Machine Learning

Weekly Project Report

Quantcats

Prachee Javiya AU1841032
Kaushal Patil AU1841040
Arpitsinh Vaghela AU1841034
Vrunda Gadesha AU2049007

Tasks Performed: Week 5

01 KNN
02 XG Boosting
03 Stacking Models

1. KNN

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

This algorithm can be used for classification and regression. For stock performance prediction, we deal with all the continuous variable, Thus KNN-Regression is suitable for this Problem definition. We found MSE using KNN Algorithm and defined elbow function to find the appropriate value of K.

```
#importing KNN classifier and metric F1score
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import mean_squared_error as mse
```

```
# Creating instance of KNN
reg = KNN(n_neighbors = 11)

# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating MSE
test_predict = reg.predict(test_x)
k = mse(test_predict, test_y)
print('Test MSE    ', k )
```

```
Test MSE     12957498854.792038
```

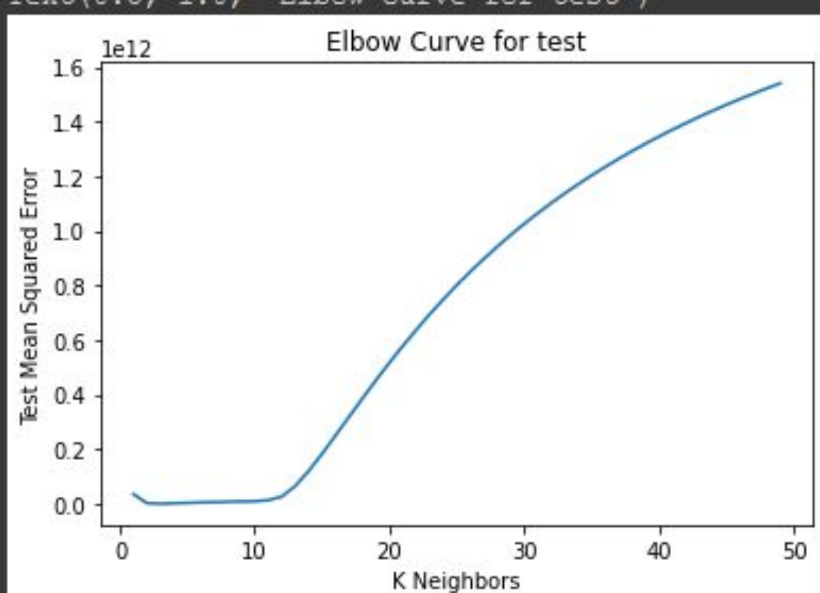
Elbow for Classifier

```
[ ]  def Elbow(K):
       #initiating empty list
         test_mse = []

       #training model for evey value of K
         for i in K:
             #Instance of KNN
             reg = KNN(n_neighbors = i)
             reg.fit(train_x, train_y)
             #Appending mse value to empty list claculated using the predictions
             tmp = reg.predict(test_x)
             tmp = mse(tmp,test_y)
             test_mse.append(tmp)

         return test_mse
```

```
[ ]  # Defining range of K
     k = range(1,50)
```

```
[ ]  # plotting the Curves
     plt.plot(k, test)
     plt.xlabel('K Neighbors')
     plt.ylabel('Test Mean Squared Error')
     plt.title('Elbow Curve for test')
```

```
Text(0.5, 1.0, 'Elbow Curve for test')
```



According to the elbow curve, here we select the value of K. (in first case, K=3 and in second case, K=4.

```
[ ] # Creating instance of KNN
    reg = KNN(n_neighbors = 3)

    # Fitting the model
    reg.fit(train_x, train_y)

    # Predicting over the Train Set and calculating F1
    test_predict = reg.predict(test_x)
    k = mse(test_predict, test_y)
    print('Test MSE    ', k )

    Test MSE     267279773.0723721
```

```
[ ] # Creating instance of KNN
    reg = KNN(n_neighbors = 2)

    # Fitting the model
    reg.fit(train_x, train_y)

    # Predicting over the Train Set and calculating F1
    test_predict = reg.predict(test_x)
    k = mse(test_predict, test_y)
    print('Test MSE    ', k )

    Test MSE     2274296617.1312137
```

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

XG Boosting

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.
We have used XGBClassifier library and have applied this on numeric fields in our dataset. Input values are standardized using StandardScaler library.

```
# split data into train and test sets
seed = 7
test_size = 0.10
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
```

Decision trees reach to level 10 to reach this accuracy

```
[45]: y_pred = model.predict(X_test)
      predictions = [round(value) for value in y_pred]

[46]: # evaluate predictions
      accuracy = accuracy_score(y_test, predictions)
      print("Accuracy: %.2f%%" % (accuracy * 100.0))

      Accuracy: 48.44%

[47]: y_pred = model.predict(X_train)
      predictions = [round(value) for value in y_pred]
      accuracy = accuracy_score(y_train, predictions)
      print("Accuracy: %.2f%%" % (accuracy * 100.0))

      Accuracy: 89.34%
```

Model Stacking

This algorithm involves building a meta-model based on prediction outputs of other different models. In our implementation, we have stacked lasso, random forest and gradient boosting algorithms.

```
lasso_pipeline = make_pipeline(LogisticRegression(random_state=0,solver='lbfgs', max_iter=1000))

rf_pipeline = make_pipeline(RandomForestClassifier(random_state=42,max_depth=18))

gradient_pipeline = make_pipeline(HistGradientBoostingClassifier(random_state=0,max_depth=18))

estimators = [('Random Forest', rf_pipeline),
              ('Lasso', lasso_pipeline),
              ('Gradient Boosting', gradient_pipeline)]

clf = StackingClassifier(estimators=estimators,
                                       final_estimator=LogisticRegression())
```

```
clf = StackingClassifier(estimators=estimators,
                                       final_estimator=LogisticRegression())
```

```
clf.fit(X_train, y_train).score(X_test, y_test)
```

```
0.4966003739588645
```

Outcomes

Accuracy scores

- K-nearest neighbours
- XG Boosting
    - Train accuracy : 89%
    - Test accuracy : 48%
- Model Stacking : 49%

Upcoming Week

01 K fold cross validation