Design and Implementation of K Nearest Neighbors Algorithm and Boosting Techniques

Aditi Helekar

March 12, 2017

Contents:

Abstract

A brief introduction to K Nearest Neighbors algorithm, boosting approach is provided and is implemented in Python. The implementation is demonstrated to identify the appropriate prediction on Testing data by training the models with Digit Recognition dataset and Amazon Baby products reviews dataset.

1. Problem Statement:

- Implement k-Nearest Neighbor algorithm.
- Implement a boosting approach using one of the learning approaches you have implemented so far.
- Submit your implementation and analysis results for the two approaches using the Digit Recognition and Amazon Reviews datasets.

2. Approach:

I have implemented neural networks in Python using scikit-learn library. Scikit-learn is library designed for implementing Machine learning techniques in Python. Scikit-learn provides libraries for Ada Boost and K Nearest Neighbors.

K Nearest Neighbors:

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree.). Selection of various components is explained below:

class sklearn.neighbors.NearestNeighbors(n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=1, **kwargs)

n_neighbors: Number of nearest neighbors depends on variation in data set.

algorithm: For small data sets (N less than 30 or so), $\log(N)$ is comparable to N, and brute force algorithms can be more efficient than a tree-based approach. Both KDTree and BallTree address this through providing a *leaf size* parameter: this controls the number of samples at which a query switches to brute-force. This allows both algorithms to approach the efficiency of a brute-force computation for small N.

leaf_size: A larger leaf_size leads to a faster tree construction time, because fewer nodes need to be created. As leaf_size increases, the memory required to store a tree structure decreases.

Boosting Technique - AdaBoost:

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. In boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

class sklearn.ensemble.AdaBoostClassifier(base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None

base_estimator: The model used for classification such as DecitionTreeClassifier or Random ForestClassifier.

n_estimators :The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The number of weak learners is controlled by the parameter n_estimators. The learning_rate parameter controls the contribution of the weak learners in the final combination.

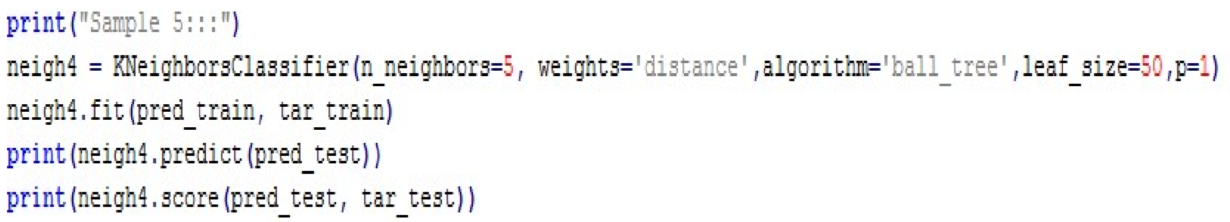
3. Implementation:

The basic algorithm used for both implementations is described below:
1. Read the csv file using pandas.io library and store it as a Dataframe.
2. Classify the attributes as samples and features to apply it to NearestNeighbors and AdaBoostClassifier. Features are also referred as predictors and samples as responses.
3. Create an object of Classifier imported from scikit learn library.
4. Using fit function, fit the dataframe to tree model. Fit function finds patterns in the given dataset.
5. For cross-validation, divide the training dataset in train and test data using train_test_split function.
6. Predict the behavior of model using predict function with test data.
7. After cross validation use the test dataset and predict again. 8. Calculate accuracy using score function.
9. Plot the loss graphs using matplotlib library.

For Amazon Dataset, I first removed all the stopwords using the nltk toolkit. Later on, I converted the words to vectors and found the occurrences of each meaningful words using CountVectorizer.

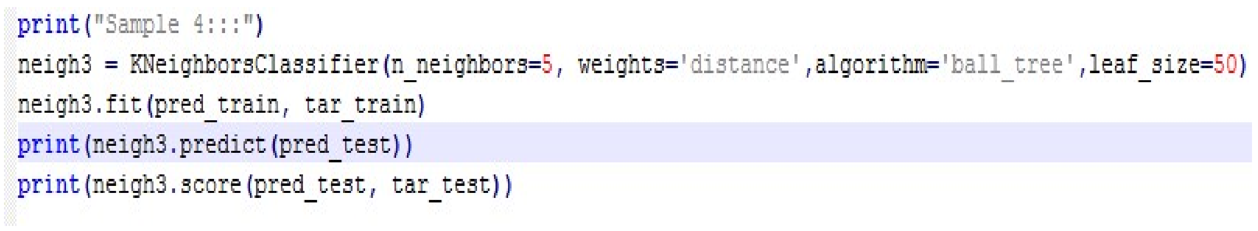
K Nearest Neighbors:

I have verified the results by modifying number of neighbors, algorithm, leaf_size. Below code snippet shows the outline of KNN for Digit Recognition Dataset:



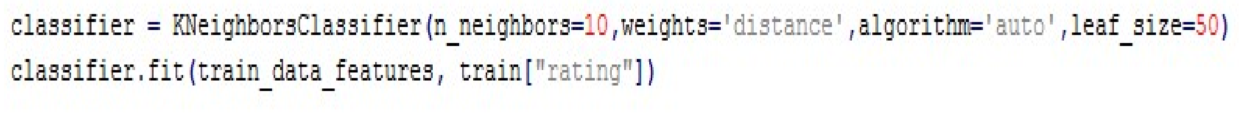
```
print("Sample 5:::")
neigh4 = KNeighborsClassifier(n_neighbors=5, weights='distance',algorithm='ball_tree',leaf_size=50,p=1)
neigh4.fit(pred_train, tar_train)
print(neigh4.predict(pred_test))
print(neigh4.score(pred_test, tar_test))
```

In the above figure, p=1 refers to using distance metrics as Manhattan distance rather than Euclidean distance.



```
print("Sample 4:::")
neigh3 = KNeighborsClassifier(n_neighbors=5, weights='distance',algorithm='ball_tree',leaf_size=50)
neigh3.fit(pred_train, tar_train)
print(neigh3.predict(pred_test))
print(neigh3.score(pred_test, tar_test))
```

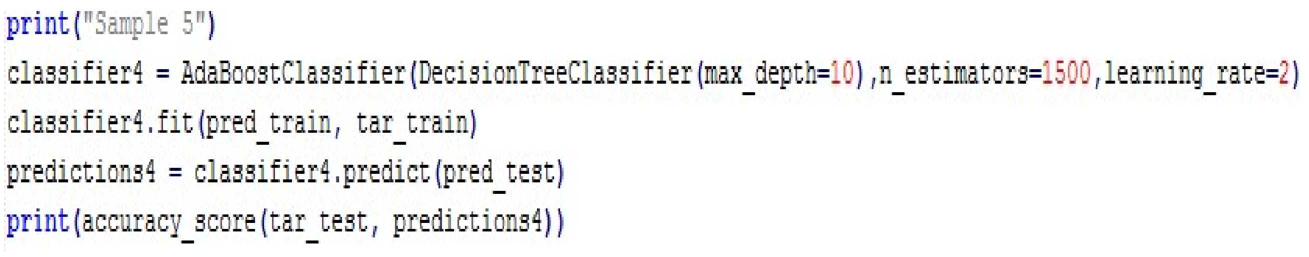
Below code snippet shows the outline of KNN for Amazon Dataset:



```
classifier = KNeighborsClassifier(n_neighbors=10,weights='distance',algorithm='auto',leaf_size=50)
classifier.fit(train_data_features, train["rating"])
```

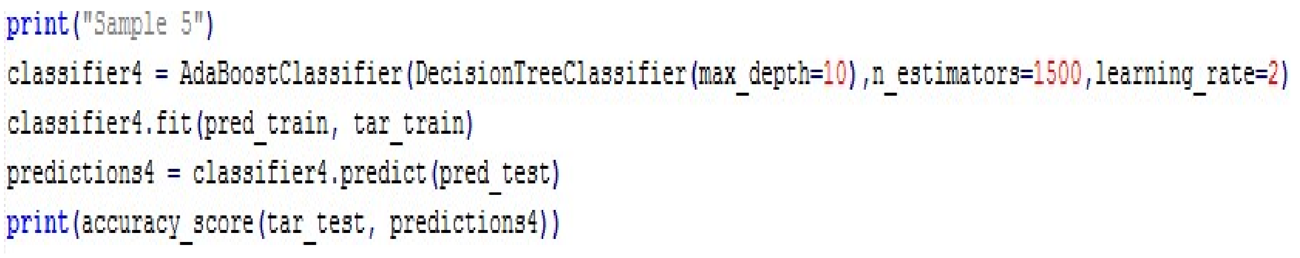
<u>Boosting Technique - AdaBoost:</u>

I have verified the results by modifying number of estimators, max_depth, learning rate. Below code snippet shows the outline of AdaBoost for Digit Recognition Dataset:



```
print("Sample 5")
classifier4 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=10),n_estimators=1500,learning_rate=2)
classifier4.fit(pred_train, tar_train)
predictions4 = classifier4.predict(pred_test)
print(accuracy_score(tar_test, predictions4))
```

Below code snippet shows the outline of AdaBoost for Amazon Dataset:

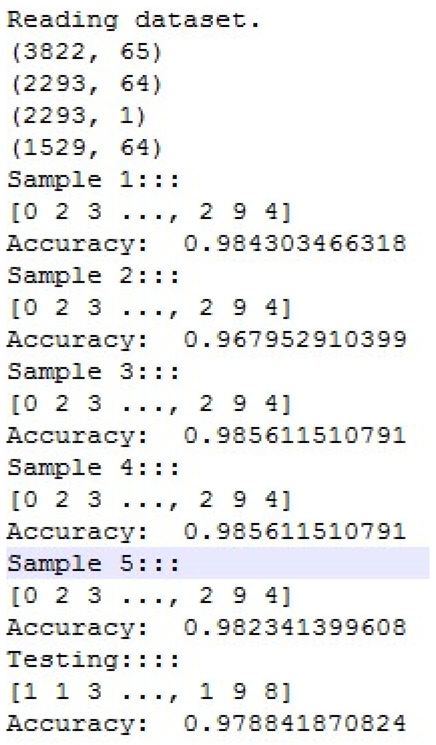


```
print("Sample 5")
classifier4 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=10),n_estimators=1500,learning_rate=2)
classifier4.fit(pred_train, tar_train)
predictions4 = classifier4.predict(pred_test)
print(accuracy_score(tar_test, predictions4))
```

4. Observations:

K Nearest Neighbors:

1. For Digit Recognition dataset:

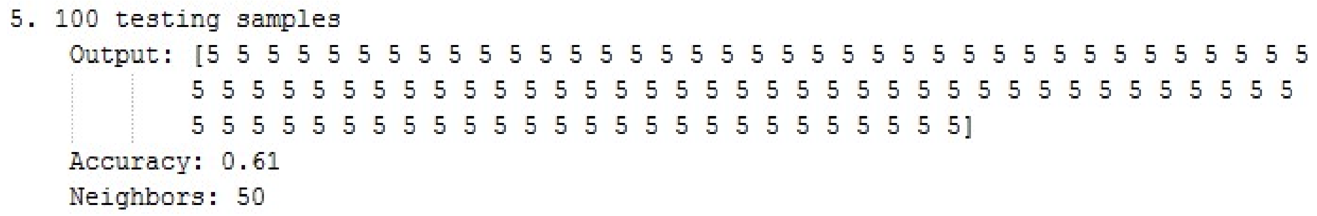


```
Reading dataset.
(3822, 65)
(2293, 64)
(2293, 1)
(1529, 64)
Sample 1:::
[0 2 3 ..., 2 9 4]
Accuracy:  0.984303466318
Sample 2:::
[0 2 3 ..., 2 9 4]
Accuracy:  0.967952910399
Sample 3:::
[0 2 3 ..., 2 9 4]
Accuracy:  0.985611510791
Sample 4:::
[0 2 3 ..., 2 9 4]
Accuracy:  0.985611510791
Sample 5:::
[0 2 3 ..., 2 9 4]
Accuracy:  0.982341399608
Testing::::
[1 1 3 ..., 1 9 8]
Accuracy:  0.978841870824
```

Sample 1: neigh = KNeighborsClassifier(n_neighbors=3)
Sample 2: neigh1 = KNeighborsClassifier(n_neighbors=50)
Sample 3: neigh2 = KNeighborsClassifier(n_neighbors=5, weights='distance')
Sample 4: neigh3 = KNeighborsClassifier(n_neighbors=5, weights='distance',algorithm='ball_tree',leaf_size=50)
Sample 5: neigh4 = KNeighborsClassifier(n_neighbors=5, weights='distance',algorithm='ball_tree',leaf_size=50,p=1)

We get maximum accuracy for Sample 4 with 5 nearest neighbors, leaf size of 5, Distance metric of Euclidean distance.

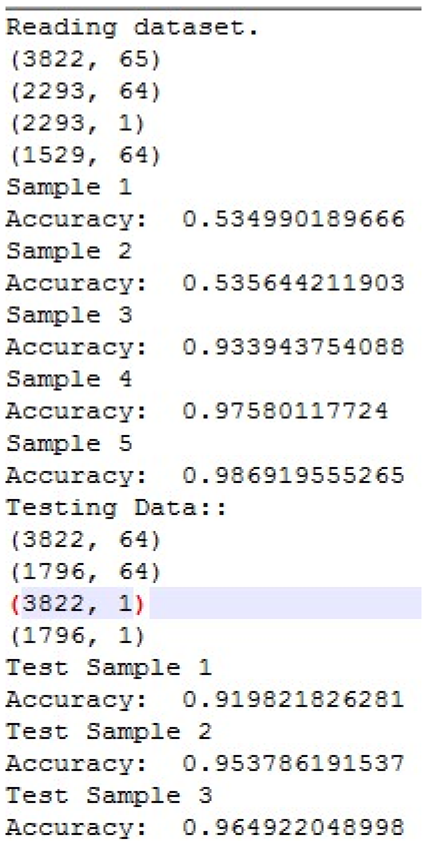
2. For Amazon Dataset:



```
5. 100 testing samples
    Output: [5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
             5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
             5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
    Accuracy: 0.61
    Neighbors: 50
```

Boosting Technique - AdaBoost:

1. For Digit Recognition Dataset:

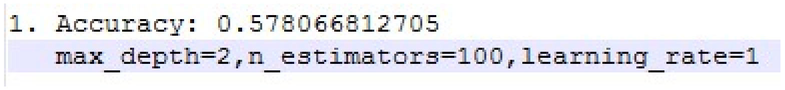


```
Reading dataset.
(3822, 65)
(2293, 64)
(2293, 1)
(1529, 64)
Sample 1
Accuracy:  0.534990189666
Sample 2
Accuracy:  0.535644211903
Sample 3
Accuracy:  0.933943754088
Sample 4
Accuracy:  0.97580117724
Sample 5
Accuracy:  0.986919555265
Testing Data::
(3822, 64)
(1796, 64)
(3822, 1)
(1796, 1)
Test Sample 1
Accuracy:  0.919821826281
Test Sample 2
Accuracy:  0.953786191537
Test Sample 3
Accuracy:  0.964922048998
```

Sample 1: classifier = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=500,learning_rate=1)
Sample 2: classifier1 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=1000,learning_rate=1)
Sample 3: classifier2 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=1000,learning_rate=2)
Sample 4: classifier3 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=5),n_estimators=1000,learning_rate=2)
Sample 5: classifier4 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=10),n_estimators=1500,learning_rate=2)

We get maximum accuracy for Sample 5 with Decision Tree Classifier, maximum depth of tree is 10, Number of maximum samples is 1500, learning rate is 2.

2. <u>For Amazon Dataset:</u>



```
1. Accuracy: 0.578066812705
   max_depth=2,n_estimators=100,learning_rate=1
```

5. Conclusion:

K Nearest Neighbors:

1. In K nearest neighbor technique, we found that as we increase the number of neighbors the accuracy decreases which means that to get precise result , number of neighbors should be as low as possible.
2. Distance metric Euclidean distance gives better results than Manhattan distance.
3. Leaf size reduces the memory usage so smaller leaf_size is advisable.

Boosting Technique - AdaBoost:

1. In AdaBoost technique, we found that as we increase the number of estimators, the accuracy increases but this number depends on dataset size.
2. Using Decision tree model as weak learner, varying the maximum depth of decision tree gives better results. As we increase the depth, accuracy increases.
3. Changing the learning rate does not change the accuracy much.

6. **References:**

1. Sklearn-neighbors.NearestNeighbors http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors
2. Nearest Neighbors, http://scikit-learn.org/stable/modules/neighbors.html
3. Ensemble methods, http://scikit-learn.org/stable/modules/ensemble.html#adaboost