Lab 5 – Ensemble Machine Learning (85)

We are going to see how we can combine different models to (potentially) improve the performance. In particular, we will look at

1) Boosted decision trees – We have multiple decision trees lined up in series. The first decision tree trains on the target with the input data. The second decision tree trains on the errors of the first tree. We continue training new trees on the error of the previous tree until we are satisfied with the performance.
2) Decision tree forest. – We have multiple decision trees lined up in parallel. Each tree trains on a subset of the data. We then combine the results together.
3) Combined heterogenous models. – We use different kinds of models in parallel and combine the results.
4) Cross validation – Create different instances of the same kind of model using all of the data for validation. Each data fold will act as validation once.

Data Prep

Make a copy of lab5_starter.ipynb.

I wanted to do training on a dataset that has some noise and has non-linear surfaces. To that end, I created a couple functions that will randomly generate spiral arms each of which has its own class. There are some global variables that you can set to guide the behavior of the generator. (Once you finish the lab, try playing with these parameters and observe the effect on the models.)

I then used lab3 to train DT, SVM and NN models on the dataset and recorded the performance. One note about computing metrics. The definition we gave for precision and sensitivity are for models that are binary (have only two classification).

TP	FP
FN	TN

And precision is TP/(FP)

But when we have more categories, the confusion matrix expands in size and we no longer have true positive and true negatives.

Hit	Miss	Miss
Miss	Hit	Miss
Miss	Miss	Hit

The accuracy generalizes well, add all the correct predictions and divide by total instances.

The other measures are trickier and involve combinations. For example, we could compute a precision for each of the categories and then combine using a weighted average.

Once we go beyond 2 categories, we have to tell Scikit how to compute the measures. We add average='weighted' to those calls. If you don't make this change, Scikit will complain that its default is 'binary' and your model is not. (We have two other options besides 'weighted' which are 'micro' and 'macro'.)

Other than that and the obvious changes to use the new generated data set called **spiral**, it is substantially the same.

```
print(confusion_matrix(y, y_pred))
print('Accuracy is  ', accuracy_score(y, y_pred))
print('Precision is ', precision_score(y, y_pred, average='weighted'))
print('Recall is    ', recall_score(y,y_pred, average='weighted'))
print('F1 is        ', f1_score(y, y_pred, average='weighted'))
```

One small change in presentation of the results, I combined the train and test results on the same line for easier comparison of the models.

Submission 1 of 12: Progress mark (10)
Screen shot of graph of spiral data frame.

Train an ADA boosted model.

Make a copy of the two cells (markdown and code) for the training/performance of the neural net classifier. Then use the **Paste Cells Below** option from the **Edit** menu.
Change the following lines:

```
### Train and evaluate ADA boosted model

from sklearn.ensemble import AdaBoostClassifier

ada_model = AdaBoostClassifier(n_estimators=150)
ada_model.fit(X,y)

y_pred = ada_model.predict(X)
print('Results for ADA on training data')
y_test_pred = ada_model.predict(X_test)
print('Results for ADA on test data')
```

1. Turning all references to use the AdaBoost model

2. We direct the model to train on the error up to 150 times. The default is 100, though we want to be careful as the more models we stack up, the more likely we end up overfitting.

Run the cell and you should see results that look something like: (Note that we are generating a random data set, so the set you have may not exhibit the same behavior. It should be relatively close.)

```
Results for ADA on train data
  Default settings
Confusion Matrix
[[723 160 184]
 [241 668 157]
 [173 204 689]]
Accuracy is   0.6502031884964051
Precision is  0.6506962900860015
Recall is     0.6502031884964051
F1 is         0.6501083847103486

Results for ADA on test data
  Default settings
Confusion Matrix
[[172  45  49]
 [ 69 164  34]
 [ 60  59 148]]
Accuracy is   0.605
Precision is  0.6080662434580344
Recall is     0.605
F1 is         0.6047187114675385
```

Record the results in the final cell.

|ADA|A,B 150 estimators|65.02|65.01|60.5|60.47|

Submission 2 of 12: Progress mark (10)
Screen shot of train/test performance for ada.

Submission 3 of 12: Analysis (5)

- How does the ADA model compare with the other models?
- Is the ADA boosted classifier overfitting?

Train a random forest model.

Make a copy of the two cells (markdown and code) for the training/performance of the ADA boosted classifier. Then use the **Paste Cells Below** option from the **Edit** menu.
In the code cell, change the following lines:

```
### Train and evaluate Random Forest model

from sklearn.ensemble import RandomForrestClassifier

rf_model = RandomForest(n_estimators=150)
rf_model.fit(X,y)

y_pred = rf_model.predict(X)
print('Results for Random Forest on training data')
y_test_pred = rf_model.predict(X_test)
print('Results for Random Forest on test data')
```

1. Turning all references to use the Random Forrest model
2. We direct the model to train in parallel 150 different decision trees. The default is 100, though we want to be careful as the more models we stack up, the more likely we end up overfitting.

Run the cell and you should see results that look something like:

```
Results for Random Forest on train data
 Default settings
Confusion Matrix
[[1067    0    0]
 [   0 1066    0]
 [   0    0 1066]]
Accuracy is   1.0
Precision is  1.0
Recall is     1.0
F1 is         1.0

Results for RandomForest on test data
 Default settings
Confusion Matrix
[[192  37  37]
 [ 27 215  25]
 [ 28  29 210]]
Accuracy is   0.77125
Precision is  0.7714964910991772
Recall is     0.77125
F1 is         0.7708379480414151
```

Record the results in the final cell.

Submission 4 of 12: Progress mark (10)
Screen shot of train/test performance for random forest classifier.

Submission 5 of 12: Analysis (5)
- How does the RF model compare with the other models?
- Is the RF classifier overfitting?

Ensembles of Different Kinds of Models.

Make a copy of the two cells (markdown and code) for the training/performance of the ADA boosted classifier. Then use the **Paste Cells Below** option from the **Edit** menu.
In the markdown cell, change the following lines:

```
### Train and evaluate Voting model
We build it out of three different kinds of classifiers
One each of DT, SVC and NN
```

In the code cell we need to add lines to import the models we will use

```
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

In the code cell we need to create the models that will be part of the voting classifier. This is the same as what we did before.

```
# Build the classifiers
vc1_dt = DecisionTreeClassifier()
vc2_svm = SVC()
vc3_nn  =  MLPClassifier(hidden_layer_sizes=(50, 25, 10),
                         solver='lbfgs')
```

We need to create a list of pairs naming each of the models we are going to use.

```
# Bundle into a list with names
vcList = [('DT',vc1_dt),
    ('SVM',vc2_svm),
    ('NN',vc3_nn)]
```

Now we are on familiar territory and need to train and evaluate the combined model.

```
vc_model = VotingClassifier(vcList)

vote_model.fit(X,y)

y_pred = vote_model.predict(X)
print('Results for Voting Ensemble on training data')
y_test_pred = vote_model.predict(X_test)
print('Results for Voting Ensemble on test data')
```

1. Turning all references to use the Voting classifier
2. We need to give the ensemble a list of the models it should use. The classifier will make copies of the model, so we could reuse the list.

Run the cell and you should see results that look something like. (Note that the

Results for Voting Ensemble on train data
Default settings
Confusion Matrix
[[966 64 37]
[72 930 64]
[77 39 950]]
Accuracy is 0.8896530165676774
Precision is 0.8901789462994087
Recall is 0.8896530165676774
F1 is 0.8896849773351814

Results for Voting Ensemble on test data
Default settings
Confusion Matrix
[[212 33 21]
[28 212 27]
[41 16 210]]
Accuracy is 0.7925
Precision is 0.793603023293865
Recall is 0.7925
F1 is 0.7927444532158884

Record the results in the final cell.

Submission 6 of 12: Progress mark (10)
Screen shot of train/test performance for voting classifier.

Submission 7 of 12: Analysis (5)
- How does the voting model compare with the other models?
- Is the voting classifier overfitting?

Cross Validation

As we train the model, there are a number of different effects that could change the results. As a couple examples, If we are doing a search it will be randomized and the behavior changes. If we train on a subset of the data, we may get overfitting to a different degree. If we have sequestered the test set, we would still like to have an idea of the performance of the model on the test set without peeking. To that end we are going to look at a technique to that will allow us to estimate that performance. The basic idea is that we are going to divide the training set into N folds. We then train the model N times. Each time a different fold will be used to evaluate the model and the other N-1 folds will comprise the training set. Since every instance is used once for evaluation, we get a better view of the data with respect to the model.

Make a copy of the cells for the data split.

In the markdown cell, change to:

```
### Cross Validate a Decision Tree model
```

The splitter we want to use should be one of the ones labeled KFold. This guarantees that the data is apportioned to each fold. We also do want each of the folds to maintain the even ratio between the classes

```
from sklearn.model_selection import StratifiedKFold'
```

We need to import the classifier and metrics

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix
```

In the code cell before the splitter, we want to create some empty lists

```
accuracy_list = np.array([])
f1_list = np.array([])
```

We need to create the splitter

```
splitter = StratifiedKFold(n_splits=10, shuffle=True, random_state=123)
```

1. We will have 10 folds
2. We shuffle before the split

We use the splitter to create the sets

```
for train_indices, validate_indices in splitter.split(spiral,
spiral['Class']):
    train_set = spiral.iloc[train_indices]
    validate_set = spiral.iloc[validate_indices]
    X = train_set[['A','B' ]]
    y = train_set['Class']

    X_validate = validate_set[['A','B']]
    y_validate = validate_set['Class']
```

1. This loop will iterate 10 times
2. We stratify on the target Class
3. We get the training set for the input features A and B
4. We get the vector for the target in the training set
5. Do similarly for the validation set

Time to compute the performance for each fold

```
model = DecisionTreeClassifier()
model.fit(X,y)
y_validate_pred = model.predict(X_validate)
accuracy = accuracy_score(y_validate, y_validate_pred)
print("Confusion Matrix")
print(confusion_matrix(y_validate, y_validate_pred))
print('Accuracy is ', accuracy)

accuracy_list = np.insert(accuracy_list, 0, accuracy)
```

6. Each time through we make a new decision tree classifier called model
7. We train the model
8. We see how well the model works on the validation fold
9. Compute the accuracy
10. Print the confusion matrix and accuracy (There are not too many of them)
11. Add the accuracy score at the front of our list

Once the loop is complete, we would like to take the values and compute an average and a standard deviation.

```
print("Accuracy values", accuracy_list)
print("   Average of ", np.average(accuracy_list))
print("   StDev of ", np.std(accuracy_list))
```

Run the cell and you should see final results that look something like.

```
Accuracy values [0.74185464 0.745    0.7775    0.6975    0.7625    0.7375
 0.7475    0.7725    0.77      0.7675   ]
   Average of  0.7519354636591478
   StDev of  0.02257794664999023
```

Submission 8 of 12: Analysis (5)

Does the cross validation give us a good idea of how well the decision tree will perform on the test set?

Submission 9 of 12: Progress (5)
Screen shot of results table.

Exploration

Pick one of the adjustable parameters for the data set and change it. You will use the new data set with a random forest classifier and compute the accuracy. Before doing that, make a prediction for the change in the accuracy.

Submission 10 of 12: Prediction (5)
- Value being changed and how much
- Predicted change in accuracy

Run the cells
1) Settings
2) Spirals
3) Data split
4) Random Forest

Submission 11 of 12: Progress (10)
- Screen shot data set graph
- Actual Random Forest performance

Submission 12 of 12: Analysis (5)
- Did your prediction match the actual performance.
- Propose an explanation