CST4050 CW1 2019

December 13, 2019

0.1 CST 4050 - Coursework 1

*** By Tan Le Ping***

0.1.1 Task 1 - Summary of Data

- **Import the required libraries to load and manipulate data.**

[44]:
```python
import pandas as pd
import numpy as np
import seaborn as sns

mydata = pd.read_csv("synthetic.csv")
```

- **To view the first few row of data.**

[45]:
```python
# To view the first 5 rows of the data
mydata.head()
```

[45]:
```
          x1         x2         x3         x4          x5          x6         x7  \
0 -14.698830  2.369710  1.089267 -1.262030 -15.650082 -16.665997  15.909853
1  -8.457451  2.182712  0.972360 -4.255289 -11.524392  -4.843399   9.557964
2  -6.541517  1.263892 -0.494469 -2.562072  -8.979410 -23.632245  15.740920
3 -18.139840  1.569545 -3.286717 -4.255045 -16.146687 -25.893126  12.005963
4 -12.500957  2.313632  5.227138  2.586718 -15.022213  -3.105726  18.070314

          x8         x9        x10  ...       x22        x23       x24  \
0 -11.121045  18.275820 -2.405075  ... -5.421817  15.233291 -3.484405
1 -10.145921   6.655710 -2.821156  ... -5.398857  20.342647 -5.395054
2  -4.460916 -16.528412 -3.901285  ... -5.339781  10.859401 -2.095555
3  -2.228017   5.853151 -2.951831  ... -5.652446  -8.674892 -9.665123
4  -7.745197   0.300133 -3.364458  ... -5.551594  13.195368 -5.089818

         x25        x26        x27       x28        x29       x30  y
0   2.755223   9.766386   6.419560  6.618973  15.171849  1.926773  1
1   2.816668  14.932127   9.134028  4.826775  12.077634  3.397375  0
2   2.945595  14.778588   2.711564 -0.090958  -5.467509  3.088641  0
3   8.876766  22.335086  10.194627  2.720710  -1.787331 -0.291131  0
```

```
4  12.362742  22.624796   4.407471  3.022274  -1.705888  6.650217  0

[5 rows x 31 columns]
```

- **Information of data.**

[46]:
```
mydata.describe()
```

[46]:
```
                x1           x2           x3           x4           x5  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean    -13.028746     2.182041    -0.331036    -1.501078   -12.622918
std       3.659720     1.314388     4.259927     1.922640     3.604514
min     -25.548066    -1.599455   -14.930338   -10.215498   -24.600418
25%     -15.588659     1.285855    -3.149624    -2.808884   -15.109200
50%     -13.072938     2.170483    -0.367062    -1.510223   -12.498793
75%     -10.534016     3.021294     2.485166    -0.237209   -10.214818
max      -2.382520     6.026316    14.980421     5.101086     2.182904

                x6           x7           x8           x9          x10  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean    -10.854249    15.199978    -1.989472     6.407913    -2.926042
std       9.750920     7.206344     6.454849    16.872978     0.673362
min     -55.753091    -4.320908   -22.643235   -51.040173    -4.907236
25%     -17.120274    10.231755    -6.188742    -4.853568    -3.383162
50%     -11.170167    15.196222    -2.026093     6.431541    -2.928571
75%      -4.522221    19.901376     2.392737    18.145285    -2.487445
max      23.826332    36.646915    19.820630    55.897492    -0.712244

            ...          x22          x23          x24          x25  \
count       ...  1000.000000  1000.000000  1000.000000  1000.000000
mean        ...    -5.472288    10.543841    -6.003123     3.746927
std         ...     0.272104     8.311382     1.873970     4.962534
min         ...    -6.378320   -14.553686   -12.804169   -10.970233
25%         ...    -5.666194     4.728117    -7.268277     0.279869
50%         ...    -5.467538    10.698797    -5.919298     3.841361
75%         ...    -5.287631    16.268073    -4.677299     7.306957
max         ...    -4.671847    36.154495    -0.188857    20.068337

               x26          x27          x28          x29          x30  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     18.425973     2.742845     3.475475     1.864313    -1.137531
std       6.134947     7.049830     2.048401    10.351793     8.543692
min      -1.014732   -18.778590    -2.594584   -30.715194   -27.231646
25%      14.232877    -2.051034     2.102943    -4.779697    -6.883752
50%      18.301716     2.989103     3.488600     1.362105    -1.012529
75%      22.495502     7.393532     4.913292     8.998301     4.753629
max      36.539176    25.577773     9.590002    31.900767    31.280122
```

```
                   y
count  1000.000000
mean      0.145000
std       0.352277
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000

[8 rows x 31 columns]
```

As per table above, data shown is not stardardised. As to retain test data unseen, it will standardisation will upon K-fold.

Below instructions to check missing value, count of target's value and the number of rows and columns of the data.

[47]:
```
mydata.isnull().sum().sum()
```

[47]: 0

[48]:
```
mydata.shape
```

[48]: (1000, 31)

[49]:
```
mydata['y'].value_counts()
```

[49]:
```
0    855
1    145
Name: y, dtype: int64
```

- The data consists of 1000 observations, 30 features (independent variables) and 1 dependent variable (y).
- Each features (independent variables) is numeric.
- The target (dependent variable) is binomial categorical 0 and 1.
- No missing values in the data.

0.1.2 Task 2 - 10-fold cross validation

- **Import the required libraries to train-test and KFold cross validation on data.**

[50]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

- **Define X and y as independent variable (features) and dependent variable(target) respectively.**

[51]:
```
X = pd.DataFrame(mydata.drop('y', axis = 1))
y = pd.DataFrame(mydata['y'])
```

- **Train and test split 'mydata' into training and test data. Training data will be used to build the classifier model while the test data will be keep for validation of the model.**

[52]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.20,␣
↪random_state=3)
```

Test set above, X_test and y_test will be hold out to cross validate the final clasiifier model.

[53]:
```
mydata2 = pd.DataFrame(np.concatenate((X_train, y_train), axis =1))

# Define X2 and y2 for training sets independent variable (features) and␣
↪dependent variable(target) respectively.
X2 = mydata2.drop([30], axis = 1)
y2 = mydata2[30]
```

- **Stratified KFold is used as it is better to sort unbalanced target variables.**

[54]:
```
nfolds = 10
sf = StratifiedKFold(n_splits=nfolds, shuffle=True, random_state=38)
sf.get_n_splits(X2,y2)
```

[54]:
```
10
```

- ** Train a K-Nearest Neighbors model with 'mydata2' and stratified cross validation. Train data will be standardised upon stratified KFold**

[55]:
```
train_scores = np.array([])
test_scores = np.array([])

for train_index, test_index in sf.split(X2,y2):
    x_train, x_test = X2.loc[train_index],X2.loc[test_index]
    y_train, y_test = y2.loc[train_index],y2.loc[test_index]

    scaler = StandardScaler()
    scaler.fit(x_train)
    x_train = scaler.transform(x_train)
    X_test = scaler.transform(x_test)

    knn = KNeighborsClassifier()
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)

    train_score = knn.score(x_train, y_train)
    test_score = knn.score(x_test,y_test)
    accuracy_score = metrics.accuracy_score(y_test, y_pred)
```

```
    train_scores = np.append(train_scores, train_score)
    test_scores = np.append(test_scores, test_score)


print("The default K-Nearest Neighbors Classification accuracy: ",␣
 ↪accuracy_score)
print("Average train score: ", train_scores.mean())
print("Average test score: ", test_scores.mean())
```

```
The default K-Nearest Neighbors Classification accuracy:  0.8
Average train score:  0.8655555555555555
Average test score:  0.79125
```

The default model shown overfitting which train accuracy is relatively higher as compared to the test accuracy.Thus, tuning parameters will be carried out to improve the model.

0.1.3 Task 3 - Parameters Tuning for K-Nearest neighbors

[56]:
```
knn
```

[56]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

- **Below tuning is to seek the optimal values of parameters: n_neighbours.**

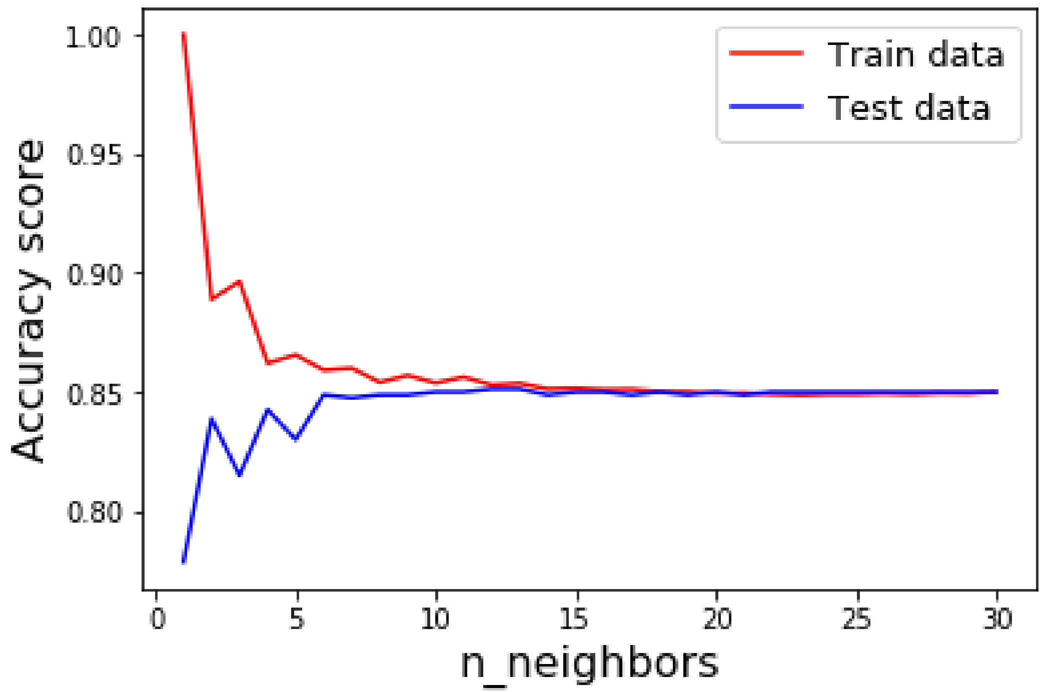
[57]:
```
k_range = list(range(1,31))

penalties = np.array([])

avg_k_scores_train = np.array([])
avg_k_scores_test = ([])

for k in k_range:
    k_scores_train = np.array([])
    k_scores_test = ([])

    # Define index for Stratified kfolds
    for train_index, test_index in sf.split(X2,y2):
        x_train, x_test = X2.loc[train_index],X2.loc[test_index]
        y_train, y_test = y2.loc[train_index],y2.loc[test_index]

        # Standardisation of data to get optima result via scalerstandard
        scaler = StandardScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)
```

```
        x_test = scaler.transform(x_test)

        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(x_train, y_train)
        y_pred = knn.predict(x_test)

        accuracy_score_k = metrics.accuracy_score(y_test, y_pred)

        train_score = knn.score(x_train, y_train)
        test_score = knn.score(x_test, y_test)


        #scores = metrics.accuracy_score(y_test, y2_pred)
        #scores2 = metrics.accuracy_score(y2_test, y2_pred)
        k_scores_train = np.append(k_scores_train, train_score)
        k_scores_test = np.append(k_scores_test, test_score)

    penalties = np.append(penalties,k)

    avg_k_scores_train = np.append(avg_k_scores_train, k_scores_train.mean())
    avg_k_scores_test = np.append(avg_k_scores_test, k_scores_test.mean())

print("KNN with tuned n_neighbors:")
print("Classification accuracy:", accuracy_score_k)
print("Average train score:", avg_k_scores_train.mean())
print("Average test score:", avg_k_scores_test.mean())
```

```
KNN with tuned n_neighbors:
Classification accuracy: 0.85
Average train score: 0.8602222222222223
Average test score: 0.8448750000000002
```

Below is the plotting of accuracy scores of train and test chages over the Knn's parameter n_neighbors values.

[58]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(k_range, avg_k_scores_train, 'r', label='Train data')
plt.plot(k_range, avg_k_scores_test, 'b', label='Test data')
plt.xlabel('n_neighbors', fontsize=16)
plt.ylabel('Accuracy score', fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.show()
```



The is overfitting pattern on the left with high accuracy gap between train and test, whereas underfitting occured on the right with low accuracy gap.

The optimal n_neighbors value which offer the best trade-off between overfitting and underfitting will be investigate as below:

[59]:
```
tunnedk = penalties[np.argmax(avg_k_scores_test)]

print ('The best n_neighbors value is', tunnedk)
print ()
```

```
The best n_neighbors value is 12.0
```

0.1.4 Task 4 - Compute accuracy of tuned model with baseline.

** Validate below tuned model(Knn with n_neighbors =12) with cross validation hold-out dataset.**

[60]:
```
knn_tuned = KNeighborsClassifier(n_neighbors=12)

#fit tuned model with Xtest, ytest(validation set)
knn_tuned.fit(X_test, y_test)

y_pred = knn_tuned.predict(X_test)
tuned_model_score = metrics.accuracy_score(y_test, y_pred)
```

```
print("Validated tuned classification accuracy :",tuned_model_score)
```

```
Validated tuned classification accuracy : 0.85
```

KNN with tuned n_neighbors: Classification accuracy: 0.85 Average train score: 0.8602222222222223 Average test score: 0.844875000000000

Default Knn model: Classification accuracy: 0.8 Average train score: 0.8655555555555555 Average test score: 0.79125

As per above data, Classification accuracy has increased about 5% from 80% of default Knn classifier to 85% of tuned knn classifier. The tuned model has reduce the gap of overfitting and underfitting. The parameter n_neighbors with value 12 is the best trade-off in between overfitting and underfitting by reducing the bias and variance of model.Accuarcy score of model does not increase after prameter value 12 which serve as the baseline.

[61]:
```
# To check model performance build with mydata2 (pred2 vs y_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
from sklearn.metrics import accuracy_score
print("Accuracy score:", accuracy_score(y_test, y_pred))
from sklearn.metrics import roc_auc_score
print("Roc_Auc_score :", roc_auc_score(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      1.00      0.92        68
         1.0       0.00      0.00      0.00        12

   micro avg       0.85      0.85      0.85        80
   macro avg       0.42      0.50      0.46        80
weighted avg       0.72      0.85      0.78        80

Confusion Matrix:
[[68  0]
 [12  0]]
Accuracy score: 0.85
Roc_Auc_score : 0.5

/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

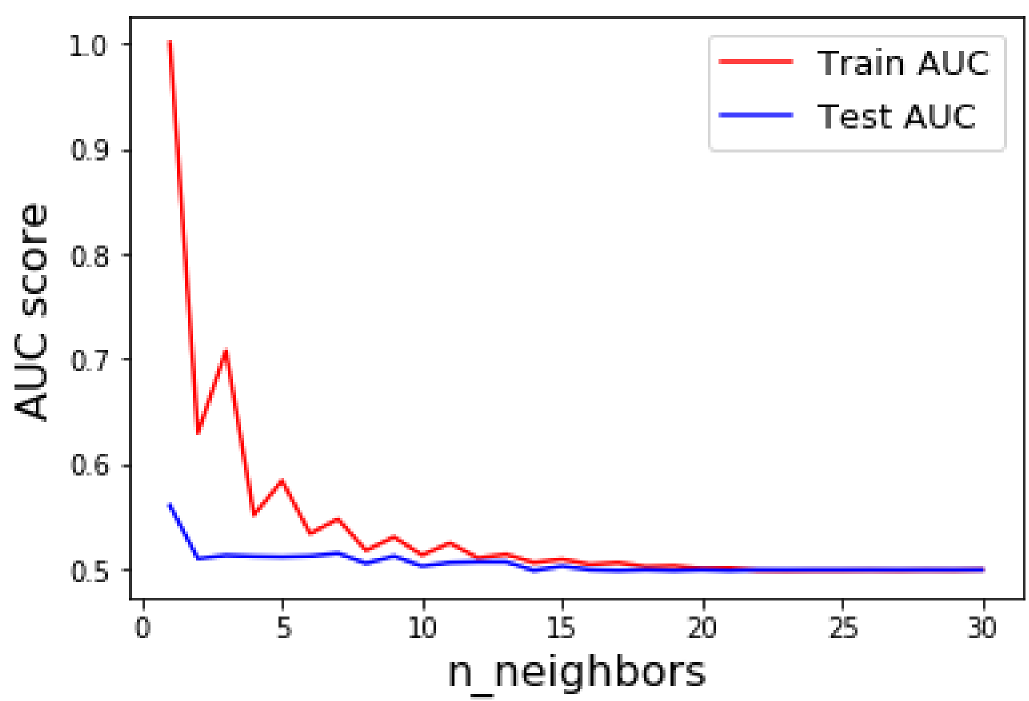
Confusion matrix of this tuned Knn model shown the prdiction of 68 True negative, 12 false negative whereas 0 for both False positive and True postive.

The classification accuracy 85% accurately predicted the True positive and True negative. Whereas the Classification error is 15% which predicted fasle postive and false negative output. The recall is the performance of true positive for this model is 0%,wheras 100% predicting True negative. The F1-score is 92% predicted 0 output.

[62]:
```
from sklearn.metrics import roc_curve, auc
# To seek the optimal value of K for KNN
k_range = list(range(1,31))

penalties = np.array([])

avg_train_results = np.array([])
avg_test_results = []

for k in k_range:
    train_results =np.array([])
    test_results =[]

    # Define index for Stratified kfolds
    for train_index, test_index in sf.split(X2,y2):
        x_train, x_test = X2.loc[train_index],X2.loc[test_index]
        y_train, y_test = y2.loc[train_index],y2.loc[test_index]

        # Standardisation of data to get optima result via scalerstandard
        scaler = StandardScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)
        x_test = scaler.transform(x_test)

        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(x_train, y_train)

        train_pred = knn.predict(x_train)
        false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,␣
↪train_pred)
        train_roc_auc = auc(false_positive_rate, true_positive_rate)
        #train_results.append(roc_auc)
        train_results = np.append(train_results, train_roc_auc)
```

```
        y_pred = knn.predict(x_test)
        false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_pred)
        test_roc_auc = auc(false_positive_rate, true_positive_rate)
        #test_results.append(roc_auc)
        test_results = np.append(test_results, test_roc_auc)

    penalties = np.append(penalties,k)

    avg_train_results = np.append(avg_train_results, train_results.mean())
    avg_test_results = np.append(avg_test_results, test_results.mean())
```

[63]:
```
#plt.plot(k_range, accuracy_score, 'c', label='KNN Model')
plt.plot(k_range, avg_train_results, 'r', label='Train AUC')
plt.plot(k_range, avg_test_results, 'b', label='Test AUC')
plt.xlabel('n_neighbors', fontsize=16)
plt.ylabel('AUC score', fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.show()
```



From the AUC score, it shows that the model is overfitting with lower value of n_neighbours whereas prefectly make prediction afer the trade off n_neighbors =12.

KNeighborsClassifier with best trade off n_neighbors=12