Machine Learning Assignment – COMP3008
Marinos Mavrommatis – mm1g10
15 May 2013

Scope

The purpose of this assignment was to apply Supervised Learning techniques on the Arrhythmia Data Set at [1]. This multivariate dataset has 452 instances and 279 attributes. Each instance belongs to one of 13 classes, the biggest of which is called "Normal", meaning that people in this class have no arrhythmia. Class Normal contains more than half of the instances. As a result, there are 207 instances spread across 12 classes (i.e. different types of arrhythmia) making the problem of learning each class very hard. For this reason, I chose to convert this multiclass classification problem into a binary one and attempted to learn how to detect arrhythmia patients in general, regardless of which type of arrhythmia they have.

In terms of learning machines, I chose to experiment with various Support Vector Machines (SVMs) implementations: Liblinear, Libsvm and Libsvm with limited Support Vectors. For this task I used the "Scikit" library for Machine Learning in Python [2]. In the rest of this report I describe the pre-processing techniques used, how the tuning of the various SVMs was performed and how the performance of each machine was measured.

Pre-processing

As is typical in machine learning, the dataset had some missing values. In particular, one of the attributes (attribute 14) had almost no data filled-in; that is, very few data instances had a value for that attribute. SVMs in Scikit can't handle missing values by default and coming up with sensible values for an almost empty column would be very difficult, so the column was removed entirely leaving the dataset with 278 attributes. Other attributes that had a few missing values were completed using the respective attribute's mean value.

Another characteristic of the data was that each attribute was in a different scale. That is, some attributes were in the range (-5.0, 5.0) while others were integers between 1 and 100. In an initial investigation, scaling all features to range (0.0, 1.0) seemed to increase the performance of the SVMs by around 10% on unseen data and so this scaling was adopted for the whole experiment.

Finally, I experimented with reducing the dimensionality of the (scaled) data using PCA. As shown in Graph 1 (Appendix 3), for the Liblinear implementation (LinearSVC in the chart's legend), keeping the 5, 10, 50 and 80 most significant dimensions gave better and better results respectively, with the best results coming from not using PCA at all.

For the next three tasks (training/parameter tuning, SVM implementation selection and generalisation performance evaluation), the dataset was split randomly into three parts of sizes 270, 91 and 91 respectively. The following three sections describe how these were used.

Tuning

Scikit provides convenient tools for tuning machine parameters. One of these is grid-search. Given a training dataset and a Python dictionary with the possible values for each parameter the program automatically iterates over all possible parameter combinations, each time training the machine and evaluating its performance using cross-validation. Values used for each parameter are shown in Listing 1 (Appendix 1).

A major limitation of grid search is that, while Scikit provides various implementations of SVMs, the choice of implementation cannot automatically be made like other parameters in the grid; instead, one grid search must be performed for each implementation. The "implementation selection" part of the dataset was used for this reason.

Implementation selection

There are three main SVM implementations provided by Scikit for data classification. The first is called LinearSVC and, as the name suggests, uses a linear kernel. It is implemented using the Liblinear library and it allows tuning the penalty parameter C and the tolerance for stopping criteria. Another implementation is SVC, which uses Libsvm and apart from C and stopping tolerance also allows choosing a kernel from: linear, polynomial, radial basis and sigmoid. A "degree" parameter can also be set but this only applies to the polynomial and sigmoid kernels according to the specification. Finally, NuSVC is another implementation based on Libsvm that provides control of the upper limit on the number of support vectors, using the "nu" parameter.

The parameters of each of the three machines were tuned separately using grid search as explained in the previous section. However, each of the machines was tuned five separate times: once on the original (but scaled) data, and once for each of the PCA reductions to 5, 10, 50 and 80 dimensions as previously explained. Each of the 15 trained and tuned machines was tested on the implementation selection set. Their accuracy (percentage of correctly classified instances) was compared to each other's as per Graph 1 and LinearSVC (Liblinear implementation) with parameters: C=1.0, tolerance=0.0001 and using the data without PCA performed better than all others, with a performance of 82% on the implementation selection set. An interesting fact is that is not shown in Graph 1 is that Libsvm with a linear kernel only scored 78% on the same dataset (again without PCA).

Apart from measuring the score of each SVM as a percentage of correctly classified instances, an ROC curve was created for each machine's predictions on the implementation selection set. The area under the graph was calculated each time giving values proportional to the accuracy score. These are listed in Appendix 2 (Listing 2).

Generalisation performance

The best performing machine (LinearSVC) was finally tested on the (unseen) final evaluation part of the dataset and classified 78% of the instances correctly giving an ROC curve with area equal to 0.79.

References (Dataset + scikit)

1. UCI Machine Learning Repository, Accessed 15 May 2013, [http://archive.ics.uci.edu/ml/datasets/Arrhythmia]
2. Scikit-learn, Accessed 15 May 2013, [http://scikit-learn.org/stable/]

Appendix 1:

Degree of kernels (not applicable to Linear kernel)	1, 2, 3, 4, 5
Tolerance for stopping criteria	0.0001, 0.001
Penalty parameter C	2^-9, 2^-8, … 2^8, 2^9
Limit support vectors (“nu”)	0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8

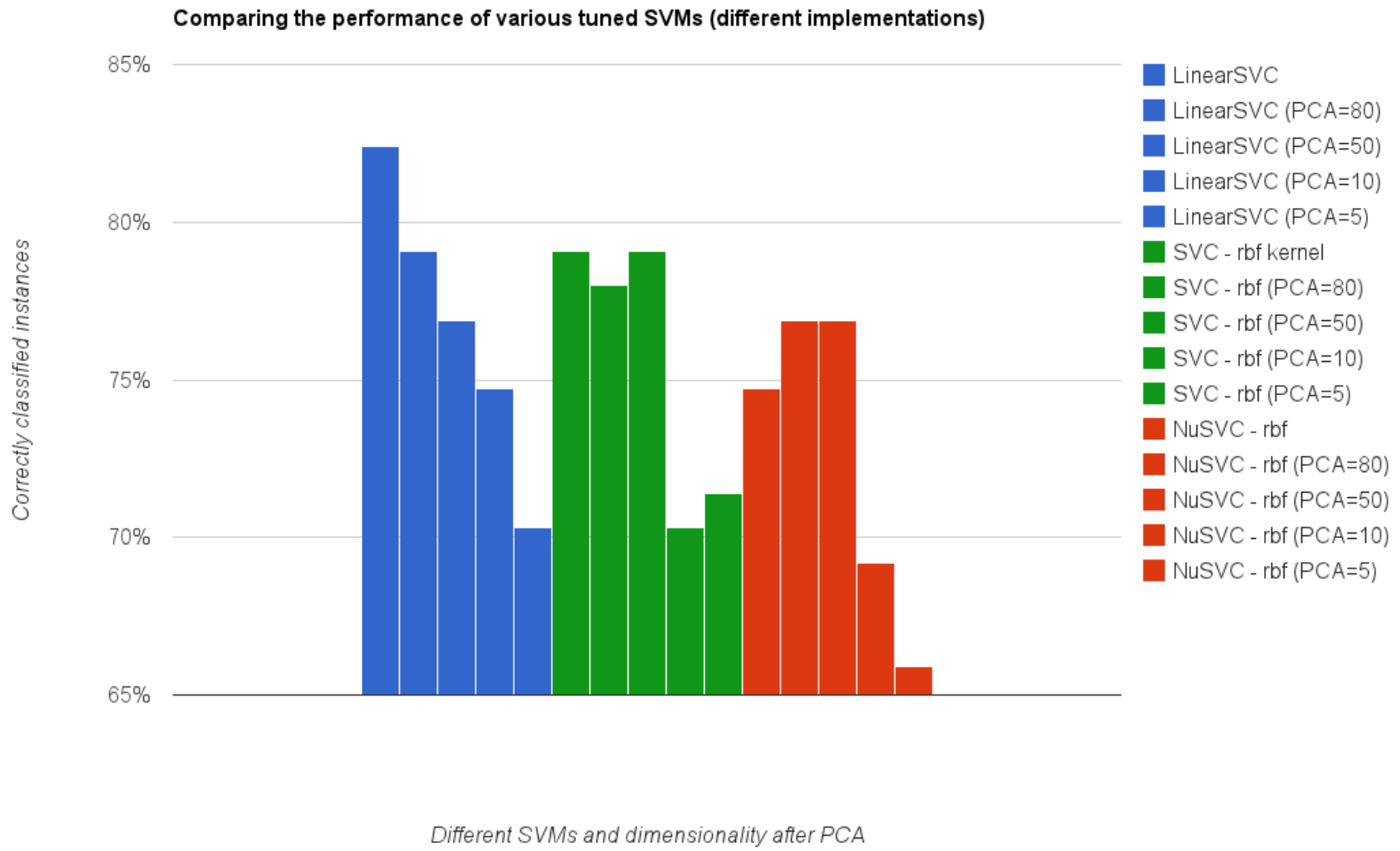
Listing 1: Values for parameters that were tuned

Appendix 2:

LinearSVC	0.808
LinearSVC (PCA=80)	0.776
LinearSVC (PCA=50)	0.75
LinearSVC (PCA=10)	0.734
LinearSVC (PCA=5)	0.67
SVC - rbf kernel	0.776
SVC - rbf (PCA=80)	0.763
SVC - rbf (PCA=50)	0.769
SVC - rbf (PCA=10)	0.667
SVC - rbf (PCA=5)	0.702
NuSVC - rbf	0.727
NuSVC - rbf (PCA=80)	0.747
NuSVC - rbf (PCA=50)	0.744
NuSVC - rbf (PCA=10)	0.683
NuSVC - rbf (PCA=5)	0.647

Listing 2: Area under ROC curve for each machine (and for each dimensionality after PCA)

Appendix 3: Performance of each machine on unseen data



Graph 1: Performance of each machine on unseen data