AUC and ensemble methods with PYTHON

Objective: The objective of today's exercise is to understand (i) how the ensemble methods bagging and boosting is used to improve the performance of classifiers, (ii) how the receiver operating characteristic (ROC) is used to evaluate the performance of two-class classification problems, and (iii) how artificial neural network and logistic regression can be generalized to multi-class classification.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* C14, C15 as well as the files in the exercise 9 folder available from Campusnet.

Preparation: Exercises 1-8

Part 1: Group discussion (max 15 min)

For the group discussion, each group should have selected a *discussion leader* at the previous exercise session. The purpose of the discussion leader is to ensure all team members understands the answers to the following two questions:

Multiple-Choice question: Solve and discuss **problem 14.1** from chapter 14 of the lecture notes. Ensure all group members understand the reason why one of the options is true and why the other options can be ruled out. (After today's exercises make sure to complete the remaining multiple-choice problems listed as part of the preparation for week 9 on the course homepage).

Discussion question: Discuss the following question in the group

- Consider the group exercise on slide 27. Suppose there are 20 positive and 50 negative objects in the training set and suppose classifier 1 and classifier 2 is obtained by varying a threshold parameter. Which two points on an AUC curve would classifier 1 and 2 correspond to?

Part 2: Programming exercises

Piazza discussion forum: You can get help by asking questions on Piazza: https://piazza.com/dtu.dk/fall2017/02450

Software installation: Extract the Python toolbox from Campusnet. Start Spyder and add the toolbox directory (`<base-dir>/02450Toolbox_Python/Tools/`) to `PYTHONPATH` (Tools/PYTHONPATH manager in Spyder). Remember the purpose of the exercises is not to re-write the code from scratch but to work with the scripts provided in the directory `<base-dir>/02450Toolbox_Python/Scripts/`

Representation of data in Python:

	Python var.	Type	Size	Description
	`X`	numpy.array	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	list	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	integer	Scalar	Number of data objects.
	`M`	integer	Scalar	Number of attributes.
Regression	`y`	numpy.array	$N \times 1$	Dependent variable (output): For each data object, `y` contains an output value that we wish to predict.
Classification	`y`	numpy.array	$N \times 1$	Class index: For each data object, `y` contains a class index, $\texttt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
	`classNames`	list	$C \times 1$	Class names: Name (string) for each of the C classes.
	`C`	integer	Scalar	Number of classes.
Cross-validation				All variables mentioned above appended with `_train` or `_test` represent the corresponding variable for the training or test set.
	`⋆_train`	—	—	Training data.
	`⋆_test`	—	—	Test data.

9.1 The receiver operating characteristic (ROC)

The receiver operating characteristic (ROC) is commonly used to evaluate and compare the performance of two-class classifiers, where one class is denoted "positive" and the other is denoted "negative." The ROC is a graphical approach for displaying the tradeoff between the true positive rate (y-axis) and the false positive rate (x-axis). For classifiers such as logistic regression and artificial neural networks that estimate the class labels by thresholding a continuous output variable, an ROC curve can be plotted by varying the threshold value.

9.1.1 Inspect and run the script `ex9_1_1.py`. The script loads the wine data (`Data/wine2`) into Python with the `loadmat` function. Notice how the data is divide the data set into 50% for training and 50% for test using stratification such that training and test have roughly equal class proportions. Fit a logistic regression model to the training set to classifiy the wines as red or white. Consider the red wines as "positive" and white wines as "negative." Notice how the script makes a plot of the ROC curve showing that the AUC is around 0.99.

Script details:

- *You have fitted a logistic regression model to the wine data before in exercise 5.2.6.*
- *As before, use cross-validation. Use* `StratifiedKFold` *method to ensure that training and test sets have roughly equal class proportions (stratification).*
- *There is a function in the course toolbox called* `rocplot()` *that can make the ROC plot and compute the AUC score for you. Import it and type* `help(rocplot)` *to learn how to use it. Notice that the rocplot code assumes the score values are all distinct.*

9.1.2 Consult the script `ex9_1_2.py`. The experiment in exercise 9.1.1 is repeated, but this time it is examined how well the type of wine can be classified using only the "Alcohol" attribute. Explain how it can be seen that the alcohol contents of the wine is not very useful for classifying wine as red or white.

Discussion:

- ◇ You have showed that using only the single attribute "Alcohol" to classify the wine as red or white performs worse than using all attributes. When using logistic regression, is it always best to use as many attributes as possible for the classification?

9.2 Ensemble methods

In this part of the exercise we will consider ensemble methods to improve the classification performance. In particular we will consider bagging and boosting methods.

In bagging we randomly sample with replacement the same number of samples as the size of the training data. This is also denoted bootstrapping and it can be shown that on average each bootstrap sample will contain approximately 63% of the samples in the data.

In boosting we adaptively change the distribution that we sample the training examples from so that the classifiers will focus on examples that are hard to classify. A particularly well known boosting method is *"AdaBoost"* which is described in section 15.4 of the lecture notes *Introduction to Machine Learning and Data Mining*.

In this part of the exercise we will work with a synthetic data set which has two classes and two attributes, x_1 and x_2. This data set cannot be classified correctly

with any linear classifier, such as logistic regression or an artificial neural network with one hidden unit, as long as only x_1 and x_2 are used as features.

9.2.1 Load the artificial dataset (`Data/synth5` file) with `loadmat` function. Inspect the data by making a scatterplot. Why can this data set not be classified correctly using logistic regression?

Inspect and run the script `ex9_2_1.py`. The script fits an ensemble of logistic regression models to the data, using the bootstrap aggregation (bagging) algorithm. Use $L = 100$ bootstrap samples. This requires creating 100 bootstrapped training sets, fit a logistic regression model to each, and combine the results of their outputs to make the final classification. Explain how the error rate is computed (on the training set) and how the decision boundary is plotted.

Script details:

- *To generate the bootstrap sample, you need to draw N data objects with replacement from the data set.*
- *You can use the function* `bootstrap()` *function from the toolbox to generate random numbers from a discrete distribution. You can write something like* `X_bs, y_bs = bootstrap(X, y, N, weights)` *to make a bootstrap sample.*
- *To make the final classification, take a majority vote amongst the classifiers in the ensemble. You can use simple class* `BinClassifierEnsemble` *from the toolbox.*
- *To plot the decision boundary, you can use the function* `dbplot()` *or* `dbprobplot()` *from the toolbox. It requires as an input an object that implements* `predict(X)` *and* `predict_proba(X)` *methods. You can call it e.g. like this:* `dbprobplot(fitted_classifier, X, y, 'auto', resolution=200)`

Bagging is known to be most effective for non-linear classifiers. Show that bagging only leads to a limited performance improvement for the logistic regression classifier.

Try also the data set in `Data/synth6` which is the] one used as an example in the lecture.

9.2.2 In the script `ex9_2_2.py`, the script `ex9_2_1.py` has been modified so that boosting is used instead of bagging. The script fits an ensemble of logistic regression models to the data, using the AdaBoost algorithm. Notice the script uses $L = 100$ rounds of boosting. This requires creating a randomly chosen training set, fit a logistic regression model to it, evaluate its performance and update the weights accordingly, and compute a classifier importance. This process is repeated $L = 100$ times, and ultimately the trained ensemble of classifiers is combined to make a final classification. Compute the error rate (on the training set) and make a plot of the decision boundary.

Script details:

- *Read the hints to the previous exercise.*

- *Note that you will need two sets of weights in the algorithm. One is a weight for each of the N data objects, that is adapted when the boosting algorithm proceeds (we could call these* `weights`*). The other is the importance weights for the L trained classifiers (we could call these* `alpha`*).*
- *You can use the function* `bootstrap()` *to generate random numbers from a discrete distribution, as before. You will need to update the 'weights' parameter in according to AdaBoost algorithm, in every iteration.*
- *To make the final classification, take a weighted majority vote amongst the classifiers in the ensemble (weighted by* `alpha`*). Note that* `alpha` *needs to be normalized so that it sums to one. ii*

Show that, if you use enough rounds of boosting, the data set can be perfectly classified.

Try also the data set in `Data/synth6` which is the one used as an example in the lecture.

Let us return to the bagging algorithm, which we found to be of little use for logistic regression. Now, we will try the algorithm on a non-linear classifier: the decision tree. Bagging applied to decision trees is often called "random forests". The Python's package `sklearn.ensemble` has implemented bagging for random trees, see the class `RandomForestClassifier()`.

9.2.3 The data set `Data/synth5` we have used so far can trivially be fitted using a decision tree. Can you explain why? We will consider a different data set, which you can load from the file `Data/synth7`. Inspect and run the script `ex9_2_3.py`. Explore the data set and explain in what sense this is more challenging for a decision tree.

Notice the script fits a random forest (an ensemble of bagged decision trees) to the data using $L = 100$ rounds of bagging. Compute the error rate (on the training set) and make a plot of the decision boundary.

Script details:

- *Type* `help(sklearn.ensemble)` *to learn about the functions for fitting random forests with bagging.*

For comparison, you can try also to fit a regular decision tree (without bagging or pruning). Does bagging appear to improve on the classification, and if so, in what sense? Observe how the classification rate and decision boundaries decrease when you reduce number of bootstrap iterations.

Try the script on the other synthetic multi-class data sets you have studied before, (`Data/synth5` ... `Data/synth7`).

9.3 Tasks for the report

Go over the past weeks exercises and get an overview of what you are missing for report 2.

References