Exercise7_1

October 30, 2021

0.1 Exercise 7: Decision Trees and Ensemble Methods

[1]:
```
%matplotlib inline
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

0.2 A) Please use the Iris dataset for this part.

Data can be imported as follows from sklearn.datasets import load_iris

Let's use random_state=0 for splitting and building all models.

1) Fit decision tree with maximum depth (max_depth) of 2 and the default gini index for building the tree. Find the model accuracy (with test data).

- To visualize the tree (optional), first, import the graphviz package from terminal using the following:

```
brew install graphviz

OR

#conda install -c anaconda graphviz
#conda install -c anaconda python-graphviz
```

Then, we can use the package to visulaize the decision tree as follows:

```
from sklearn.tree import export_graphviz
import graphviz

 dot_data=export_graphviz(FittedTreeModel,class_names=dataset.target_names,   feature_na

graph = graphviz.Source(dot_data)
graph
```

[2]:
```
iris_dataset = load_iris()
X_train, X_test, Y_train, Y_test = train_test_split(iris_dataset['data'],␣
 ↪iris_dataset['target'], random_state=0)

m = 2
treeModel = DecisionTreeClassifier(max_depth=m)
treeModel.fit(X_train, Y_train)

preds = treeModel.predict(X_test)
acc = np.count_nonzero(preds==Y_test) / len(preds)
print('Accuracy is %.4f.' % acc)
```

```
Accuracy is 0.8947.
```

2) Use random forests for classification of Iris examples. The random forests combines 4 decision trees, each of maximum depth 2 and maximum number of features considered at each split is 2. What is the model accuracy? Compare preformance to previous part.

[3]:
```
forestModel = RandomForestClassifier(n_estimators=4, max_features=2,␣
 ↪max_depth=2, random_state=0)
forestModel.fit(X_train, Y_train)

preds = forestModel.predict(X_test)
acc = np.count_nonzero(preds==Y_test) / len(preds)
print('Accuracy is %.4f.' % acc)
```

```
Accuracy is 0.9474.
```

(AP): The accuracy for the random forests is improved by ~6 percentage points over the regular decision tree classifier.

3) Use AdaBoost with 4 decision tree models to perform the classification of the Iris species. What is the accuracy? Increase the number of models from 4 to 14. Find the accuracy. Comment on the results.

[4]:
```
for m in range(4, 15):
    boostModel = AdaBoostClassifier(n_estimators=m)
    boostModel.fit(X_train, Y_train)

    preds = boostModel.predict(X_test)
    acc = np.count_nonzero(preds==Y_test) / len(preds)
    print('Accuracy for ' + str(m) + ' models is %.4f.' % acc)
```

```
Accuracy for 4 models is 0.9737.
Accuracy for 5 models is 0.9737.
Accuracy for 6 models is 0.8947.
Accuracy for 7 models is 0.8947.
Accuracy for 8 models is 0.9737.
```

```
Accuracy for 9 models is 0.9737.
Accuracy for 10 models is 0.8947.
Accuracy for 11 models is 0.8947.
Accuracy for 12 models is 0.9737.
Accuracy for 13 models is 0.9737.
Accuracy for 14 models is 0.8947.
```

(AP): The accuracy seems to oscillate between 89% and 97% as the number of models are increased. This is likely because the dataset is not big enough to support adding several extra models.

4) Use Bagging with 4 decision tree models, and find the accuracy. Repeat with 14 models. Do you think it will overfit as the number of base models increases? Comment on results.

[5]:
```
for m in [4, 14]:
    baggingModel = BaggingClassifier(n_estimators=m)
    baggingModel.fit(X_train, Y_train)

    preds = baggingModel.predict(X_test)
    acc = np.count_nonzero(preds==Y_test) / len(preds)
    print('Accuracy for ' + str(m) + ' models is %.4f.' % acc)
```

```
Accuracy for 4 models is 0.9737.
Accuracy for 14 models is 0.9737.
```

(AP): Bagging is not generally susceptible to overfitting as the number of estimators increase, because all of the models are independent.

5) Use Bagging with 4 LDA base classifier instead of the default decision tree classifier, and find the accuracy. Hints:
-Change input argument of BaggingClassifier as follows base_estimator=LDA( )

-Need to add: import "from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA"

[6]:
```
baggingModel = BaggingClassifier(n_estimators=4, base_estimator=LDA())
baggingModel.fit(X_train, Y_train)

preds = baggingModel.predict(X_test)
acc = np.count_nonzero(preds==Y_test) / len(preds)
print('Accuracy is %.4f.' % acc)
```

```
Accuracy is 0.9737.
```

0.3 B) Adaboost algorithm

In this part we will implement the Adaboost algorithm without using the scikit-learn AdaBoost-Classifier method. We can however use the DecisionTreeClassifier of Scikit-learn to build the base models. We will use synthetic data below of two classes and 4 features. Implement the Adaboost Algorithm using two base models that are Decision Tree classifiers, each has a single split only.

Find the accuracy using the test data and verify the accuracy with that obtained using AdaBoost-Classifier function of scikit-learn. - Please split the X and Y given below into train and test sets. We will use the train set to build and fit the models, then evaluate with samples from the test set. Let's use random_state =0 - Initialize the weights of the training data based on the size of the training set. If the size of the training set is n, then the initial weight of each sample is 1/n. - We will have two base models, each is a Decision Tree classifier with a single split. - Following the Adaboost algorithm discussed in the lecture, you would need to evaluate weights of each sample as well as weight of each base model. - Note that the .fit method takes training data and can take weight vector for each sample (model.fit(X_train,Y_train,sample_weight=weights)). - After fitting each of the base models, and evaluating each model weight, predict the first sample in the test data (first sample is at index 0). You will need to write the code for combining the predicitions of the base models using their weights. Compare the final prediction of the first test example with the actual label to see if it is correct or not. - Find the overall accuracy using all the test data and verify using the built-in AdaBoostClassifier function.

[7]:
```
from sklearn.datasets import make_classification

#creating synthetic data from 2 classes and 4 features
X, Y = make_classification(n_samples=10000, n_features=4, n_classes=2,␣
 ↪n_clusters_per_class=1, weights=[0.5], random_state=0)
Y=2*Y-1 # this will just make class clabels encoded as 1 and -1 instead of 0␣
 ↪and 1

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0)
weights = [1/len(X_train)] * len(X_train)
alphas = []
clfs = []

for k in range(0, 2):
    clf = DecisionTreeClassifier(max_depth=1, random_state=0)
    clfs.append(clf)
    clf.fit(X_train, Y_train, sample_weight=weights)
    preds = clf.predict(X_train)

    # compute weighted error
    err = 0
    for wt, y_est, y in zip(weights, preds, Y_train):
        if y_est != y:
            err += wt

    # calculate alpha
    alpha = 0.5 * np.log((1-err) / err)
    alphas.append(alpha)

    # update weights
    for i in range(0, len(X_train)):
        if preds[i] != Y_train[i]:
```

```
            mult = -1
        else:
            mult = 1
        weights[i] = weights[i] * np.exp(-1 * alpha * mult)

    # normalize weights
    tot = sum(weights)
    weights = [wt / tot for wt in weights]
```

[8]:
```
# evaluate on first test sample
running_total = 0
preds = [clf.predict(X_test) for clf in clfs]

for alpha, pred in zip(alphas, preds):
    running_total += alpha * pred[0]

if running_total > 0:
    est = 1
else:
    est = -1

print(f'For the first test sample, the prediction is class {est}. Label is␣
 ↪class {Y_test[0]}.')
```

```
For the first test sample, the prediction is class 1. Label is class 1.
```

[9]:
```
# evaluate on all test samples
disc_preds = []

for i in range(0, len(X_test)):
    running_total = 0
    for alpha, pred in zip(alphas, preds):
        running_total += alpha * pred[i]

    if running_total > 0:
        disc_preds.append(1)
    else:
        disc_preds.append(-1)

acc = np.count_nonzero(disc_preds==Y_test) / len(disc_preds)
print('Accuracy is %.4f.' % acc)
```

```
Accuracy is 0.8808.
```

[10]:
```
# verify using built-in AdaBoostClassifier
```

```
boostModel =␣
 ↪AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1,␣
 ↪random_state=0),
                                  n_estimators=2, random_state=0)
boostModel.fit(X_train, Y_train)

preds = boostModel.predict(X_test)
acc = np.count_nonzero(preds==Y_test) / len(preds)
print('Accuracy is %.4f.' % acc)
```

```
Accuracy is 0.8808.
```