1 Introduction

For this assignment I've used three classification methods on the same dataset in order to compare their results and to evaluate their performances. For two of them I have also estimated the best parameters to fit my data and I used the best result among the possible parameters for the comparison with the other methods.

The three classification methods that we are going to compare are Support Vector Machine classifier, Naive Bayes classifier and Random Forest classifier.

2 Tools

I have implemented this assignment using Python 2.7.6 that is an high level programming language. Moreover this tool provide a library named scikit-learn[1] that contains a lot of useful API to perform machine learning task. In my case, I used this external library for the implementation of the K-fold algorithm, for the classifiers and for the evaluation of each of them. This tool is quite powerful and can easily be integrated within each software produced with Python.

Another useful library that I have imported into my project is called Numpy[2] that is a collection of really optimized mathematical operations. So in order to get the average value of every array that I built I used this tool.

3 Software structure

The goal of this assignment is evaluate and compare the results of three different classification methods on a random binary classification problem.

So first of all we need to build our dataset, and for this we used a feature provided by scikit that creates a random classifier based on some parameters. In my case I specified that my dataset is going to have 1000 samples, with 10 feature each and obviously that it will be a binary classifier (meaning that we are going to have only two classes).

After that we can use K-fold cross validation in order to test our classifiers. So I used another feature provided by scikit-learn for the partition of my data set into 10 distinct subsample. Then, for each k of the K partitions, I have used the K-k remaining samples to train my classifiers and the k-sample as validation. This operations is repeated k times and so each observation is used for validation exactly once.

So for each of the ten folds that I have created I trained and validated my three classifiers. Then for each of them I have calculated accuracy, F1 and AucRec value.

4 Classifiers

For this assignment I considered three classifiers: Support Vector Machine classifier, Naive Bayes classifier and Random Forest classifier. Each of them is based on a different idea and has obviously different characteristics. Let's see some details:

4.1 Support Vector Machine

A Support Vector Machine (SVM) is an algorithm that construct an hyperplane which can be used for classification and other tasks (e.g. regression). Intuitively, the best hyperplane that we can use is the one that has the largest distance to the nearest training-sample point of any class (named functional margin). In general the larger the margin the lower the error of the classifier.

Moreover in the assignment for this classifier we had a list of possible "C" values. C is one of the parameters required by the scikit-learn implementation of the SVM classifier. This parameter is also known as "penalty parameter" and is used to tell the SVM how much you want to avoid misclassifying each training example. For large values of C, the classifier will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

So in order to decide the best C value I tried a 5-fold cross validation for each of the possible values of C. After that I have choose the C value that produce the highest average value of F1 over the 5 iterations.

4.2 Naive Bayes

The Naive Bayes classifier is conceptual easier then the other two classifiers: it evaluate the "prior-probability" of each label :

$$prior - probability\ of\ labelA = \frac{num\ of\ labelA}{total\ number\ of\ labelA\ cases}$$

After that it easily can classify every new sample: the algorithm place the new sample into the system with all the other samples, then it gatherer the k most near points irrespective of their class labels (k chosen a priori). Once we have these points we can evaluate the "posterior-probability". The likelihood of the new sample given LabelA is calculated as

$$likelihood\ of\ X\ given\ labelA = \frac{num\ of\ labelA\ in\ the\ vicinity\ of\ X}{total\ number\ of\ labelA\ cases}$$

And then, with those parameters we can evaluate the posteriori-probability of X that is used by the classifier in order to give a label to a new sample:

$$Posterior\ probability\ of\ X\ being\ labelA = prior\ probability\ of\ labelA \times Likelihood\ of\ X\ given\ labelA$$

So this value is exactly our label: as our assignment is a binary classifier we know that if this value is greater then 50% we are going to assign this sample to the labelA side, otherwise it's going to the labelB.

4.3 Random Forest

The last classifier is the Random Forest classifier that use an ensemble approach. Ensemble is a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of weak learners can come together to form a strong learner.

Our method starts with a standard machine learning technique called a decision tree which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it

traverses down the tree the data gets bucketed into smaller and smaller sets. So the random forest combine a lot of decision tree in order to get a strong classifier by averaging the results of many weak classifiers.

For this method we have choose among a set of possible values the best for each K-Fold iteration: in details for each K-Fold iteration we have tried and validated with a 5-fold cross validation the best n_estimators parameter. This argument of the function is responsible for the number of decision tree in the forest.

5 Evaluation

For the evaluation I considered three measures: for each K-Fold iteration and for each method I have collected accuracy, F1 and Auc Roc values.

The accuracy is defined as a statistical measure of how well a binary classification test correctly identifies or excludes a condition and is calculated by:

$$Accuracy = \frac{num\,of\,true\,positives\ +\ num\,of\,true\,negatives}{total\,number\,of\,classification}$$

The F score can be interpreted as a weighted average of the precision and recall (remark: precision is the number of correct positive results divided by the number of all positive results, and the recall is the number of correct positive results divided by the number of positive results that should have been returned).

$$F1 = 2 \times \frac{Precision \times Recall}{Precision\ +\ Recall}$$

Last evaluation criteria is Auc Roc (Area Under Curve of Receiver Operating Characteristic). The Roc is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. In particular the Auc test that we use consider the area under the curve as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative').

6 Results

So with all those statistical information that I have I can try to give some results on the methods performances. I tried to run my program a lot of times and the results that we can see in Figure 1 are a good representation of most of them. From what we can see in the results among the methods the best average performances in each measure are achieved by the Random Forest algorithm, followed by the SVM and the last in my classification is the Naive Bayes method.

However if we take a deeper look into the folds we can notice that the SVM classifier overtake the Random Forest in some run. In fact in a few runs of my software the SVM did also overtake the Random Forest on the average values. Yet in most of my tests the third method is the best one. Surely every time the Naive Bayes is the worst ad cannot compete with the others.

So in conclusion for my test the best classifier algorithm for a binary classification problem is the Random Forest but also SVM can be used as its performances are quite similar. Instead we can state that the Naive Bayes classifier has minor performances on this kind of problem.

```
Accuracy:
Fold    SVM             Naive           RndFrst
-------------------------------------------------
0       0.960000        0.930000        0.950000
1       0.880000        0.860000        0.920000
2       0.940000        0.870000        0.960000
3       0.900000        0.870000        0.940000
4       0.910000        0.880000        0.930000
5       0.850000        0.800000        0.880000
6       0.920000        0.900000        0.970000
7       0.870000        0.800000        0.870000
8       0.940000        0.900000        0.940000
9       0.850000        0.830000        0.900000
-------------------------------------------------
        0.902000        0.864000        0.926000
F1:
Fold    SVM             Naive           RndFrst
-------------------------------------------------
0       0.960000        0.929293        0.948454
1       0.888889        0.870370        0.924528
2       0.943396        0.876190        0.961538
3       0.915254        0.894309        0.948276
4       0.903226        0.872340        0.923077
5       0.851485        0.818182        0.880000
6       0.913043        0.888889        0.966292
7       0.863158        0.787234        0.860215
8       0.942308        0.907407        0.943396
9       0.823529        0.804598        0.886364
-------------------------------------------------
        0.900429        0.864881        0.924214
AucRoc:
Fold    SVM             Naive           RndFrst
-------------------------------------------------
0       0.960384        0.930172        0.949780
1       0.880808        0.860606        0.923232
2       0.941224        0.871578        0.962963
3       0.896652        0.856346        0.941711
4       0.912121        0.882828        0.930303
5       0.850000        0.800000        0.880000
6       0.921212        0.898990        0.968687
7       0.870132        0.799277        0.868928
8       0.942834        0.899356        0.941224
9       0.845573        0.828029        0.900857
-------------------------------------------------
        0.902094        0.862718        0.926769
--- 195.417955875 seconds ---
```

Figure 1: Report of my software in console

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Travis E. Oliphant. Numpy: Python for scientific computing, 2007–.