COMS 573

LAB ASSIGNMENT 3: LAB REPORT

HOMAGNI SAHA

TASK 1:

For the adaboost technique, we used the adaboost.SAMME algorithm in the scikit learn library of python as M1 algorithm was not available. We implemented random forests and adaboosting using decision stumps as asked in the question. For decision stump we had to fix depth =1 and number of leaves=2 for decision trees. We experimented with the learning rate for adaboosting and the number of estimators used for both adaboosting and random forest. Cross validation is used to calculate confusion matrices.

Adaboosting:

Parameters used

For adaboosting we fixed learning rate at 0.1 and maximum number of estimators at 400 (best result reported) and 10000. (additional exploration)

Results:

The following are the results taken directly from the program output in the terminal

testing ada\_discrete performance

The mean training accuracy on ada\_discrete 0.684958784288

The mean testing accuracy on ada\_discrete 0.730792079208

Printing the confusion matrix for ada\_discrete

[[238 0]

[ 63 0]]

On varying the maximum estimators to 10000, the following confusion matrix was obtained. (also see fig 1 for the corresponding error rate curve)

[[226 12]

[ 43 20]]

Random Forests:

Parameters used

For random forests, we maintained number of estimators at 400, the maximum depth at 1, the minimum samples split at 2 and minimum samples leaf at 1. (In order to use decision stumps)

Results:

The following are the results taken directly from the program output in the terminal

testing random\_forest performance

The mean training accuracy on random\_forest 0.696069895399

The mean testing accuracy on random\_forest 0.790693069307

Printing the confusion matrix for random\_forest

[[238 0]

[ 63 0]]

printing relative importance of each feature using random forests

[ 0.4825 0.2575 0.23 0.03 ]

Discussion:

For adaboosting the graph of error rate was seen to saturate early when number of estimators were around 100, it started fluctuating again around 2000 and then remained approximately flat. It might indicate the choice of using decision stumps as base estimators may not be the best choice for adaboosting. Intuitively, if decision stumps are used rather than decision trees we are forcing the model to look only at the high-level features. Interestingly the testing accuracy was seen to be higher than the training accuracy for both the methods which indicates that a greater amount of data might have been needed for cross-validation to work nicely. Both the algorithms show high true positives.

Figure 1 below shows how the adaboosting is varying for the dataset based on the number of estimators.

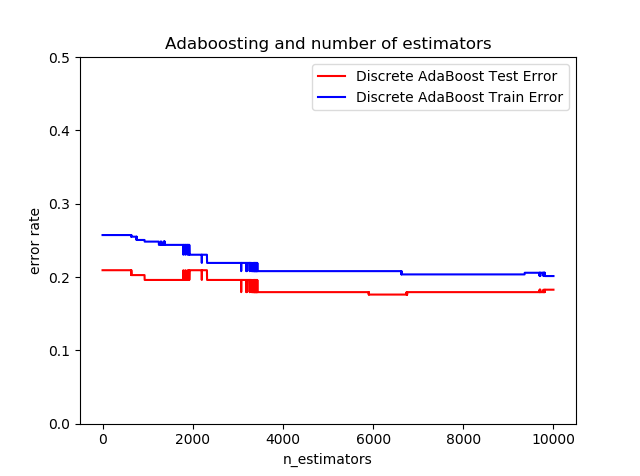


Figure 1: Adaboosting

TASK 2:

In this task we used the 5 mentioned models on the dataset.

Neural Networks:

Parameters used:

There are lots of parameters to experiment with that are provided in the scikit learn library. We experimented with learning rate, the solver and the activation. For the remaining parameters the defaults offered by scikit-learn were used.

Learning rate=0.01, solver= adam, activation=logistic, hidden layer sizes=(150,100,50,) (This means the second layer just after the input has 150 neurons, third 100, and fourth 50, followed by the output layer)

Results:

testing Neural networks performance

The mean training accuracy on Neural networks 0.742736562065

The mean testing accuracy on Neural networks 0.790693069307

Printing the confusion matrix for Neural networks

[[238 0]

[ 63 0]]

The results are comparable with the results obtained in [[1]](http://neuroph.sourceforge.net/tutorials/BloodTransfusionSc/blood_transfusion_sc.html), where a java toolkit for neural networks was used and extensive testing was done based on multi-layer perceptron models.

K nearest neighbors:

Again, the functions provided by sci-kit learn was used to construct the model and train it on the data.

Parameters:

We experimented on the number of neighbors(n\_neighbors), the weighting (use distance weighting or use uniform weighting), the distance metric, and the distance metric factor(p)

n\_neighbors=200, weighting =distance, metric=minkowski, p=2, leaf\_size=10( only affects speed of query)

Results:

testing k-Nearest Neighbor performance

The mean training accuracy on k-Nearest Neighbor 0.58026079771

The mean testing accuracy on k-Nearest Neighbor 0.700825082508

Printing the confusion matrix for k-Nearest Neighbor

[[228 10]

[ 53 10]]

Logistic regression:

Parameters used:

Penalty metric= l1 norm, inverse of regularization strength (C) =0.1 (strong regularization), class weights=balanced, maximum\_iterations (for a liblinear solver)= 200. Other parameters were the default ones provided by scikit learn.

Results:

testing Logistic regression performance

The mean training accuracy on Logistic regression 0.591655077897

The mean testing accuracy on Logistic regression 0.727722772277

Printing the confusion matrix for Logistic regression

[[192 46]

[ 20 43]]

Naïve Bayes:

Parameters used:

The sci-kit learn Bernoulli Naïve Bayes technique was used.

alpha : float, optional (default=1.0, used 0.8 in the experiment) Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

binarize : float or None, optional (default=0.0, used 0.02 in the experiment) Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

fit\_prior : boolean, optional (default=True, used False in the experiment and got huge improvement) Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

Results:

testing Naive Bayes performance

The mean training accuracy on Naive Bayes 0.738292117621

The mean testing accuracy on Naive Bayes 0.79399339934

Printing the confusion matrix for Naive Bayes

[[238 0]

[ 62 1]]

Decision Tree:

Parameters:

criterion : string, optional (default=”gini”, used “gini” and “entropy” in the experiment, reported for the best) The function to measure the quality of a split.

splitter : string, optional (default=”best”, used default as well as random splitting ) The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max\_depth : int or None, optional (default=None, used =4 in experiment ) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

min\_samples\_split : int, float, optional (default=2, used 100 in experiment ) The minimum number of samples required to split an internal node: If int, then consider min\_samples\_split as the minimum number. If float, then min\_samples\_split is a percentage and ceil(min\_samples\_split \* n\_samples) are the minimum number.

min\_samples\_leaf : int, float, optional (default=1, used 50 in experiment) The minimum number of

max\_features : int, float, string or None, optional (default=None, used 4 in experiment ) The number of features to consider when looking for the best split.

All other parameters are used as set in default by scikit learn.

Results:

(for splitter=best)

testing Decision trees performance

The mean training accuracy on Decision trees 0.644958784288

The mean testing accuracy on Decision trees 0.790693069307

Printing the confusion matrix for Decision trees

[[223 15]

[ 39 24]]

A slightly different result was obtained when using splitter=random

testing Decision trees performance

The mean training accuracy on Decision trees 0.742736562065

The mean testing accuracy on Decision trees 0.790693069307

Printing the confusion matrix for Decision trees

[[238 0]

[ 63 0]]

Ensemble unweighted majority vote classifier:

Parameters:

The weights for each classifier were the same

Results:

testing voting\_classifier performance

The mean training accuracy on voting\_classifier 0.740514339843

The mean testing accuracy on voting\_classifier 0.790693069307

Printing the confusion matrix for voting\_classifier

[[238 0]

[ 63 0]]

Ensemble unweighted majority vote classifier:

Parameters :

The relative weightage given to each classifier was based on their respective best testing accuracies. The weights vector was chosen to be (read from left to right corresponding to Neural Network (NN), KNN, Logistic Regression (LR), Naive Bayes (NB) and Decision Tree (DT))

W= [0.79,0.70,0.72,0.79,0.79]

The weights are directly the float values (up to first 2 digits after 0) for testing accuracy of each of the classifier

Results:

testing voting\_classifier performance

The mean training accuracy on voting\_classifier 0.740514339843

The mean testing accuracy on voting\_classifier 0.790693069307

Printing the confusion matrix for voting\_classifier

[[238 0]

[ 63 0]]

On trying different random seeds a slightly different result (not the best) was also observed.

testing voting\_classifier performance

The mean training accuracy on voting\_classifier 0.624958784288

The mean testing accuracy on voting\_classifier 0.790693069307

Printing the confusion matrix for voting\_classifier

[[235 3]

[ 58 5]]

Discussion:

The maximum training accuracy achieved is ~75% and the maximum testing accuracy is ~80%. The best testing accuracy is possible to be achieved using any of the above five classifiers by proper tuning of hyperparameters. Unfortunately, logistic regression and k-nearest neighbors performed poorly for the given data compared to the rest. However, they were better able to capture the false negatives as seen from the confusion matrix. By looking at the confusion matrix in general for all the classifiers, it seems like the number of true positives were more for both training and testing data, whereas the true negatives could have been the lowest.

As expected, the ensemble classifier with uniform weightage was able to capture the best of all the models and achieve close to 80% accuracy. The best model for ensemble classifier that used weighting proportional to testing accuracies of base models performed identically to the one with uniform weightage, however on changing random seeds, different confusion matrices were obtained. This may point to the case where the ensemble classifier needed models that were trained on more data which contained more useful features.

For neural networks increasing the number of hidden layers beyond three did not change the best result in any way. This might indicate that more data was necessary. For K nearest neighbors the increase in accuracy was observed to be proportional to the number of included nearest neighbors. Using 200 nearest neighbors would seem like an overuse, but it should work well for increasing number of data points. For logistic regression, a strong regularization seemed to work better than changing any other parameters. For Naïve Bayes technique, a huge improvement was observed on changing the prior over the data to be uniform. For Decision trees, choosing the splitter was crucial. Slightly different accuracies and somewhat different confusion matrices resulted on using splitter=random or splitter=best. It is reported in the results section.

Task 3:

Ensemble classifier built from all 7 models:

Parameters:

A weighting vector was used just like earlier (in task 2). The relative weightage given to each classifier was based on their respective best testing accuracies. The weights vector was chosen to be (read from left to right corresponding to Adaboosting, Random Forests, Neural Network (NN), KNN, Logistic Regression (LR), Naive Bayes (NB) and Decision Tree (DT))

W= [0.73,0.79,0.79,0.70,0.72,0.79,0.79]

The weights are directly the float values (up to first 2 digits after 0) for testing accuracy of each of the classifier

Results:

testing voting\_classifier performance

The mean training accuracy on voting\_classifier 0.678292117621

The mean testing accuracy on voting\_classifier 0.790693069307

Printing the confusion matrix for voting\_classifier

[[238 0]

[ 63 0]]

Discussion:

We noted that the maximum testing accuracy remained close to 80%. Using weighting based on individual base model testing acuuracy did not seem to change the results compared to uniform weighting. This may be attributed to the fact that all the models have close testing accuracies ~75%. Another problem that might have been faced is that the testing data had some features which the training data had less instances of. In general, the accuracies should increase with more training data.

A Principle Component analysis was done to analyze the nature of training and testing data. Using the PCA function of scikit learn (PCA.explained variance ratio) we could obtain an estimate of the ratio of variance and infer whether the training data is biased or not.

Getting a PCA visualization of the training data

pca explained variance ratio

[ 9.99819717e-01 1.63212310e-04 1.70706984e-05 1.91235021e-37]

This may mean that the training data contains more of a certain type of feature.

Getting a PCA visualization of the testing data

pca explained variance ratio

[ 9.99801349e-01 1.69673740e-04 2.89773049e-05 8.96061951e-37]

This may mean that the testing data contains more of a certain type of feature.

References:

1. <http://neuroph.sourceforge.net/tutorials/BloodTransfusionSc/blood_transfusion_sc.html>
2. <http://scikit-learn.org/stable/modules/ensemble.html>