound). It enables by the way ability to localize the sound source and overcome the "cocktail party problem" [*Wang, Brown 2006*].
- Basically set of filters mimicking nature of the impulse response of the mammalian cochlea (for given amplitude a and frequency f):

  $$g(t,n)=at^{(n-1)}e^{(-2\pi bt)}\cos(2\pi f t+\varphi) \quad ,$$

  where n is filter's order and b is filter's bandwidth.

- Imitating the nature non-linear filtering by Cochlear Hair Cells [*Dallos 1985*].
- A kind of deliberately imperfect integration of values. Time degradation factor is involved (older values have smaller impact). An imitation of the neuron's nature.
- Dividing data into blocks. Acquiring proper shift in time (start point of the first block) requires tests to be performed and luck.
- Normalization for a better fit into Machine Learning algorithms.

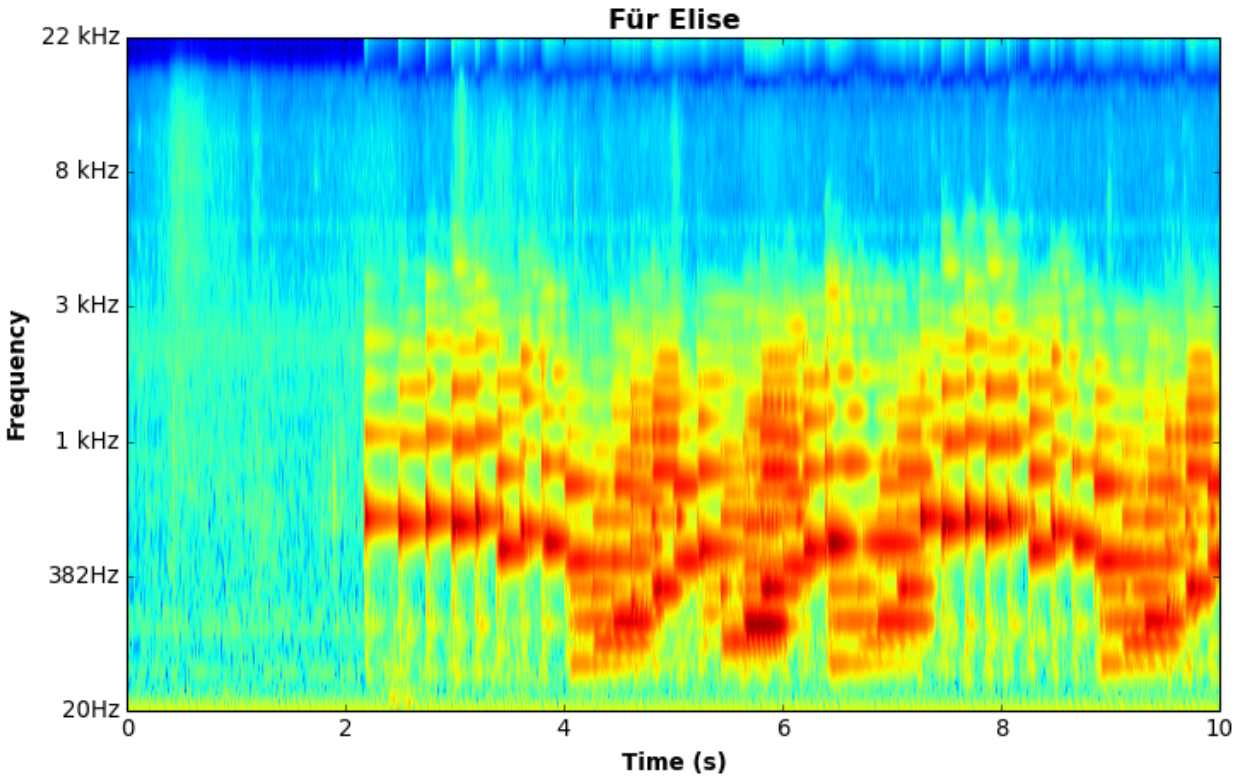

Figure 3.2: Gammatone spectrogram of *Für Elise* [*Heeris*].

With a little knowledge on classical music one can even recognize the processed audio sample of *Für Elise* by its characteristic tones order (Figure 3.2). For example, the first notes played by left hand start in the fourth second (notes between 2 and 4 second being the best-known). Sound samples used in this work also feature a short silence period at the beginning and at the end of file. Those regions are assigned to the *rest-class* since in the real-world usage there would be no data about start and end time of an event. Features are extracted mainly by searching for signal pitches and their correlation in time and/or frequencies.

4. CONCEPT AND IMPLEMENTATION

Initially the goal was to create a function performing selection of the best from the given parameters range for available data. For model comparison sake it was however necessary to record all the estimated models performances for given set of parameters and classification case. This chapter gives an overlook on the submodules or functions implemented to achieve the goal set at the beginning, as well as related to this topic work.

4.1. Related Work

Robust Soft Learning Vector Quantization algorithm is not a significantly old approach to Machine Learning (publication in 2003). It was already proved as better choice than original updated LVQ 2.1 algorithm (for a hand-written letter recognition case [*Seo, Obermayer 2003*]). A Probabilistic LVQ (PLVQ) generalization improvement was proposed [*Schneider, Geweniger, Schleif, Biehl, Villmann 2011*] promising even better results at least for artificial dataset (subjected for future work). An improvement of the window parameter can influence the learning curves, however, without influence on the final performance [*Witoelar, Ghosh, Vries, Hammer, Biehl 2010*].

Support Vector Machine is about twice as old as RSLVQ. It is widely used for classification as well as for regression. For extending SVM to multiclass classification popular approaches are One-vs.-Rest and One-vs.One (with its improvement DAGSVM). [*Hsu, Lin 2002*]. Directed Acyclic Graph SVM (DAGSVM) consciously constructs (with regards to the location of classes on the feature space) a classification decisions graph to narrow down the number of classifications [*Platt, Cristianini, Shawe-Taylor 2000*]. The implementation of SVM chosen for this work is well established LIBSVM package [*Chang, Lin 2011*]. In the first paper SVM already turned out to be fairly good in hand-written digits recognition [*Cortes, Vapnik 1995*].

Over the years also the basic k-Nearest Neighbour algorithm underwent many improvements. From adaptation to regression [*Altman 1992*] to alternative voting-scheme in order to overcome the class imbalances [*Coomans, Massart 1982*]. The version used in this work is the default kNN implementation included with Matlab [*Mathworks*].

4.2. Concept

Figure 4.1: Main activities flow for models' performance assessment.

Figure 4.1 depicts the basic flow of activities for gathering models performances:

- Configuration part creates and manages the parameters object within program operation. Quite often to achieve recursion or retrieve models from memory the binary boolean value is switched in the copy of the configuration object.
- Data Extraction takes care of feature extraction from audio files and comprise training matrix together with labels vector (or matrix in binary case) and label identities (to indicate the source of the training instance for the stratification purpose). Data is also normalized in terms of the mean and the variance.
- Models Generation performs, mentioned in Fundamentals chapter, Nested k-Fold Stratified Cross-Validated performance estimation for a singled-out set of parameter. The only one parameter set will always be selected as the best one what allows to gather information about every single model.

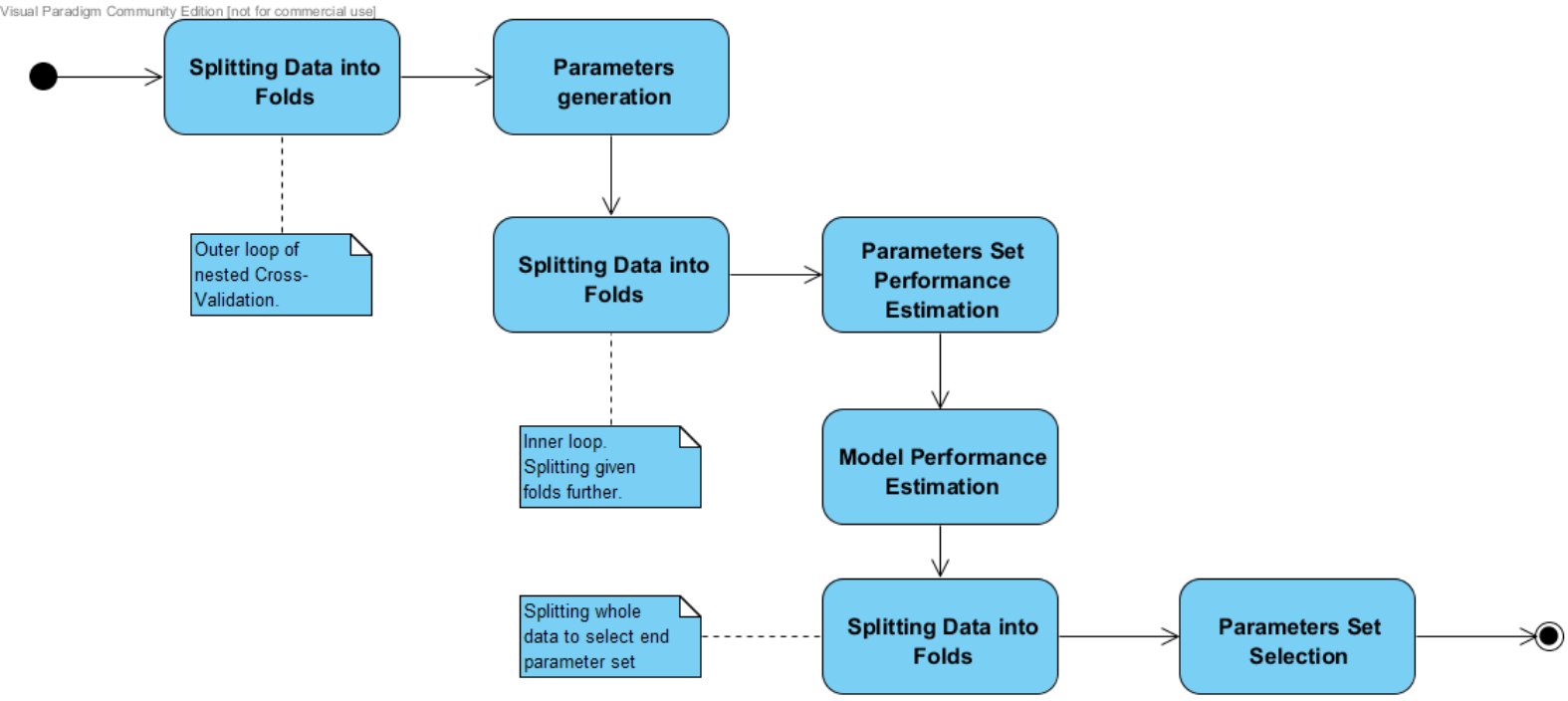

Figure 4.2: Nested k-Fold Cross-Validation to assess the model performance and choose parameters set from a given set.

As mentioned earlier, the module performing Nested k-Fold Stratified Cross-Validation for a given list of parameter sets (Figure 4.2) was used as a validator for each single parameter set. After estimating the generalization performance of the given algorithm by Nested Cross-Validation, the algorithm is trained on the whole unfolded data to choose the best set of parameters for given dataset for future classification (in case of models performances generation it is always the only one set).

```
>> multiClassModel

genPerformance: 0.5759
 parameters: [7x1 double]
   genStats: [1x1 struct]
```

```
>> multiClassModel.genStats

  sensitivity: [16x1 double]
  specificity: [16x1 double]
```

```
>> multiClassModel.parameters

    1.0000
    0.0000
    0.5000
    0.9000
    0.9000
    0.0010
    0.5759
```

```
>> multiClassModel.genStats.sensitivity

    0.3236
    0.1857
    0.5756
    0.3545
    0.3923
    0.2514
    0.0407
    0.8550
    0.1250
    0.8740
    0.3277
    0.7583
    0.4581
    0.5298
    0.4500
    0.1136
```

Figure 4.3: Model structure for Multiclass classification.

Figure 4.3 shows the model structure acquired for multiclass classification case. Only all the parameters and performance measures are taken into account. The trained model itself is never used for classifying further data. Therefore it is never being saved. A model in this work consists of 3 fields: generalization performance, list of parameters and additional performance measures (sensitivity and specificity). In multiclass classification the values of those performances are calculated per-class. Therefore, there are 16 values (*rest-class* class is also enclosed in performance calculation).

```
>> binaryModels

15x1 struct array with fields:

    genPerformance
    parameters
    genStats
```

```
>> binaryModels(1)

    genPerformance: [15x1 double]
        parameters: {1x15 cell}
          genStats: {1x15 cell}
```

```
>> binaryModels(1).genPerformance

    0.6612
    0.7775
    0.8295
    0.7704
    0.8319
    0.6806
    0.6235
    0.8801
    0.7208
    0.7690
    0.6638
    0.7978
    0.6981
    0.8306
    0.7850
```

```
>> binaryModels(1).parameters{1}

    1.0000
    0.0000
    0.1000
    0.9000
    0.9000
    0.1000
    0.6612
```

```
>> binaryModels(1).genStats{1}

    sensitivity: [2x1 double]
    specificity: [2x1 double]
```

```
>> binaryModels(1).genStats{1}.sensitivity

    0.5941
    0.7814
```

Figure 4.4: Model structures for binary classification.

In two-class classification case the list of models is being saved (one for each class, shown in Figure 4.4). In general usage the Nested Cross-Validation module selects the best model for each class. The generalization performances are calculated for each class. Since it is the binary classification, there are only two values of sensitivities (and specificities) for each class and they correspond each other (sensitivity for the first class is equal to specificity for the second class in the considered model).

One of the techniques used to achieve more reliable models (even at the expense of lower performance) is stratification. Therefore an additional vector together with labels vector had to be introduced. Stratified Cross-Validation takes not only classes distributions into account. When splitting data into folds, individual data instances obtained from one audio sample are assigned only to one fold (to prove that the audio event features were properly extracted, not just the ones coming from specific audio sample or recording studio).

4.3. Environment

The following software applications were used for the implementation:

- Microsoft Windows 8.1 Pro
- MathWorks Matlab R2014a 64bit.

All tests were run on the same hardware specification:

- Processor AMD FX-4100 3,6GHz
- RAM memory 1666Mhz 4GB
- SSD Hard Disk Drive.

Typical system performance values were monitored. RAM memory was never fully allocated (mostly under 60% load). Regardless, SDD Drive would unlikely be a performance bottleneck. Monitoring CPU usage revealed that only one core was used by Matlab despite 4 cores being in the processor unlocked.

4.4. Important Implementation Aspects

During the implementation and the testing part some compromises and decisions had to be made. A limited computational power and scarce input data (for some classes) had to be kept in mind. Following list describes most of the issues:

- Nested k-Fold Stratified Cross-Validation caused the training and testing stages of algorithms to be executed multiple times (nested cross-validation for model generalization and the second cross-validation for final parameter set selection). Only the $k=5$ for k-fold Cross-Validation was chosen because of that. Anyway, the generation of all models for all three algorithms took several days.

- $$fair\ balanced\ accuracy = 1 - \sqrt{\frac{\left(1-\frac{1}{q}\sum_{i=1}^{q} sensitivity_{c_i}\right)^2 + \left(1-\frac{1}{q}\sum_{i=1}^{q} specificity_{c_i}\right)^2}{2}}$$

  The values of the Fair Balanced Accuracy were calculated by averaging the values of sensitivities and specificities per-class. For each class confusion matrix was determined and respective values of TP, TN, FP, FN were used to calculate sensitivity and specificity for that class.

- Each calculated model (described earlier in the current chapter) was saved into the file for later examination and comparison purposes. To achieve unique names, the parameters vector values were applied to hash function. Parameters vector also holds the generalization performance (only for easier review of those parameters later).

- LIBSVM implementation provides weights parameters assignment to counterweight the class imbalance problem. Before each training part, the weight for each class (also in two-class classification) were calculated.

- For two-class classification LIBSVM was separately trained for each binary labels vector (to achieve per-class performance measures and make the comparison between algorithms more reliable).

- Final multiclass classification in One-vs.-Rest approach is not executed. Encountered classification contradictions and ambiguities would have to be resolved [*Tax, Duin 2002*]. This operation would possibly introduce some bias towards any of algorithms.

5. EVALUATION

In this chapter all results are presented. Different aspects of algorithms' mechanics are looked upon as well as carried out performances calculation. In the first subchapter the available sample data is discussed. The following section cover the performances evaluation and observations regarding other metrics. The last subchapter compares the results of adopted algorithms.

As described in Implementation chapter, the Nested k-fold Stratified Cross-Validation is performed for all algorithms and classification variants to calculate generalization performance. The folds count is 5 (because of computational power restrictions).

5.1. Input Data

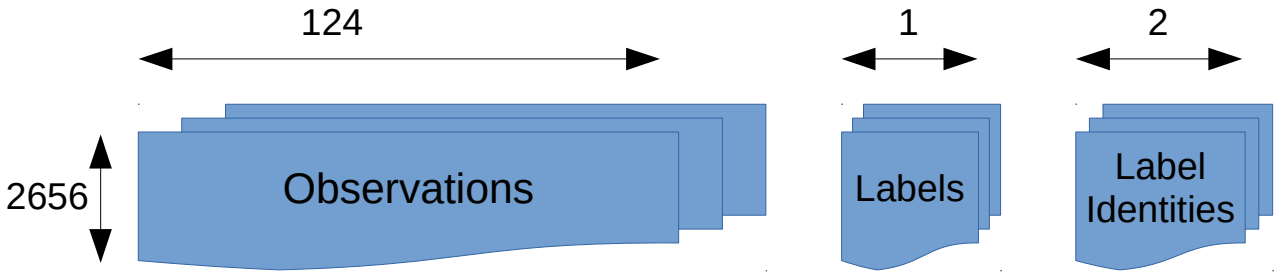

Figure 5.1: Format of Input Data for algorithms after feature extraction.

The final dataset used as input data for algorithms (in multiclass classification case) goes as follows (shown in Figure 5.1):

- the Values matrix of the length of 2656 and 124 dimensions,
- Labels vector of the length of 2656,
- Identities Labels used for stratification of the size of 2656x2.

In binary classification case the Labels vector is expanded into matrix for all possible One-vs.-Rest binary vectors of labels. It does not provide any additional information. Labels Identities matrix holds information of the origin of the given observation instance – class as well as source file (for stratification purpose). Data is normalized only altogether, not for each validation chunk.

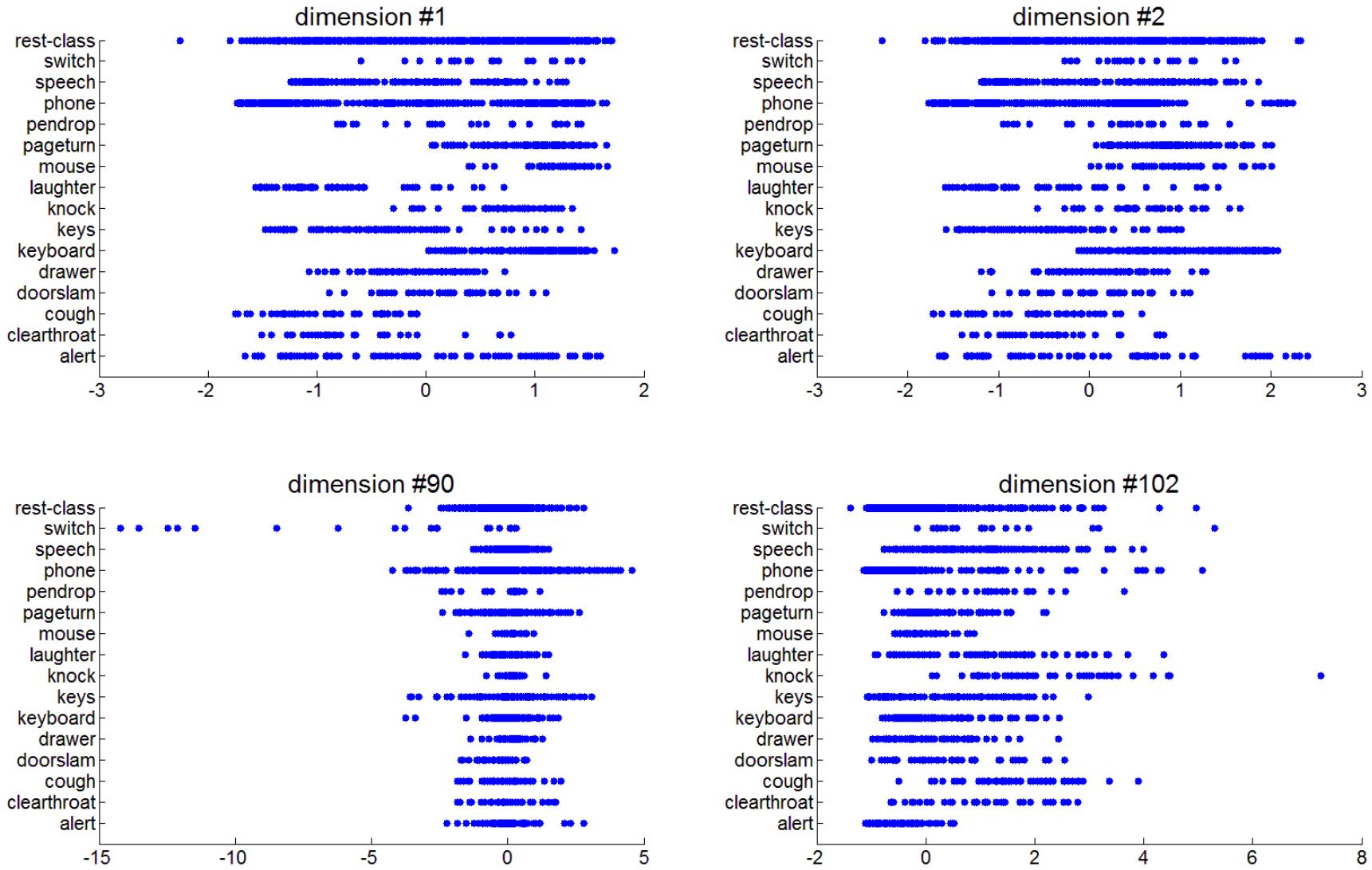

Figure 5.2: Input data values distribution of four dimensions after normalization.

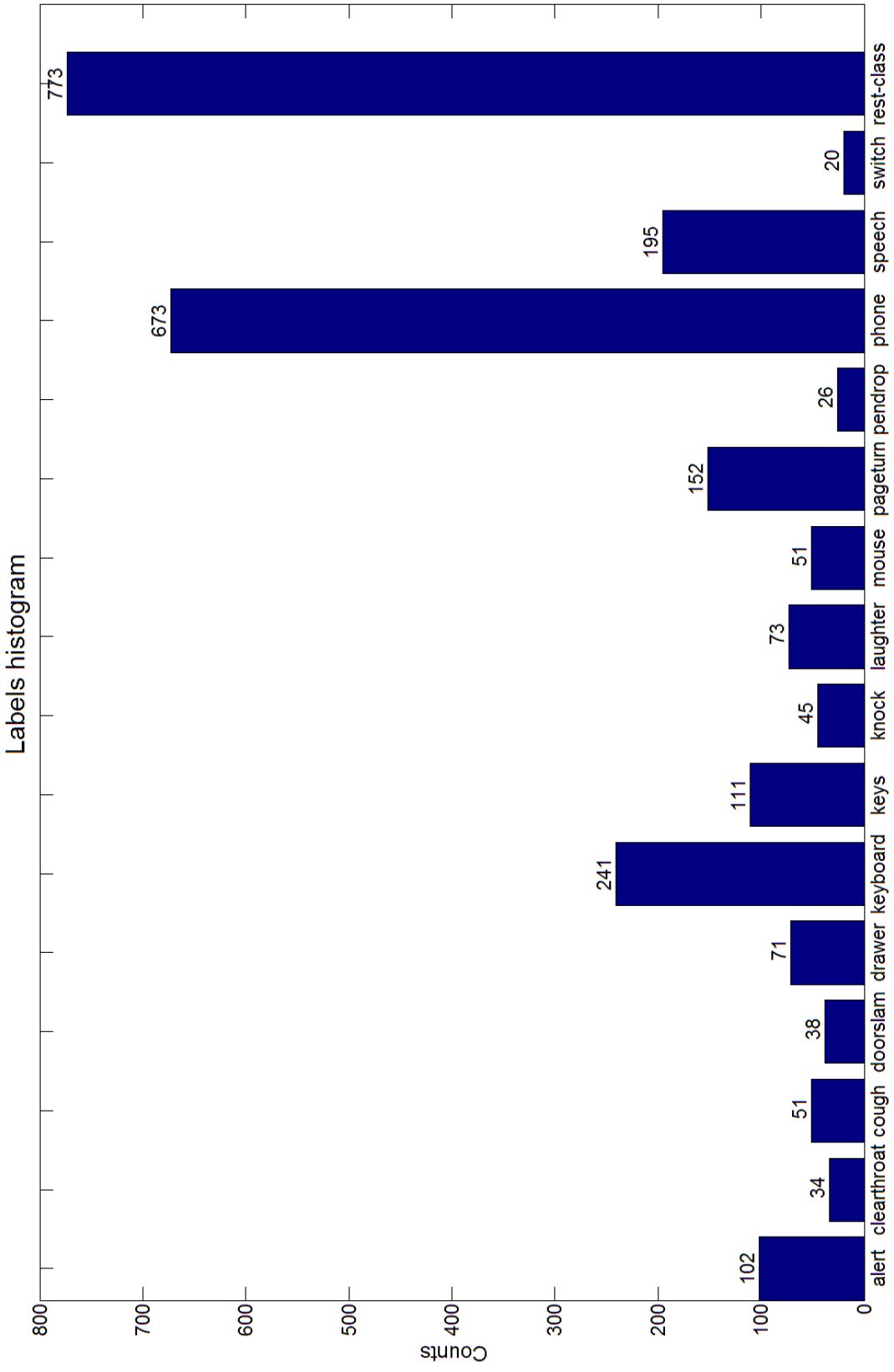

Figure 5.3: Histogram of all the labels

Figures 5.2 and 5.3 present the dataset used in this paper. The class imbalance is enormous. The number of sound samples is the same for each class – 20 .wav samples. For some sound events the feature extractor creates much more data. Algorithms with non-balanced performance metrics would strongly prefer the *phone* and *rest-class* label as the output of any classification. Considering only one dimension, data does not seem sepparable. Many feature spaces (in this work – 124) make it, however, quite possible to discern respective classes. The dimension #90 alone separate fairly good the *switch* class (the least populated one) from the rest classes. The dimensions #102 shows the exceptional value occurred for the *knock* class. In the case of so numerous dimensions count this is not an issue, however, within only the dimensions #102 some naive classifiers (for instance kNN with not so smooth classification) would adjust their classification boundary for this exception sample.

5.2. k-Nearest Neigbhours

The implementation of k-Nearest Neighbours algorithm is provided natively by Matlab. The only parameter chosen for model selection is the value of k in range from 1 to 20.

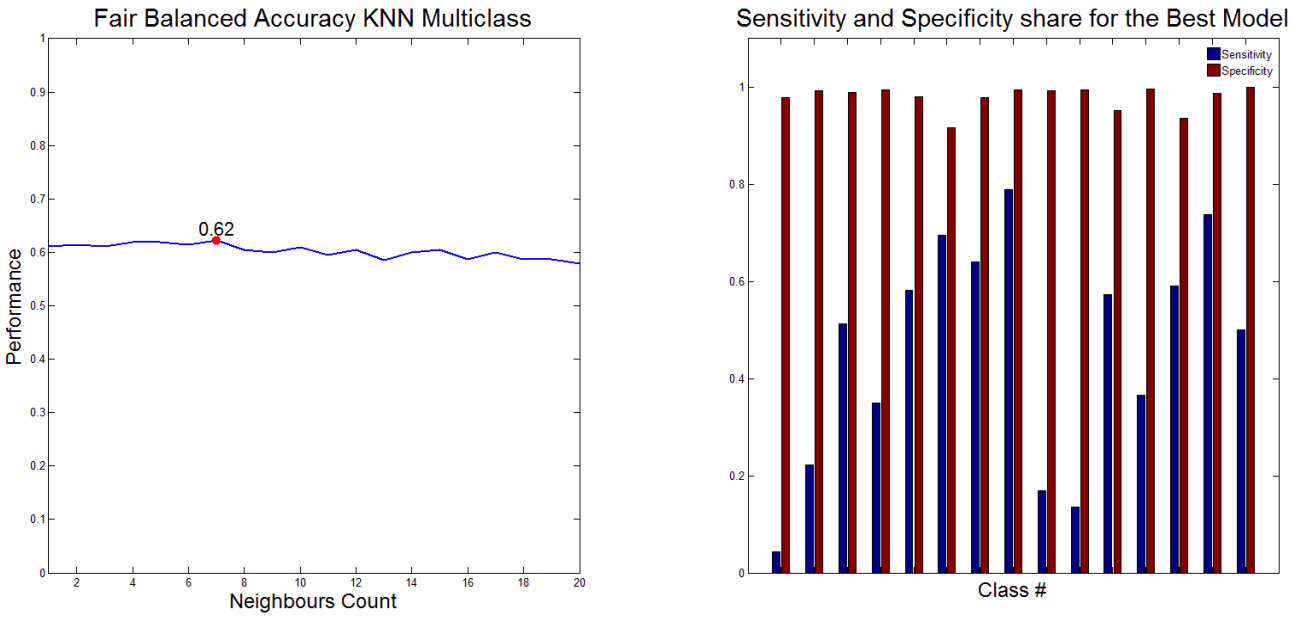

Figure 5.4: Multiclass kNN classification performance.

Multiclass classification performance generalization for different models does not show give better results with growing k parameter value. Anyway, the best Fair Balanced Accuracy value was given for the neighbours count of **7** (Figure 5.4). The performance of **0.62** alone does not give enough information. As expected, sensitivities values are very low (even when using performance metric balancing the specificity and sensitivity).

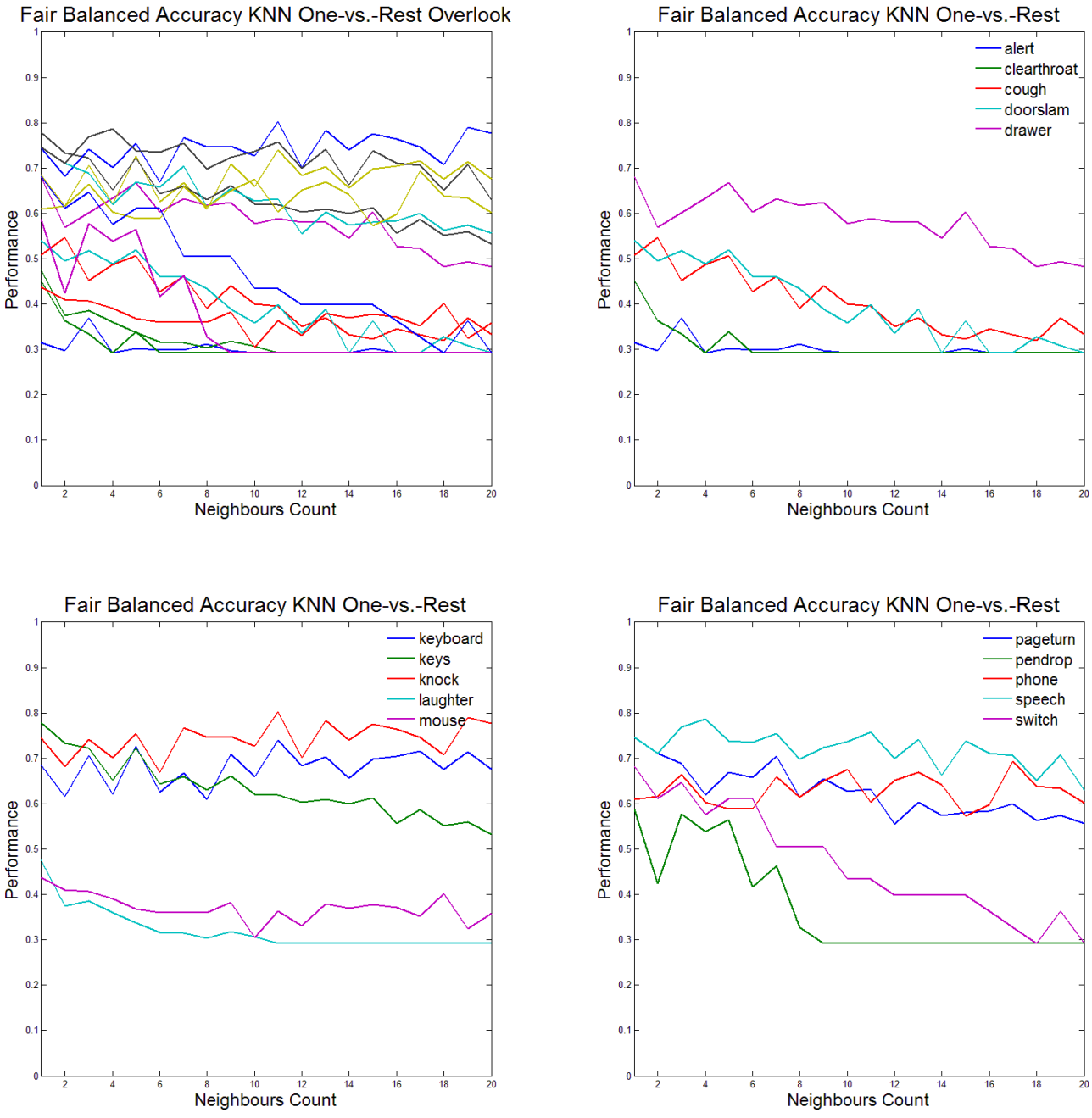

Figure 5.5: One-vs.-Rest kNN classification performance.

Figure 5.5 presents the distribution of performances for each class in One-vs.-Rest kNN classification. Unlike the multiclass classification performance, One-vs.-Rest classification is sensitive to the chosen k parameter value for some classes (especially *pendrop* and *doorslam*). Mostly the lower value of k is preferred with the exception for some cases (for example, $k=17$ preferred for *phone* classifier). However, the difference in performance in these cases was not so large.

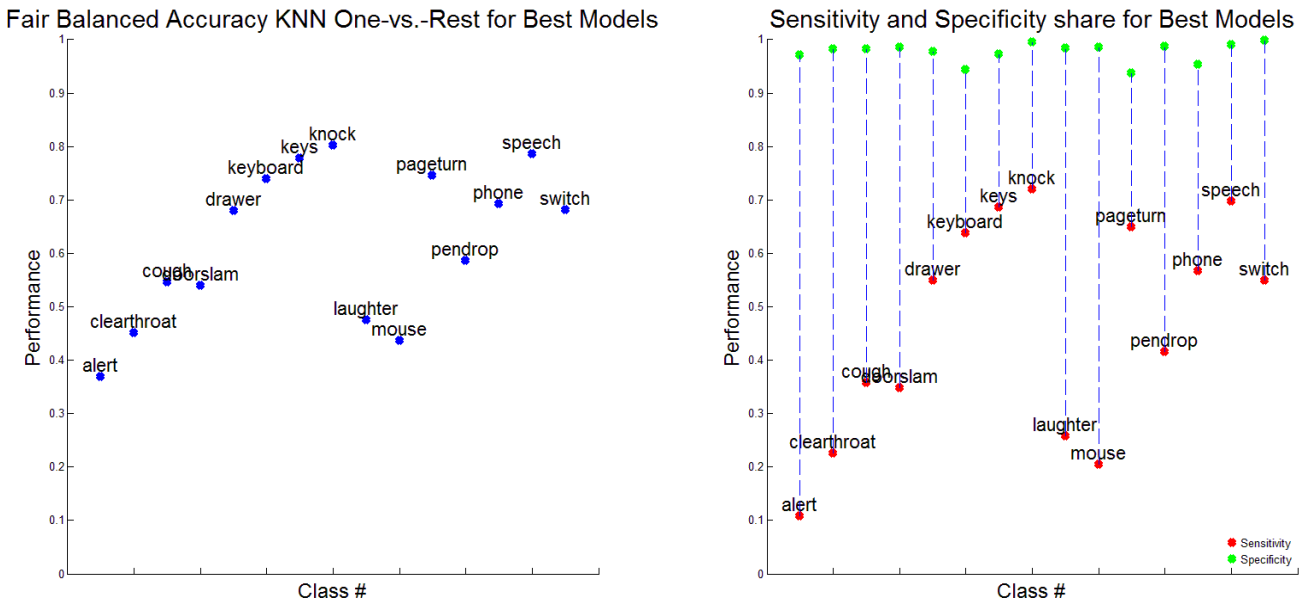

Figure 5.6: One-vs.-Rest kNN classification performance.

From all the One-vs.-Rest classifiers the best in each class was chosen. The respective performances and sensitivities measures are scattered in Figure 5.6. Like in multiclassification approach, obtained specificities are quite large and sensitivities very low. However, the minimum value of sensitivity in this case is about two times bigger than in multiclassification case (it happens for the same class – *alert*). Interestingly, the distribution of per-class performances is consistent with the distribution of sensitivities (because of almost constant specificity value). Higher values of k are sensitive to class imbalances, what is here shown.

5.3. Support Vector Machine

The implementation of SVM algorithm comes from LIBSVM for Matlab package [*Chang, Lin 2011*]. Sticking to the recommendations (with regards to the computation time required) following parameter value ranges were used in grid search mode to generate the models [*Hsu, Chang, Lin 2003*]:

- $K=\{0,2\}$ kernel types (LIBSVM notation) – linear and RBF kernel,
- $C=\{10^{-5},10^{-3},0.1,10,10^{3},10^{5},10^{7},10^{9},10^{11}\}$ – cost values used to control the misclassification rate on the margin of the hyperplane being calculated,
- $\gamma=\{10^{-13},10^{-11},10^{-9},10^{-7},10^{-5},10^{-3},0.1,10\}$ – gamma values used for non-linear classification with RBF kernel. For linear kernel $\gamma=0$ was applied since it is the RBF function parameter in this case (to control the shape of hyperspheres).

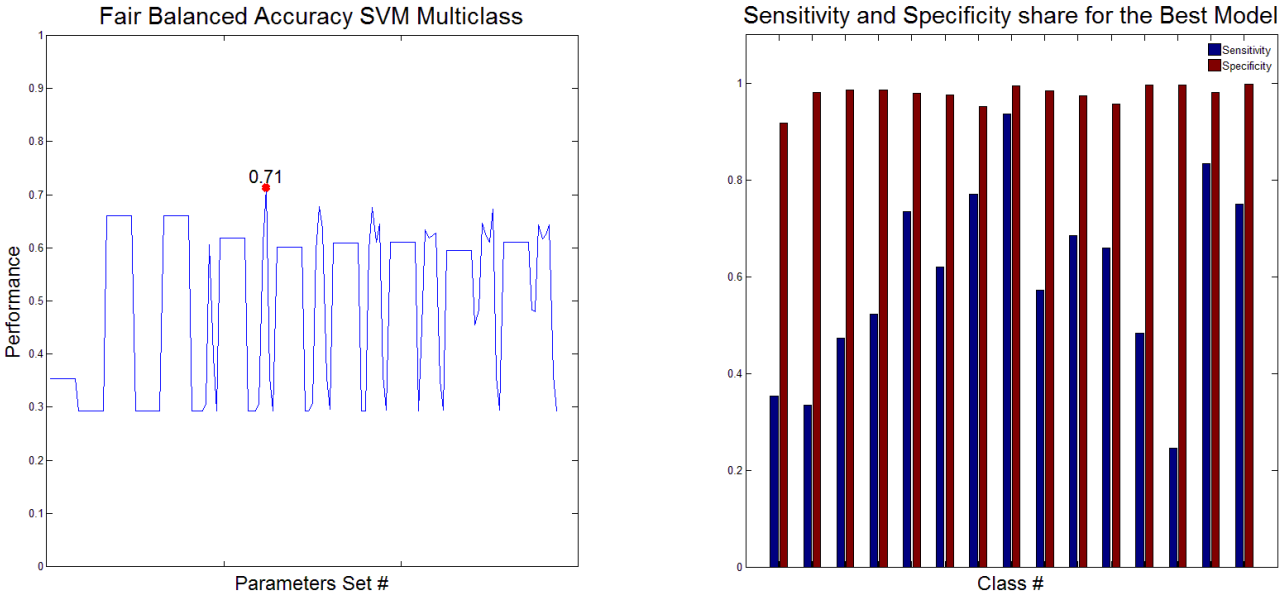

Figure 5.7: Multiclass SVM classification performance.

LIBSVM implementation provides natively the multiclass classification. Internally it uses One-vs.-One version of extending two-class classifiers into multiclass case. Figure 5.7 shows achieved performances for given set of parameters and sensitivity and specificity distributions for the best model. A few sensitivities are reasonably large but still some of them are quite low. Surprisingly the lowest value of sensitivity has the classification of the most populated class – *phone* (excluding *rest-class*).

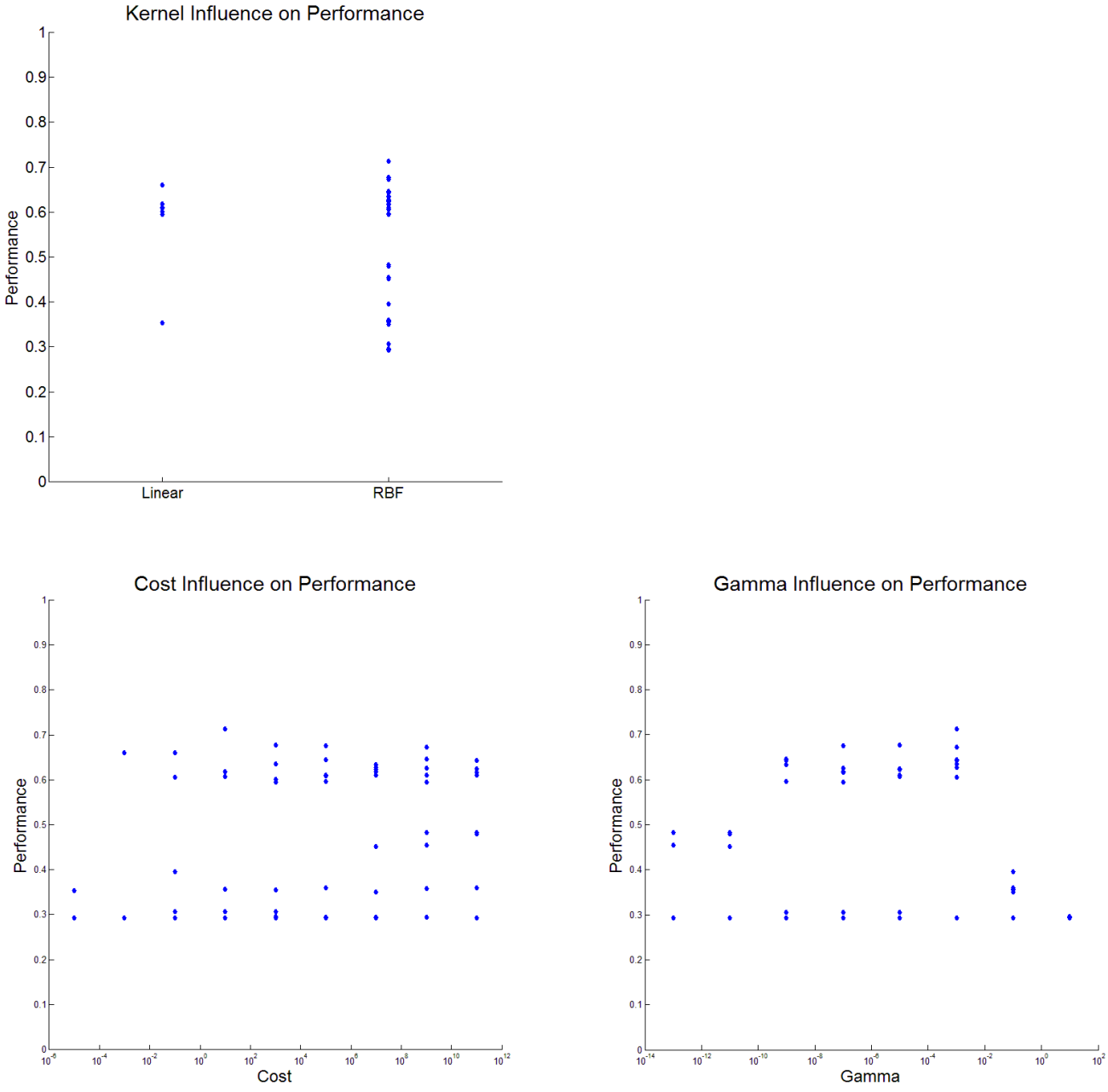

Figure 5.8: Multiclass SVM classification performance.

Searching for the influence of respective parameters on the performance in Figure 5.8 results in conclusion that RBF kernel should be preferred for that type of data. When number of features is relatively large compared to the number of instances, mapping the data to non-linear classifier (using for example RBF kernel) is not needed. The accuracy of the models created by linear and non-linear kernel in that case is comparable [*Hsu, Chang, Lin 2003*]. Actually in that case the number of dimensions – 124 turns out to be not large enough compared to the number of instances – 2656 in order to neglect the benefits of RBF (or any) kernel. Staying with the RBF kernel (where both Cost and Gamma parameter matter) the, influence of both Cost and Gamma values on performance should not be analysed separately. Interestingly, performance values are clearly distributed in sets of values; for medium values of Gamma only the extreme values of performances were acquired. More inspection is needed, especially more precise grid search.

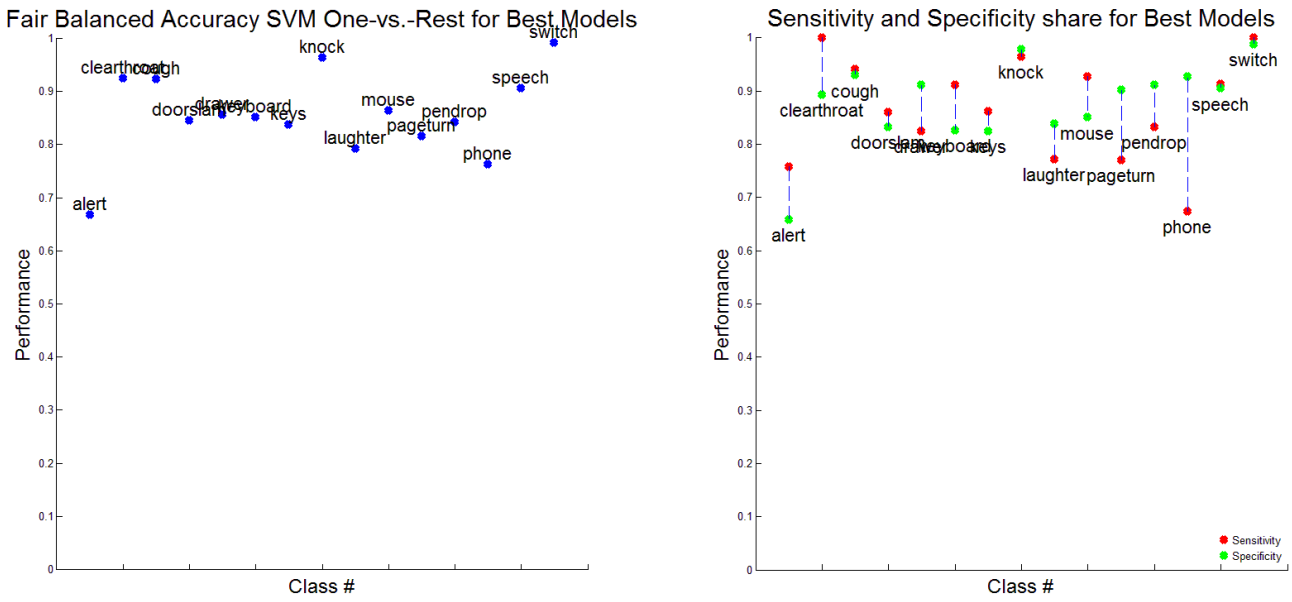

Figure 5.9: One-vs.-Rest SVM classification performance.

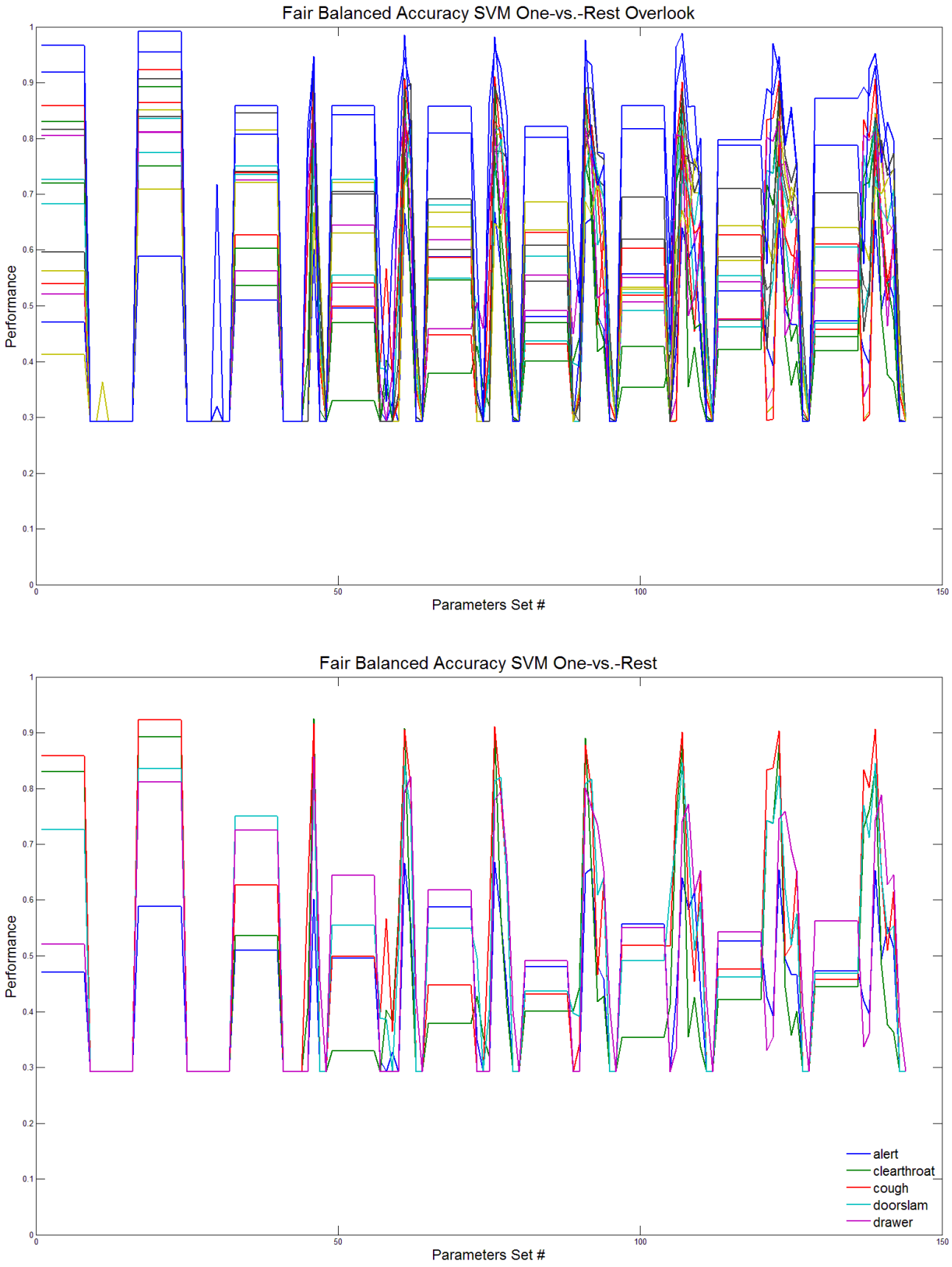

Figure 5.10: One-vs.-Rest SVM classification performance.

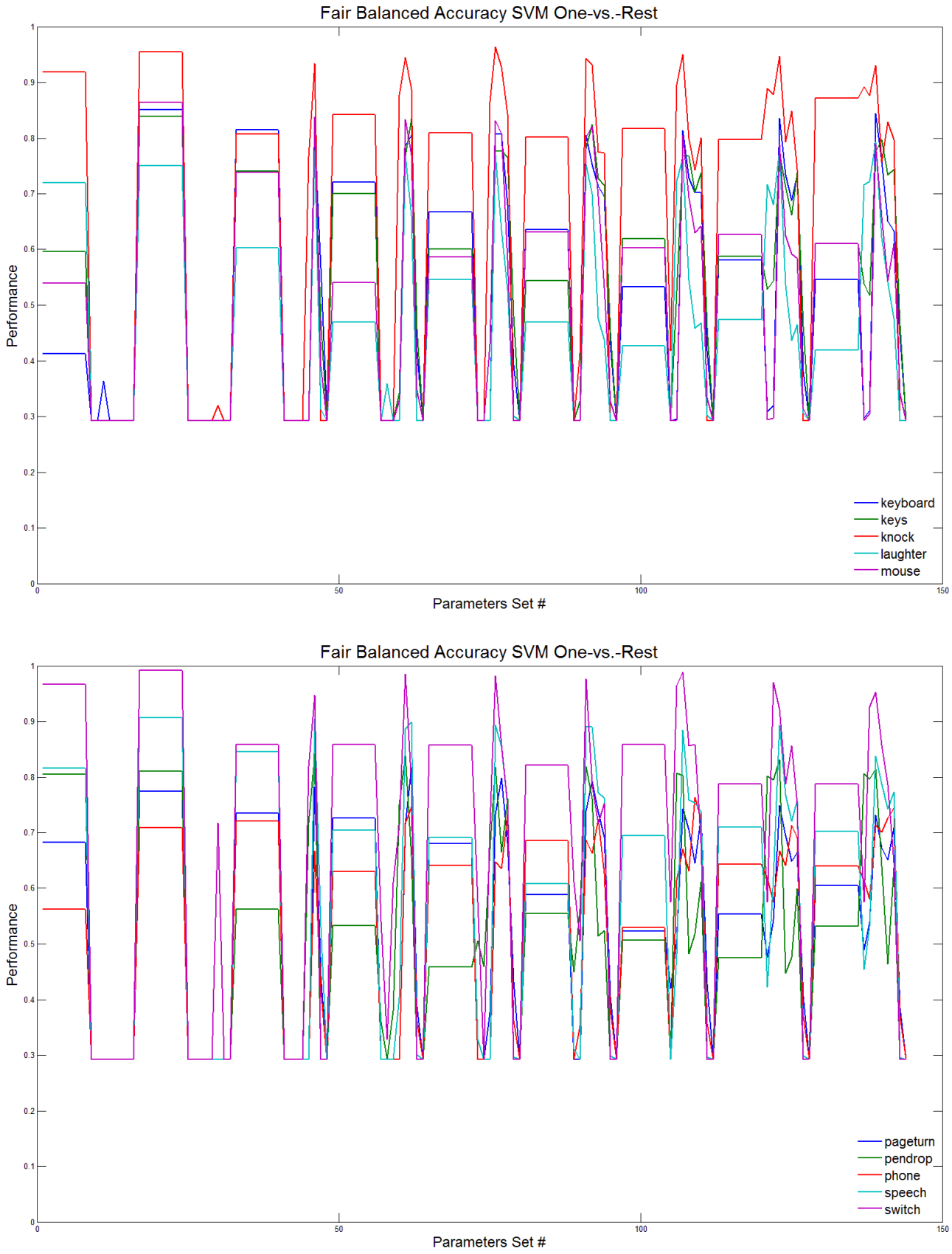

Figure 5.11: One-vs.-Rest SVM classification performance.

Comparing with distribution of per-class performances for One-vs.-Rest kNN, the mean performance as well as overall variance in SVM case is better (shown in Figure 5.9). Interestingly, some of the best-per-class models have higher sensitivity than specificity values. Especially sensitivities are way much better than kNN in binary case.

Figures 5.10 and 5.11 show achieved performances for each class model given different parameters set in One-vs.Rest classification. Similar to the multiclass case, some sets of parameters block the learning capabilities of the SVM to about 0.3 value performance. Periodic constant value of performance indicates independence (in some range of values) of performance of some parameter (parameter sets are computed using nested loops).

5.4. Robust Soft Learning Vector Quantization

RSLVQ implementation [*Seo, Obermayer 2003*] was tested with all the possible parameters sets. It was decided to measure performance and mutual influence of 3 of them:

- $number\ of\ prototypes\ per class = \{1,2,3,4,5,6,7,8,9,10\}$. During initial tests it was noticed that higher number of prototypes per-class was not affecting performance positively. Therefore only quite small range was kept for main tests.
- $\alpha = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ – learning rate value. Adjusting the size of weights update step one can leap the local maximum of performance to reach a better one.
- $class\ noise\ factor = \{0.001, 0.01, 0.1, 1\}$ – Class noise factor attempts to blur initial prototypes coordinates (spreading them around). This could lead to another convergence points set with, hopefully, better performance measure.

Other important parameters like ε and both alpha and sigma annealing rates were kept constant with respective values: $10^{-6}, 0.9, 0.9$. ε is responsible for stopping-condition of the algorithm. Small enough difference between performances after weights update is considered as the convergence point. Annealing rates are factors decreasing step size after each update. For fine search they should be kept close to 1 or parameterized.

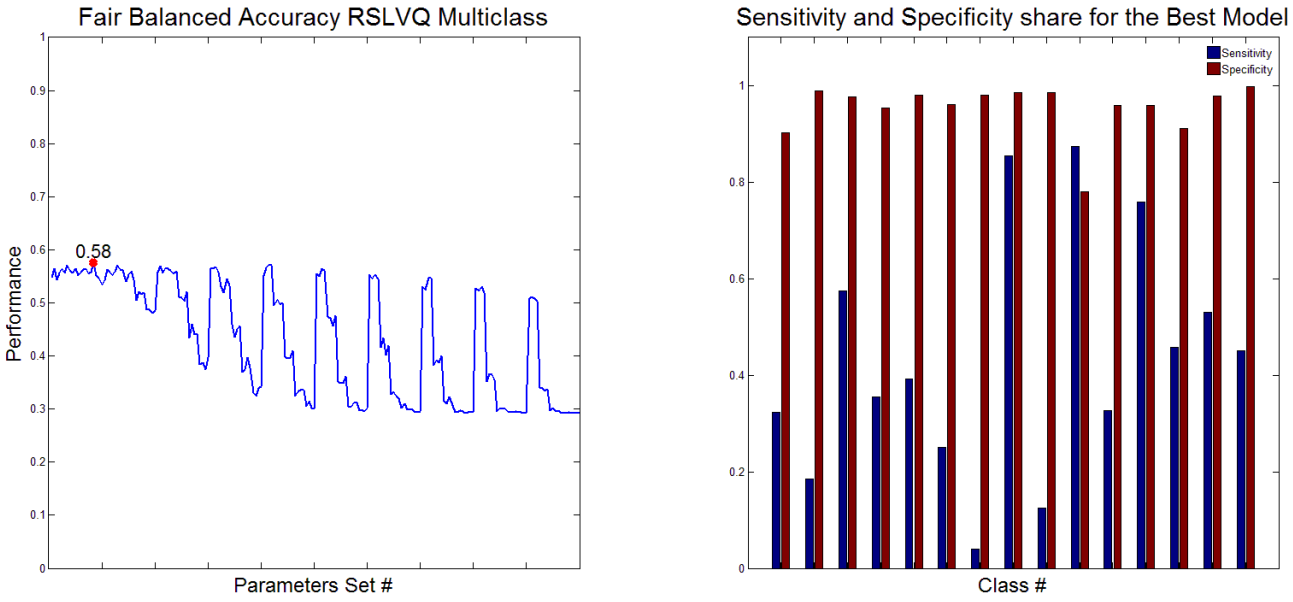

Figure 5.12: Multiclass RSLVQ classification performance.

The Fair Balanced Accuracy plot in Figure 5.12 affirms the decision about low number of prototypes per-class to be preferred. The most outer for-loop of parameters generation is the iteration over $number\ of\ prototypes\ perclass$. Therefore the first tenth of the presented axis presents all parameters sets with $number\ of\ prototypes\ perclass = 1$. Like in kNN case, the sensitivities distribution for multiclass classification is (in most cases) almost unacceptable.

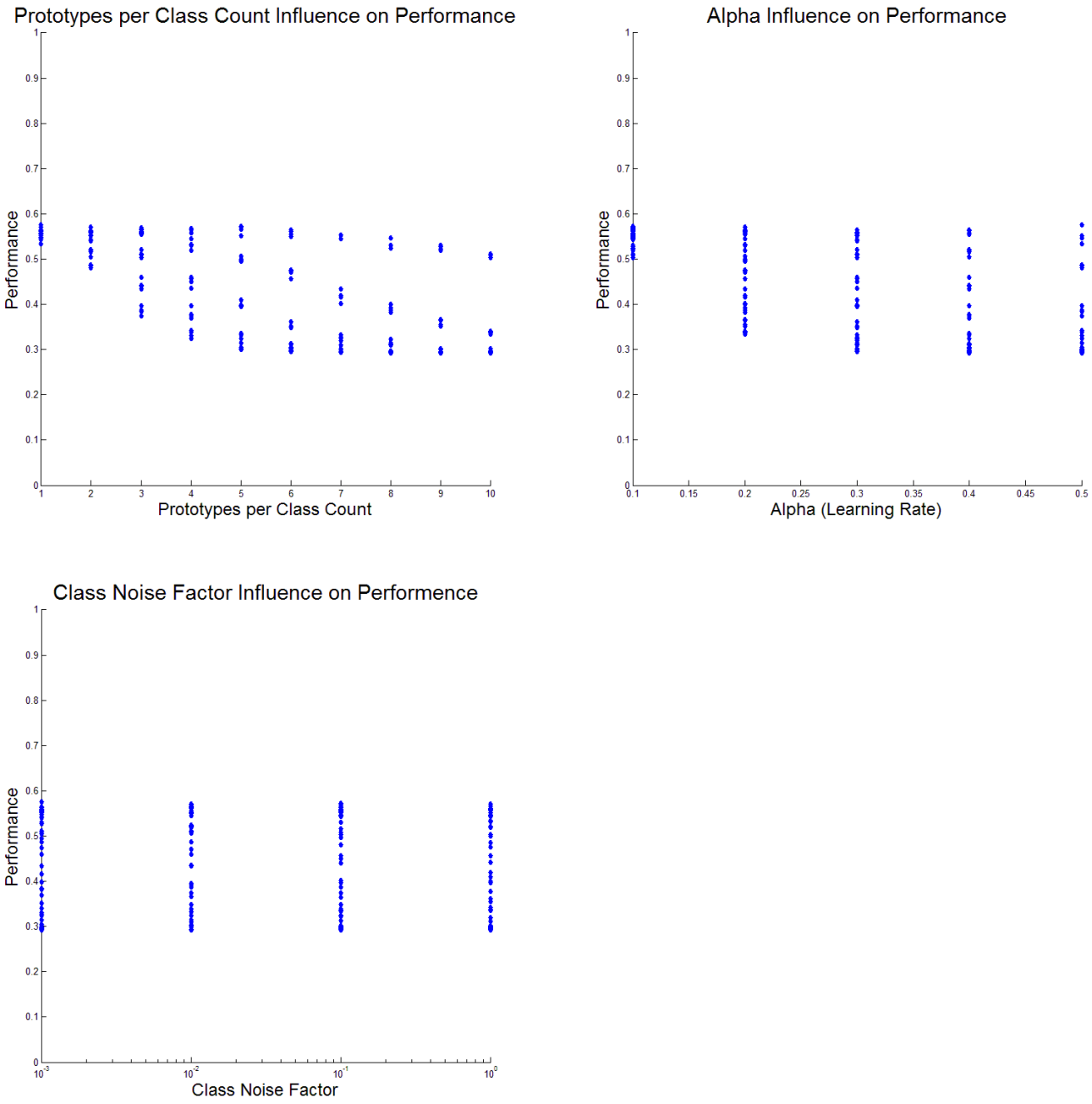

Figure 5.13: Multiclass RSLVQ classification performance.

As expected, low alpha values are preferred (Figure 5.13). However, lowering alpha increases training time of RSLVQ. The least complex model achieved the best distribution of performances; more complexed models were converging into less optimal prototypes sets. Sweeping with values of *class noise factor* has not practically affected performances.

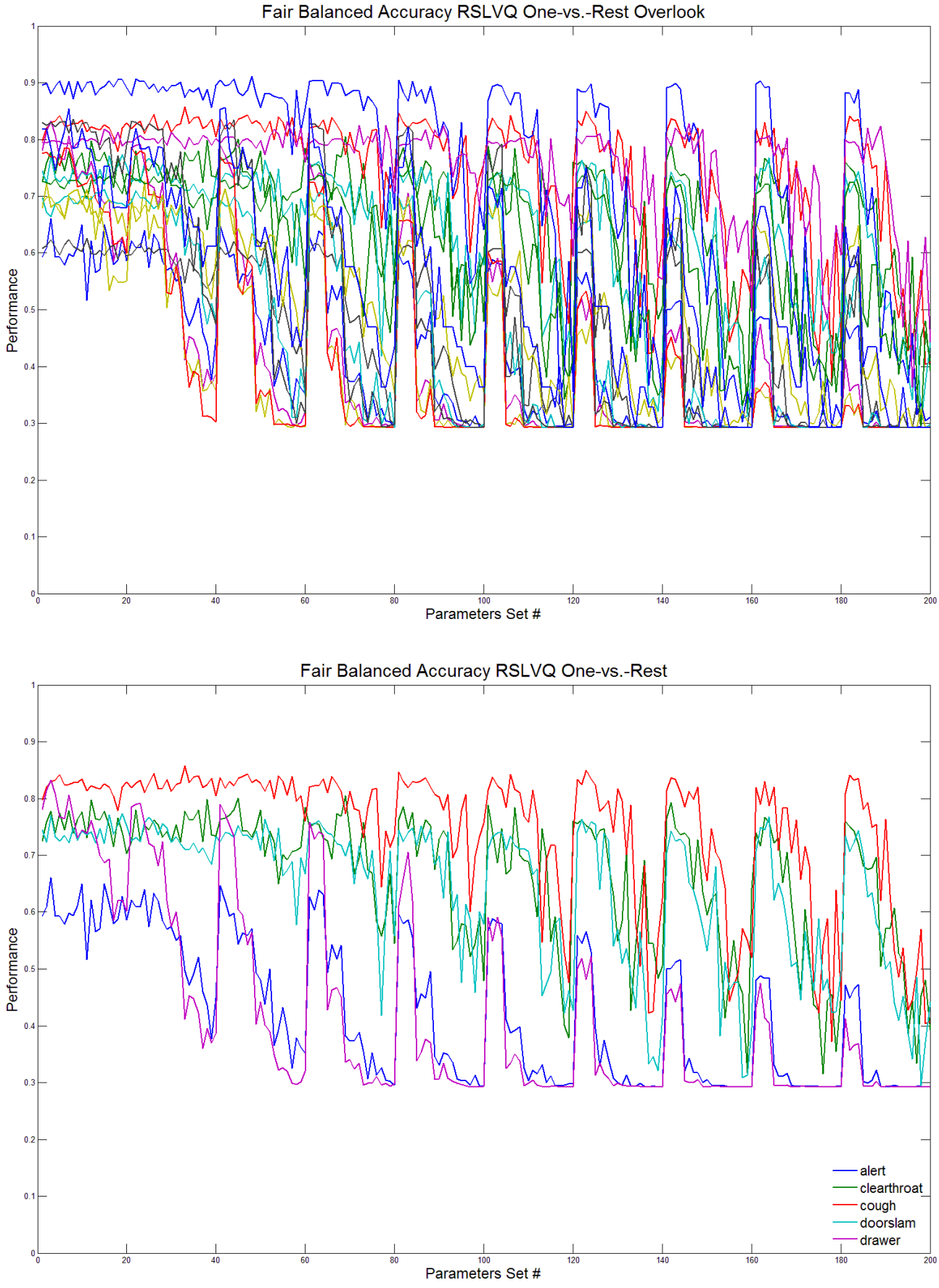

Figure 5.14: One-vs.-Rest RSLVQ classification performance.

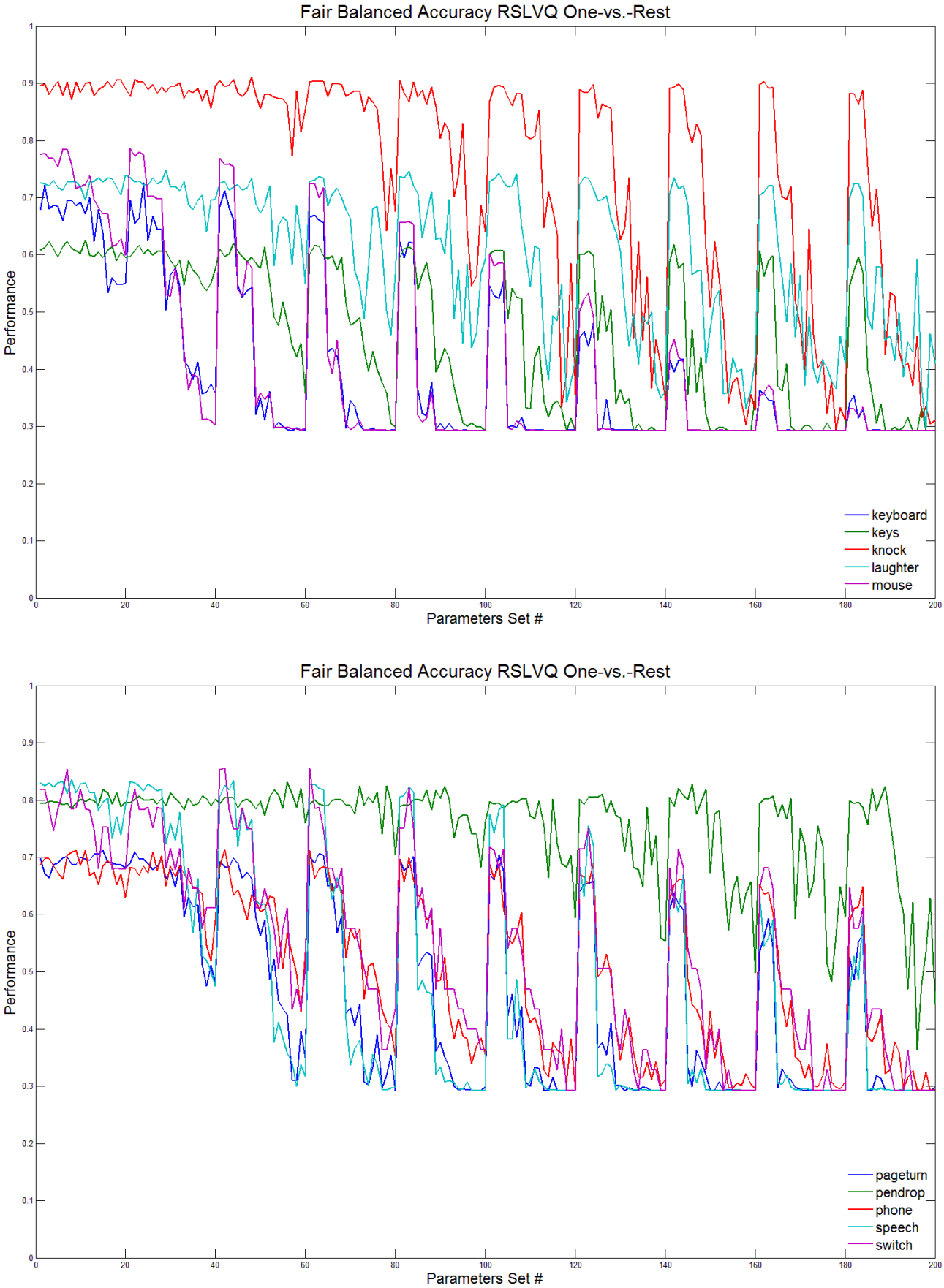

Figure 5.15: One-vs.-Rest RSLVQ classification performance.

Performance distributions for One-vs.-Rest variant of RSLVQ classifier with respect to the parameters set have similar curvature as the performance results for multiclass classification (Figure 5.15). Between classes, shapes of acquired plots are similar, only sometimes shifted. The performance of 0.3 seems to be the minimal achievable value in any given situation (parameters set or class).

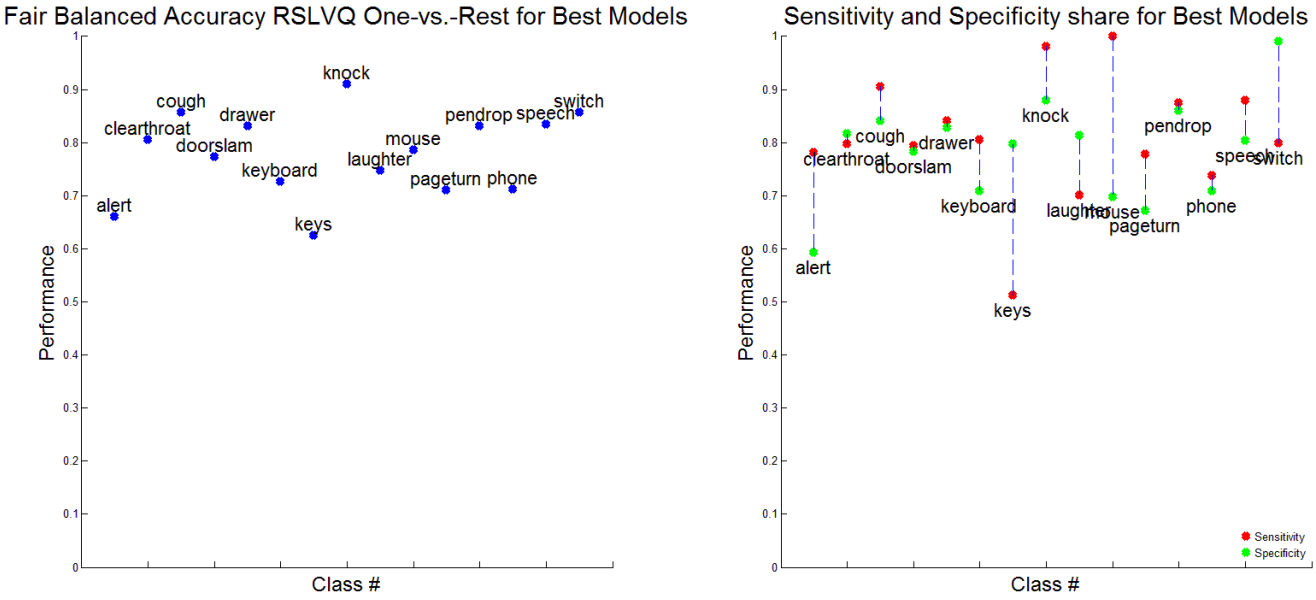

Figure 5.16: One-vs.-Rest RSLVQ classification performance.

Like in SVM One-vs.Rest classification, neither sensitivity nor specificity is favoured in averaged terms. However, some classes stand out a little (like *keys* or *mouse*). For both multiclass and two-class cases class *mouse* was classified with good performance – particularly high sensitivity (compared to other classes).

5.5. Comparison of Results

Since comparing the overall performance score alone does not give much information. In following subchapter emphasis is placed also on sensitivity and specificity comparison.

5.5.1. Multiclass Classification

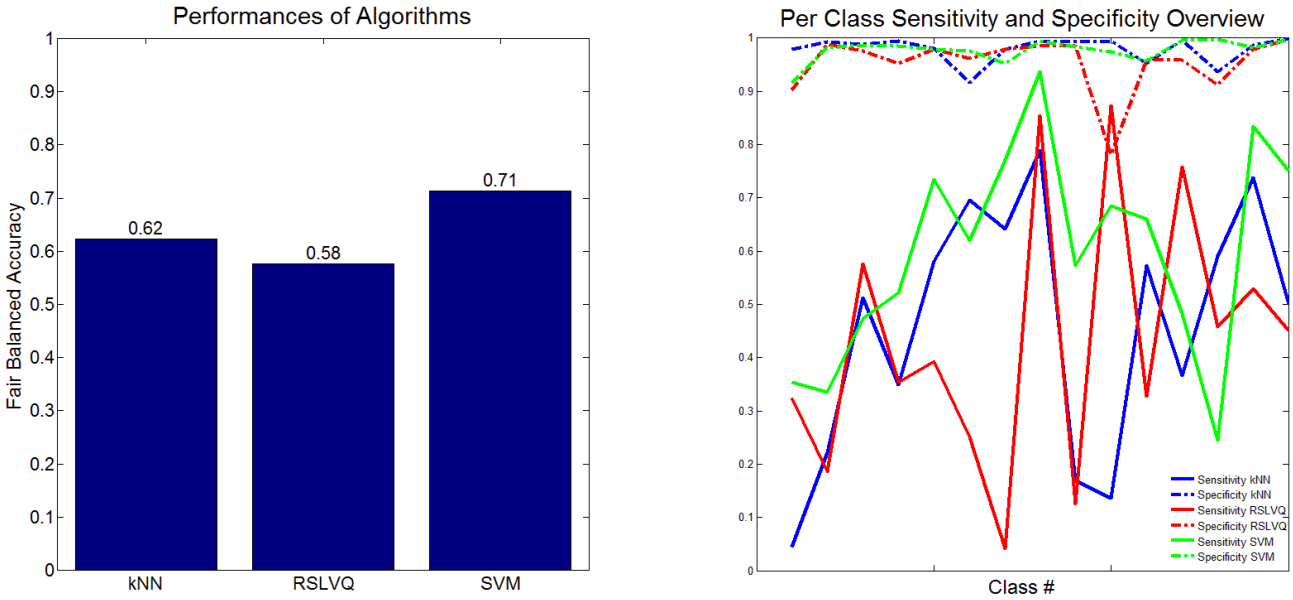

Figure 5.17: Comparison of multiclass classification performance.

SVM outperforms other algorithms in terms of performance (Figure 5.17). Almost in all terms SVM returns better sensitivity values. From the specificity point of view all the algorithms give similar results (with an exception of the previously already mentioned class *mouse* for RSLVQ performing noticeably worse specificity in favour of higher sensitivity).

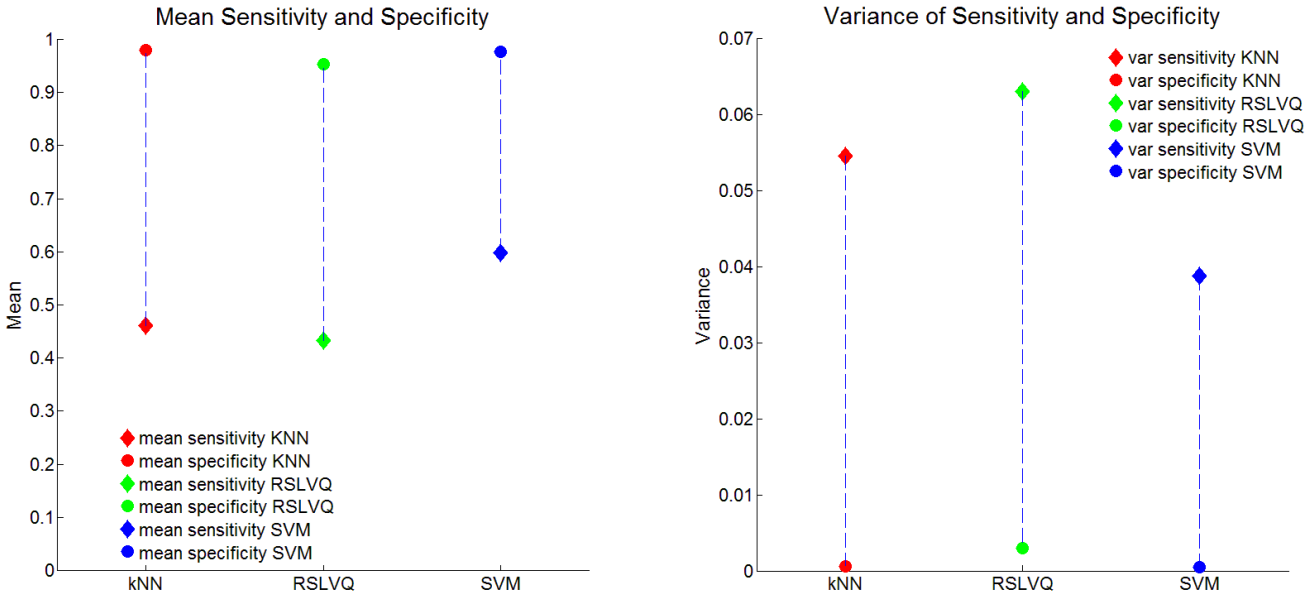

Figure 5.18: Comparison of multiclass classification performance.

In terms of mean value and variance of specificity – all algorithms performed well (scattered in Figure 5.18). Only sensitivities were more diverse. SVM performed especially better than RSLVQ in terms of sensitivity variance. With given requirement of overall better performance SVM algorithm would be preferred.

5.5.2. One-vs.-Rest Classification

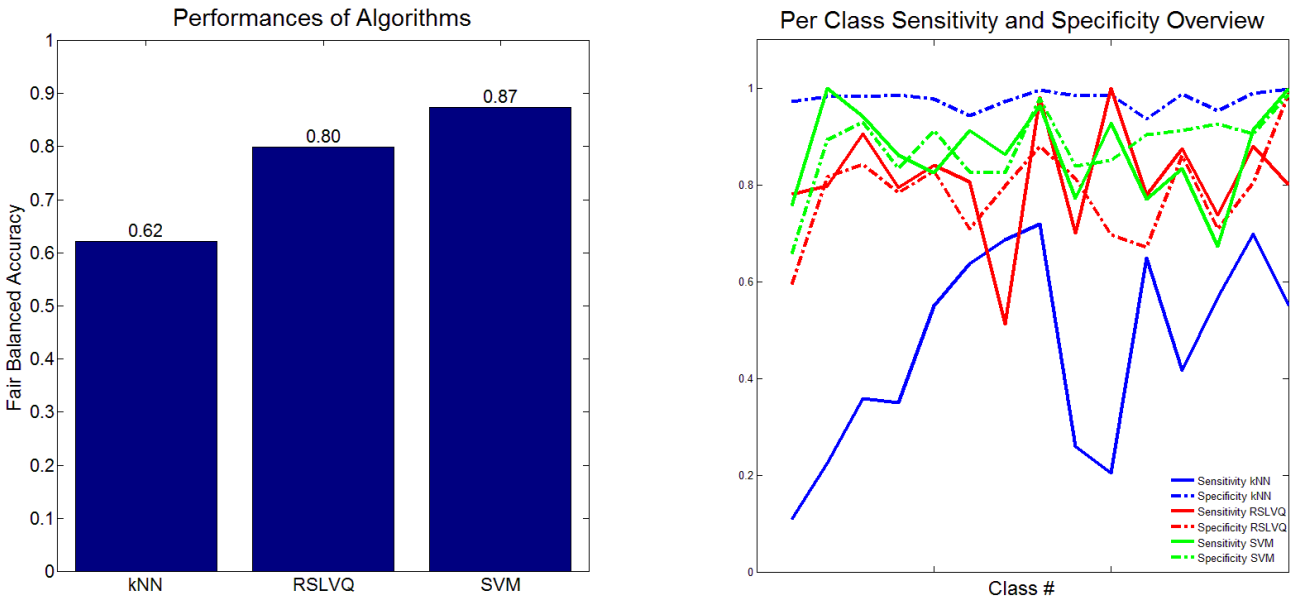

Figure 5.19: Comparison of One-vs.-Rest classification performance.

The worst performing algorithm is in this case easily distinguishable (Figure 5.19). Even having ability to better separate the classes (only 2, not 16 like in multiclass case) kNN does not show any improvement; apart from having largest specificity values, kNN achieved far poorer sensitivities. Still SVM wins in terms of overall performance. However, RSLVQ is this time quite close (with lowest sensitivity for *keys* class, the same as in multiclass case).

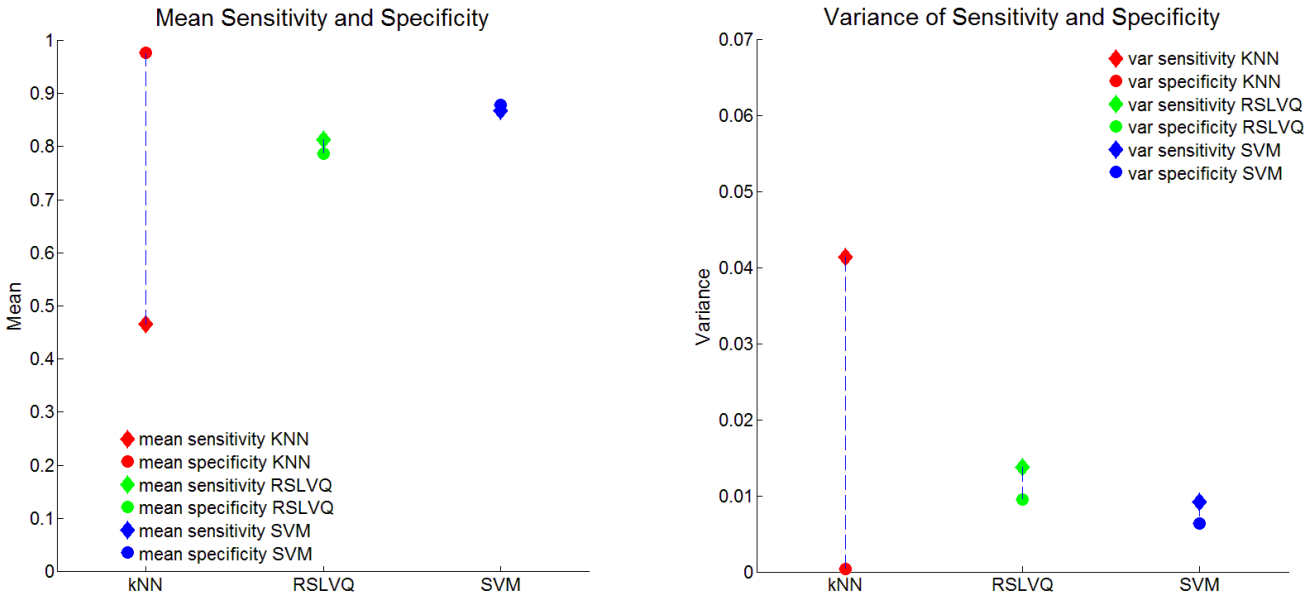

Figure 5.20: Comparison of One-vs.-Rest classification performance.

Similar to multiclass classifiers, SVM performed the best. Although kNN achieved the highest specificity rates, its sensitivities were far worse than other algorithms (both in terms of the average value and variance). Applying One-vs.-Rest classification lowered the variance of both performance rates significantly.

5.5.3. Time and Memory Consumption

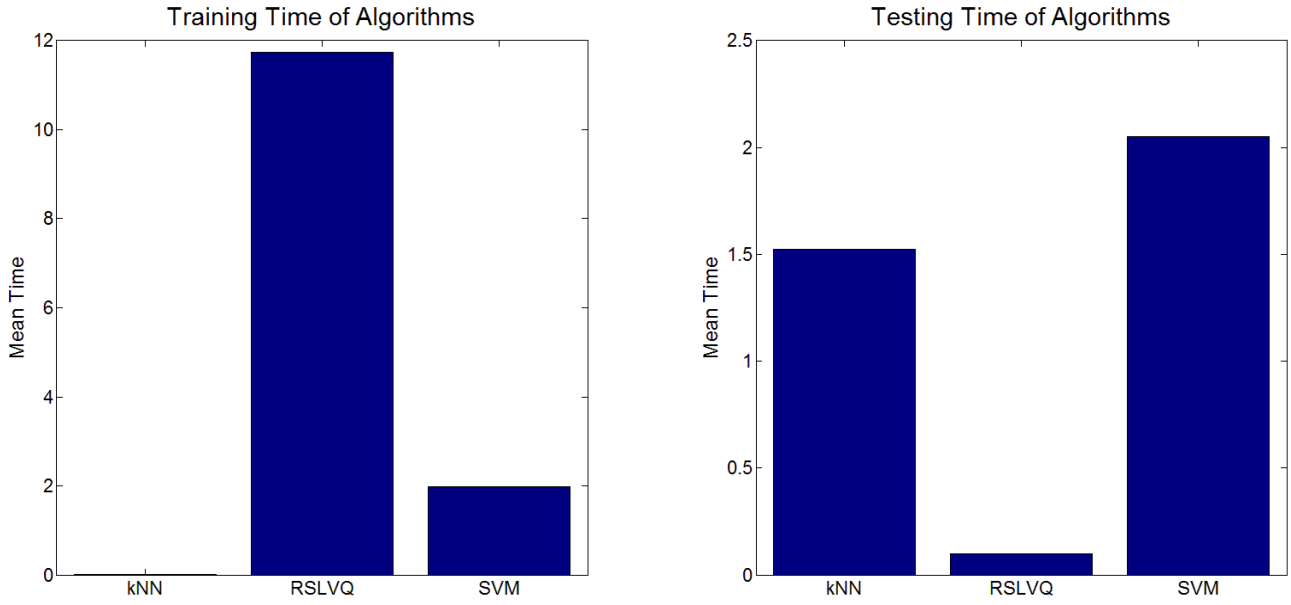

Figure 5.21: Training and testing time on the whole dataset.

Despite being almost always a runner-up in comparison with SVM, RSLVQ has one division where it excels – testing time of classification. Training and testing times are practically not comparable between themselves in case of ratio, average or any other measure (it strictly depends on the available data and desired usage). Training time for kNN algorithm is stupendously low. This results from actually no training part per se, only "gathering" the information of the whole training data. By attempting to generalize input data, the number of prototypes in RSLVQ is scarce compared to all prototypes in kNN model. Because of RBF kernel chosen as parameter for the best model, SVM requires more time to predict label. For linear cases hyperplane can be computed in feature space (SVM implementations focused on linear kernel), whereas for RBF each input has to be translated into non-linear space.

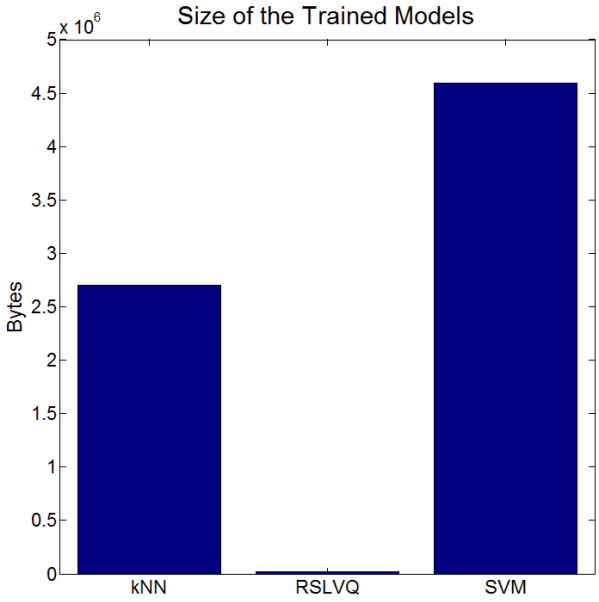

Figure 5.22: Memory usage of a single multiclass model.

By design kNN stores the whole training dataset. With growing dataset the size of kNN models grows linearly. When the memory restriction is given this characteristic of kNN would exclude it instantly. Memory usage of RLSVQ depends mostly on the value of *number of prototypes perclass* parameter (actually also on the size of the feature space; number of dimensions). The size of RSLVQ model does not depend on the number of training instances. SVM model turned out to be quite large in this case. It is connected with usage of rather sophisticated LIBSVM library which supports many configuration options. For linear kernel storing only hyperspace parameters would reduce the size of the model substantially.

6. CONCLUSION

The purpose of this work was to compare Robust Soft Learning Vector Quantization algorithm on the features extracted from the real-world data. As a reference point two popular classification algorithms were used: k-Nearest Neighbour and Support Vector Machine. Whereas achieved performance values for RSLVQ were not outstanding, a very low classification time metrics gives RSLVQ a real hope for implementations in real-time classifiers (what sound events classifier should actually be for).

6.1. Discussion and Further Work

- *Findings:* A big potential in RSLVQ has been found. Because of the meaningfully slower prediction time (remaining comparable performance to SVM in One-vs.-Rest classification) this algorithm can be implemented in real-time classifiers.
- *Imbalanced Data Issues:* The problem of class imbalanced data is ubiquitous in real-world data [*Japkowicz, Stpehen 2002*]. Therefore attention should be given on that matter when applying any Machine Learning algorithm to learn on the given data. Oversampling the dataset was excluded because of the unusably large training and testing times. Undersampling should be considered (after proving that the representation of most populated classes had redundant repeated instances). However, it is not certain if any improvement would occur:

  (...) there is some evidence that re-balancing the class distributions artificially does not have much effect on the performance of the induced classifier, since some learning systems are not sensitive to differences in class distributions. (...) some data sets are immune to class imbalance problem [Guo, Dong, Yang, Zhou 2008].
- *Results Averaging approaches:*Macro- and Micro-averaging are two approaches of averaging the performance measure for multiclass classification case [*Asch 2013*]. In Macro-averaged measure score the per-class evaluation measures are averaged. The evaluation measure is in general a function of specific cells from confusion matrix for each class (TP, FP, FN, TN). The Macro-averaged results are computed as follows (for a given evaluation measure function B):

$$B_{macro}=\frac{1}{q}\sum B(TP_{c_i},FP_{c_i},TN_{c_i},FN_{c_i}) \quad .$$

Micro-averaged measure calculates the value of the evaluation function B only once. The arguments from the confusion matrix are averaged per-class values of error rates (TP, FP, FN, TN):

$$B_{micro}=B(\frac{1}{q}\sum TP_{c_i},\frac{1}{q}\sum FP_{c_i},\frac{1}{q}\sum TN_{c_i},\frac{1}{q}\sum FN_{c_i}) \quad .$$

The approach chosen in this work is somewhere between (as shown in Important Implementation Aspects chapter). Micro-averaging should give better results on imbalanced data [*Asch 2013*].

- *Results reliability:* For the sake of simplicity and computational restrictions data normalization was performed only just before applying whole dataset to the test framework. To achieve better and more reliable results normalization should be performed before each model training (for each fold).

- *Parameter search refinement:* Parameters sets selection should be refined. Dense parameter search around high performance values would exploit more successfully local extrema. Learning rates parameters (and most of all epsilon condition stopping the prototypes update) around extrema should be adjusted to more precision at the cost of learning time.

- *Training instances scarcity:* On account of so scarce representation of some classes, splitting the dataset into stratified folds can influence the performance significantly. For the class *switch* there were only 20 training instances available (one instance for each file as a matter of fact). Nested 5-fold Stratified Cross-Validation assigns in this case 4 instances of *switch* class for each fold in the outer loop. The inner loop attempts to assign 4 provided instances to its 5 inner folds. Therefore one fold has no representation of *switch* class what impacts the performance value and reliability of the acquired model.

- *Further reserch:* The first approach that should be carried out is to analyze the influence of σ parameter on the RSLVQ performance measures for sound events.

6.2. Summary

All the algorithms were tested on the same framework for Nested k-Fold Stratified Cross-Validation. Performance metric used in model selection was improved accuracy value (Fair Balanced Accuracy) promoting similar values of sensitivity and specificity rates. Features extracted from audio events samples were applied in multiclass and One-vs.-Rest classification cases. Different parameters values for all algorithms were analyzed to find the best fitting models. RSLVQ turned out to outperform other algorithms in prediction time. For future work more research and knowledge about the data is needed as well as refined parameters tests.

7. BIOGRAPHY

Mary Jo Foley, *Microsoft's 'Cortana' alternative to Siri makes a video debut*, 4.03.2014, http://www.zdnet.com/microsofts-cortana-alternative-to-siri-makes-a-video-debut-7000026987/

Apple Launches iPhone 4S, iOS 5 & iCloud, 30.08.2014, http://www.apple.com/pr/library/2011/10/04Apple-Launches-iPhone-4S-iOS-5-iCloud.html

The Aircraft, Direct Voice Input feature, 30.08.2014, http://www.eurofighter.com/the-aircraft

B. H. Juang, Lawrence R. Rabiner, *Automatic Speech Recognition – A Brief History of the Technology Development*, Elsevier Encyclopedia of Language and Linguistics 2005, ISBN 0-08-044299-4

DeLiang Wang, Guy J. Brown, *Computational Auditory Scene Analysis*, IEEE 2006, ISBN 978-0-471-74109-1

Sambu Seo, Klaus Obermayer, *Soft Learning Vector Quantization*, Neural Computation 2003

T. Kohonen, *Learning Vector Quantization*, Otaniemi, 1986

E. Fix, J.L. Hodges, *Discriminatory Analysis, Nonparametric Discrimination Consistency properties*, Randolph Field 1951

David M.J. Tax, Robert P.W. Duin, *Using two-class classifiers for multiclass classification*, IEEE 2002

http://www.visual-paradigm.com/

Tobias May, Remi Decorsière, Chungeun Kim, Armin Kohlrausch , *Two!EARS. The auditory front-end framework - User manual*, 2014

IEEE AASP Challenge: Detection and Classification of Acoustic Scenes and Events, 11.01.2015, http://c4dm.eecs.qmul.ac.uk/sceneseventschallenge/

Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press Oxford 1995

Joe McCarthy, *Python for Data Science*, 24.12.2014, http://nbviewer.ipython.org/github/gumption/Python_for_Data_Science/blob/master/2_Data_Science_Basic_Concepts.ipynb

Vincent Labatut, Hocine Cherifi, *Accuracy Measures for the Comparison of Classifiers*, Galatasaray University, Computer Science Department

Keith Gibbs, *Accuracy and* Precision, 28.12.2014, http://www.schoolphysics.co.uk/age16-19/General/text/Accuracy_and_precision/index.html

Ivo Trowitzsch, *Two!Ears Dynamic Auditory Scene Analysis*, 11.01.2015, https://prezi.com/gcks55qh9wyz/twoears-2/

Bill Dimm, *Predictive Coding Performance and the Silly F1 Score*, 21.12.2014, http://blog.cluster-text.com/2013/05/08/predictive-coding-performance-and-the-silly-f1-score/

J. Gorodkin, *Comparing two K-category assignments by a K-category correlation coefficient*, Computational Biology and Chemistry 2004

Nitesh V. Chawla, *The Data mining and knowledge discovery handbook*, Sptringer 2005, ISBN 0-387-24435-2

Perry Sprawls, *Image Characteristics and Quality*, 02.01.2015, http://www.sprawls.org/ppmi2/IMGCHAR/

Tom Fawett, *ROC Graphs: Notes and Practical Considerations for Data Mining Researches*, HP Laboratories Palo Alto 2004

David J. Hand, Robert J. Till, *A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems*, Kluwer Academic Publishers 2001

Ron Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, International Joint Conference on Artificial Intelligence 1995

B. Elfron, *Estimating the error rate of a prediction rule: improvement on cross-validation*, Journal of the American Statistical Association 78 1983

Paul Martin, *k-fold cross validation*, 03.01.2015 http://homepages.inf.ed.ac.uk/pmartin/tutorial/case_studies.html

Damjan Krstajic, Ljubomir J Buturovic, David E Leahy, Simon Thomas, *Cross-validation pitfalls when selecting and assessing regression and classificaiton models*, Journal of Cheminformatics 2014

David M.J. Tax, Robert P.W. Duin, *Using two-class classifiers for multiclass classification*, 2002 IEEE, ISBN 1051-4651/02

Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, *A Practical Guide to Support Vector Classification*, 2003

MathWorks, k-nearest neighbor classification, 01.06.2015, http://de.mathworks.com/help/stats/classificationknn-class.html

scikit-learn: machine learning in Python, 3.4 Nearest Neighbors, 03.01.2015, http://scikit-learn.org/0.11/modules/neighbors.html

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda,Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg, *Top 10 algorithms in data mining*, Springer-Verlag London Limited 2007

Corinna Cortes, Vladimir Vapnik, *Support-Vector Networks*, Machine Learning 1995

I. Guyon, B. Boser, V. Vapnik, *Automatic Capacity Tuning of Very Large VC-dimension Classifiers*, Advances in Neural Information Processing Systems 1993

Simon Haykin, *Neural Networks a Comprehensive Foundation*, Pearson Education 1999, ISBN 81-7808-300-0

T. Kohonen, *Self Organising Maps*, Springer 1997

Petra Schneider, Michael Biehl, Barbara Hammer, *Hyperparameter Learning in Robust Soft LVQ*, European Symposium on Artificial Neural Networks 2009

Michael Quinion, *Garbage in, garbage out*, 11.01.2015

http://www.worldwidewords.org/qa/qa-gar1.htm

Ivo Trowitzsch, *Two!Ears Computational Auditory Scene Analysis*, 11.01.2015, https://prezi.com/na45fzj_ukcz/twoears-1/

Simon Ciba, *Automatic Classication of Everyday Sound Events after Preprocessing by a Binaural Auditory Model*, Studienarbeit, Technische Universitat Berlin, 2011

Peter Dallos, *Response Characteristics of Mammalian Cochlear Hair Cells*, The Journal of Neuroscience 1985

Jason Heeris, *Gammatone Filterbank Toolkit 1.0*, 11.01.2015. http://detly.github.io/gammatone/

Petra Schneider, Tina Geweniger, Frank-Michael Schleif, Michael Biehl, Thomas Villmann, *Multivariate Class Labeling in Robust Soft LVQ*, European Symposium on Artificial Neural Networks 2011

A.W. Witoelar, A. Ghosh, J.J.G. de Vries, B. Hammer, M. Biehl, *Window-based Example Selection in Learning Vector Quantization*, Neural Computation 2010

Chih-Wei Hsu, Chih-Jen Lin, *A Comparison of Methods for Multiclass Support Vector Machines*, IEEE Transactions on Neural Networks, 2002

John C. Platt, Nello Cristianini, John Shawe-Taylor, *Large Margin DAGs for Multiclass Classification*, MIT Press 2000

Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

N. S. Altman, *An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression*, The American Statistician 1992

D. Coomans, D.L. Massart, *Alternative k-Nearest Neighbour Rules in Supervised Pattern Recognition: Part 1. k-Nearest Neighbour Classification by Using Alternative Voting Rules*, Analytica Chimica Acta 1982

Mathworks, Classification Using Nearest Neighbors, 08.01.2015, http://de.mathworks.com/help/stats/classification-using-nearest-neighbors.html

Nathalie Japkowicz, Shaju Stephen, *The Class Imbalance Problem: a Systematic Study*, Intelligent Data Analysis 6 2002

Xinjian Guo, Yilong Yin, Cailing Dong, Gongping Yang, Guangtong Zhou, *On the Class Imbalance Problem*, Fourth International Conference on Natural Computation 2008

Vincent Van Asch, *Macro- and micro-averaged evaluation measures*, 2013

http://de.mathworks.com/matlabcentral/fileexchange/46035-confusion-matrix—accuracy--precision--specificity--sensitivity--recall--f-score

http://de.mathworks.com/matlabcentral/fileexchange/21212-confusion-matrix---matching-matrix-along-with-precision--sensitivity--specificity-and-model-accuracy/content/cfmatrix/cfmatrix2.m

8. ANNEX

- Table of performance metrics for multiclass classification.
- Table of performance metrics for One-vs.-Rest classification.

Algorithm	kNN		RSLVQ		SVM	
Fair Balanced Accuracy Performance	0.6229		0.5759		0.7130	
Sensitivities	0.0437		0.3236		0.3541	
	0.2219		0.1857		0.3349	
	0.5126		0.5756		0.4736	
	0.3492		0.3545		0.5218	
	0.5813		0.3923		0.7342	
	0.6957		0.2514		0.6193	
	0.6411		0.0407		0.7713	
	0.7889		0.8550		0.9374	
	0.1684		0.1250		0.5726	
	0.1360		0.8740		0.6853	
	0.5737		0.3277		0.6597	
	0.3667		0.7583		0.4833	
	0.5904		0.4581		0.2455	
	0.7378		0.5298		0.8343	
	0.5000		0.4500		0.7500	
Specificities	0.9788		0.9029		0.9173	
	0.9935		0.9885		0.9814	
	0.9890		0.9763		0.9864	
	0.9947		0.9533		0.9855	
	0.9812		0.9797		0.9788	
	0.9160		0.9613		0.9759	
	0.9784		0.9796		0.9525	
	0.9945		0.9849		0.9949	
	0.9933		0.9856		0.9837	
	0.9950		0.7810		0.9743	
	0.9518		0.9585		0.9575	
	0.9957		0.9591		0.9969	
	0.9366		0.9120		0.9969	
	0.9871		0.9794		0.9814	
	0.9994		0.9981		0.9978	
Model parameters	k	7	Prototypes Per Class Count	1	Kernel	RBF
			Epsilon	10^{-6}	Epsilon	10^{-3}
			alpha	0.5	Cost	10
			Alpha Annealing Rate	0.9	Gamma	10^{-3}
			Sigma Annealing Rate	0.9		
			Pro Init	10^{-3}		

Table 1: Multiclass performances for the best models

Algorithm	**kNN**	**RSLVQ**	**SVM**
Fair Balanced Accuracy Performances	0.3691	0.6612	0.6683
	0.4523	0.8053	0.9250
	0.5398	0.8578	0.9232
	0.5460	0.7733	0.8465
	0.6809	0.8319	0.8574
	0.7400	0.7265	0.8513
	0.7779	0.6255	0.8388
	0.8021	0.9112	0.9638
	0.4756	0.7477	0.7929
	0.4377	0.7860	0.8638
	0.7464	0.7115	0.8168
	0.5873	0.8312	0.8426
	0.6927	0.7126	0.7630
	0.7868	0.8351	0.9061
	0.6818	0.8567	0.9916
Sensitivities	0.1087	0.7814	0.7575
	0.2257	0.7976	1.0000
	0.3585	0.9063	0.9417
	0.3493	0.7950	0.8616
	0.5494	0.8412	0.8258
	0.6378	0.8061	0.9119
	0.6875	0.5125	0.8624
	0.7202	0.9818	0.9646
	0.2585	0.7009	0.7717
	0.2051	1.0000	0.9273
	0.6496	0.7792	0.7711
	0.4167	0.8750	0.8333
	0.5679	0.7378	0.6739
	0.6988	0.8807	0.9138
	0.5500	0.8000	1.0000
Specificities	0.9718	0.5941	0.6585
	0.9826	0.8175	0.8940
	0.9835	0.8420	0.9301
	0.9864	0.7836	0.8335
	0.9779	0.8282	0.9118
	0.9443	0.7096	0.8269
	0.9729	0.7985	0.8247
	0.9967	0.8799	0.9786
	0.9841	0.8135	0.8396
	0.9869	0.6974	0.8508
	0.9373	0.6722	0.9035
	0.9878	0.8614	0.9124
	0.9535	0.7098	0.9270
	0.9904	0.8041	0.9053
	0.9984	0.9904	0.9881

Table 2: One-vs.-Rest performances for the best models