Machine Learning Winter 2014 – Exercise 2

The third exercise deals with advanced experiments on particular topics in machine learning. The exercise shall be done in groups of 2 students, which may be the same as in the first two exercises. Please arrange with your colleagues, and then apply to the same group in TUWEL.

You will have to present your findings in a presentation for all the course participants in January. You will have to be present during the whole duration of the time slot you will be presenting; participation in the other slot is optional.

You have to select your topic by applying to one of the groups - the topic will be indicated in the group name. All group members should apply to the same group - so please, do not apply to a group where already a student from a different group has applied to... If you have an interesting idea of any other topic, please don't hesitate to contact us and discuss it with us. If we see it fitting to the topics of the course, you'll get our ok to work on your idea.

General comments for the exercise

This exercise is supposed to be done using the WEKA API. The main reason for this is that WEKA provides a vast number of different learning algorithms, and additional features such as a defined format for writing & reading results, for performing significance tests, etc.

If you, for whatever reason, want to implement your solution in a different environment, then please contact us (mayer@ifs.tuwien.ac.at & musliu@dbai.tuwien.ac.at), and explain why the exercise can be performed in the same quality in the different environment.

Your submission should contain

- a zip file with all needed files (your source code, your code compiled, data sets used, the WEKA libraries and any other libraries you are using)
  - Provide a means to compile your classes, preferably with an ANT build file, and a short how-to explaining the way to start your program (which is the main class, which command-line options does it expect, ..)
- a report, describing
  - Your task
  - Your solutions (also describe failed approaches!)
  - Your results, evaluated on different datasets, parameters, ...
  - An analysis of the results
- Where applicably, your program shall be configurable via command-line options or a configuration file, to modify parameters, evaluation types, etc...
  - Aka - it should not be needed to modify the code to change these options

If using WEKA, please use the most recent version from http://www.cs.waikato.ac.nz/ml/weka/ !

A useful framework for command-line-option is Apache commons CLI (http://commons.apache.org/cli/), which takes most of the burden to parse the supplied arguments. JSAP (http://martiansoftware.com/jsap/) might be another solution.

The descriptions of the different topics are not to be read as complete specifications, they just summarise the general idea and goals. If there are certain unclarities, please do not hesitate to ask in the forum.

The exercises can be grouped in two types:

1. The first one deals with large-scale evaluations, where you shall employ the WEKA API during your experiments, to allow for a more automated evaluation, and an evaluation beyond what you can obtain from the WEKA Explorer / Experimenter, resp. from manually calling each classification run.
2. The second type of exercises deals more with demonstrating specific properties of certain machine learning algorithms.

A. Large-Scale evaluation

In these exercises, you shall employ the WEKA API during your experiments, to allow for a more automated evaluation, and an evaluation beyond what you can obtain from the Explorer / Experimenter.

You shall use the WEKA API to persist your results in either WEKA ARFF files (these are not only used for input data, but also to store results), or a database. For this, you can utilise the WEKA implementations of ResultListener, such as InstancesResultListener or DatabaseResultListener.

As you shall do mass evaluation, you should generate tables and diagrams automatically from the WEKA result files, using the WEKA API for tables (e.g. using ResultMatrix and subclasses), and using any Java chart library available for diagrams.

You can re-use the data sets you used in your first exercise, as you should be familiar with them - which should help you in the evaluation. ***Mind however that you shall perform experiments with a large number of data sets, so you will need to choose additional data sets.***

1. Ensemble classifier learning & evaluation

Using algorithms available in WEKA, develop an ensemble classifier that uses different algorithms, and different parameters of these classifiers. Implement two different strategies for combining the individual predictions of the classifiers, namely majority voting and weighted majority voting.

For the weighted majority voting, use a part of your data as validation set to estimate the classifiers accuracy (e.g. on the class it predicts), and use this value subsequently as weight. You should do this accuracy estimation not by doing a single split on the training set, but by doing cross-validation on the training set to achieve a more sound estimation of the expected performance.

Your program shall allow the user either to specify which algorithms (and parameters) to use (via command-line, or via a configuration file), or have some heuristics on building a "good" ensemble of classifiers.

Afterwards, run a suite of experiments on different data sets and different classifier ensembles, and compare the results of the ensemble with the results of each individual classifier used, and also compare requirements in runtime, etc.. Analyse the results in regard of whether you can always find better results, in whether one combination strategy is dominant over the other, and whether there is a set of (nearly) optimal combination of classifiers.

2. Automatic feature selection & evaluation

WEKA provides a set of feature selection algorithms, both supervised and unsupervised ones. In this task, you should evaluate the effect of the different techniques, and different parameters to it. For this exercise, we will provide you with a set of data sets from the music classification domain, which (contrary to e.g. text classification) does not yet widely employ feature selection.

Develop a small framework that on a set of different data sets automatically applies different feature selection techniques. There are two different topics to choose:

a) Evaluate the number of selected features

Evaluate the effect of the desired number of features after the selection (or thresholds used in the selection technique to decide how many features are kept). Vary the number of features from very few to very high, and record and analyse the results; specifically, for each feature selection technique and each target number of features selected, record the the runtime and classification accuracies. Then analyse the chart of performance and runtime for each method on several dataset, and try to make an evaluation of a potentially optimal number (or range) of features to select. The user should be able to specify the different numbers of features, or a step-size; if none of these is provided, think of a good strategy yourself (e.g. smaller step-size for smaller number of features, then increasing, depending on the total number of features in the dataset).

b) Compare feature selection methods

Analyse the differences in the rankings between the different methods, i.e. analyse how each individual feature is ranked. I.e. investigate whether there is a trend in that a certain subset of features is selected by all (or most) techniques - do they tend to select a specific subset of features, or does that vary a lot? Is there a trend in supervised and unsupervised methods?

Further, investigate whether a combination of feature selection methods is more beneficial, i.e. by taking those features that get selected by most algorithms, or by combining the top N from each algorithm.

For both topics, your program shall allow the user either to specify which feature selection techniques (and parameters) to use (via command-line, or via a configuration file), or otherwise simply use all available techniques.

3. Automatic evaluation of optimal k for k-nearest neighbour

The number of neighbours k to select in k-NN is depending on the dataset, and not easy to determine. Further, there is no clear idea on which distance function is best to use. In this exercise, you shall thus evaluate the performance of k-NN on different datasets when you vary the value of k by the means of a program that automatically increases the number of k, until further increases don't seem feasible anymore (e.g. the performance is continuously falling for a period, or fall below a percentage of the beast performance achieved).

Further, also compare these results when using different distance metrics, e.g. a Manhattan distance, or edit distance instead of the standard euclidean distance.

Your program should automatically record the statistics achieved for each value of k. Perform experiments on a multitude of datasets, and evaluate whether there is a trend for an optimal number of k, and for an optimal distance measure.

4. Automatic comparison of k-nearest neighbour search optimisation & evaluation

k-nearest neighbours, even though a very simple approach, yields good results in many classification tasks. One major issues is however the classification time on larger data sets, mainly stemming from the expensive (pair-wise) comparisons needed to find the nearest neighbours. In this exercise, you shall thus evaluate different optimisation strategies for this neighbour search.

A set of such strategies is in implemented in WEKA in combination with the IBk and LWL classifiers. Develop an evaluation framework that on different data sets automatically evaluates the effect of applying these strategies on the runtime and performance of the algorithms, and compares the different strategies against each other, and against the "base-line" of using the classic linear (brute-force) search. Record how long it takes to build the search optimisation structures, and how the optimisation changes the time needed for the classification process, and how the overall time needed compares to the base line. Try to estimate what the "break-even" point of time consumption is in regard to the size of the dataset and the dimensionality, i.e. at which combination of size and dimensionality building the search optimisation structure is equal in overall runtime than the linear search.

Further evaluate in detail the effects on the search optimisation strategies on the accuracy of the algorithm - does that differ? Also, even if the accuracy stays the same, are there data samples that get classified differently by the different strategies? Is there a trend that the optimisation strategies, when they predict a different class than the "base-line", all classify the pattern in the same (other) class, or is the prediction different for all strategies?

Subsequently, try to analyse which strategies lead to a similar nearest-neighbour selection.

Your program shall allow the user either to specify which search optimisation strategies (and parameters) to use (via command-line, or via a configuration file), or otherwise simply use all available strategies

5. Model Selection

Implement a framework that allows for model selection, i.e. to select the best performing model from a given set of models. The user shall be able to specify which algorithms (and which parameters for these) he would like to use. If no models are provided, use a pre-defined list of algorithms (with different parameters).

You shall do classical model selection, i.e. training the models on one set, selecting the best model on another set, and finally testing the generalisation powers on a third set not previously used.

Evaluate how well the model selection performs - is the model selected always the one having the best generalisation power? How different is the generalisation performance from the performance on the set you used for model selection? Is there a trend in which classifiers are more stable, i.e. have similar performance on the set used for selection and on the one for the final estimation, and which ones tend to fluctuate more?

Also, vary the size of model selection and generalisation estimation sets, to analyse what a reasonable size for each of those sets is. Perform you evaluation on a number of different datasets, with different characteristics.

6. Evaluation of large-scale learning tools

Working with a large data set (some for music classification into genres can be provided) you should evaluate the performance of various toolkits especially targeted to large-scale experiment. Specifically, you shall compare the runtime performance and results of the classifiers provided by at least the following frameworks

- Apache Mahout
- CERN Root
- Machine Learning Library (MLlib)

7. Large-scale evaluation with GPU supported toolkits

In this task, you shall conduct a survey on available machine learning toolkits that utilise GPUs, and evaluate and compare their performance on large data sets (some for music classification into genres can be provided). Specifically interesting is the runtime compared to normal CPU computation. Another interesting aspect is the repeatability of the experiments - as GPUs provide only floating point precision, is there any noticeable effect observed when re-running the very same experiments for several times?

You shall primary work with your own GPU during the initial phase of the task, but we also have a server with dedicated GPU blades running Ubuntu Linux on which you should finally run your experiments on.

B. Demonstration of algorithms

In this set of exercises, you shall demonstrate how certain algorithms are trained, and how they form decision boundaries.

The task involves creating/generating or finding datasets that demonstrate particular challenges and properties of an algorithm. In most cases, these can be 2 and 3 dimensional data sets, as they are easy to visualise.

The task further involves implementing a GUI that can be used to visualise the data, and also visualise how decisions and decision boundaries are formed. In most cases, you should be able to use the WEKA API to do the learning of the algorithms.

To this end, you should provide a visualisation of the data points in the 2D/3D space, where classes are indicated by colour coding. You can reuse any kind of library that helps you with plotting the data sets, potentially you can make use of the plots provided by WEKA itself.

8. Perceptron and Linear SVMs

Reuse and adapt/develop an implementation of (standard) Perceptrons for WEKA or another ML toolkit. Then, pick a number of datasets that provide examples of simple linear separation.

Provide an interface that visualises the data set and its classes. In that interface, provide controls where the user can select the parameters for Perceptron and SVM learning (you can reuse the e.g. controls WEKA provides for the classifiers), and visualise how each of these classifiers separates the space, also indicating margin boundary, and for SVMs also the support vectors.

Then pick a number of datasets that provide almost linear separation, and compare how the SVM can find a solution with a soft margin, while the Perceptron will not converge to a stable solution.

9. Ada Boost

Reuse and adapt/develop an implementation of (standard) Perceptrons for WEKA or another ML toolkit. Then, pick a number of datasets that would be easily separable by polynomial functions.

Provide an interface that visualises the data set and its classes. In that interface, provide controls where the user can select the parameters for Ada Boost learning, using a Perceptron (you can reuse e.g. the controls WEKA provides for the classifiers).

Then, provide an interactive mode where you visualise the decision boundary and a performance measure (e.g. accuracy) after each step in the boosting algorithm.