[CM7]

[29]:
```
# fitting the model on training set
knn =  KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train.values.ravel())

# predicting the target on test set
y_pred = knn.predict(X_test)

#calculating accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the basic KNN model with best k on the test set is', accuracy *␣
 ↪100, '%')

# calculating AUC
pred_prob = knn.predict_proba(X_test)
auc = roc_auc_score(y_test, pred_prob, average = 'macro', multi_class = 'ovr')
print('AUC:', auc)

# calculating f-score
f_score = f1_score(y_test, y_pred, average = 'weighted')
print('f-score:', f_score)
```

```
The accuracy of the basic KNN model with best k on the test set is 90.47619047619048 %
AUC: 0.9912393162393163
f-score: 0.9081458224315365
```

[30]:
```
# fitting the model on training set
knn =  KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train, y_train.values.ravel())

# predicting the target on test set
y_pred = knn.predict(X_test)

#calculating accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the basic KNN model with best k on the test set is', accuracy *␣
 ↪100, '%')

# calculating AUC
pred_prob = knn.predict_proba(X_test)
auc = roc_auc_score(y_test, pred_prob, average = 'macro', multi_class = 'ovr')
print('AUC:', auc)

# calculating f-score
f_score = f1_score(y_test, y_pred, average = 'weighted')
print('f-score:', f_score)
```

```
The accuracy of the basic KNN model with best k on the test set is 85.71428571428571 %
AUC: 0.9928418803418803
f-score: 0.8627943485086342
```

Upon testing the model with k values of 5 & 10, we observe that k=5 has a higher accuracy and f-score

compared to k=10.

Improved Model

[31]:
```
# Improved Model

# importing libraries
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

#Standard Scalar Normalization

# sc = StandardScaler()
# # Compute the mean and standard deviation based on the training data
# sc.fit(X_train)
# # Scale the training data to be of mean 0 and of unit variance
# X_train_normalized = sc.transform(X_train)
# # Scale the test data to be of mean 0 and of unit variance
# X_test_normalized = sc.transform(X_test)
# # Scale the validation data to be of mean 0 and of unit variance
# X_val_normalized = sc.transform(X_val)

# MinMax Scaler Normalization

scaler_min_max = MinMaxScaler()
# Fit object to data
scaler_min_max.fit(X_train)
# Get transformed train data
X_train_normalized = scaler_min_max.transform(X_train)
# Get transformed test data
X_test_normalized = scaler_min_max.transform(X_test)
# Get transformed val data
X_val_normalized = scaler_min_max.transform(X_val)

# train the model with the classifier's default parameters
#using different weighting schemes (default, manhatten, eculidean)

knn = KNeighborsClassifier(metric='manhattan', weights='distance')
#knn = KNeighborsClassifier(metric='manhattan', weights='distance')
#knn = KNeighborsClassifier()
knn.fit(X_train_normalized, y_train.values.ravel())
y_pred = knn.predict(X_test_normalized)
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the improved KNN model with default parameters on the test set␣
 ↪is', accuracy * 100, '%')
```

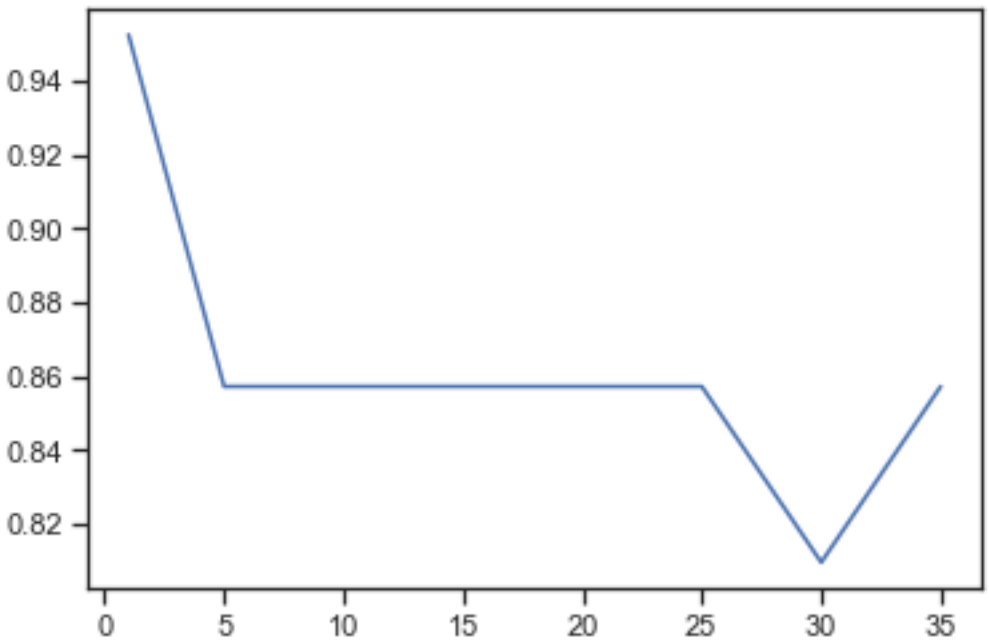
```
The accuracy of the improved KNN model with default parameters on the test set is 100.0 %
```

Upon trying different Normalization methods (Standard Scalar , MinMax Scalar), we observed better accuracy with MinMaxScalar. With Standard scalar normalization, we got an accuracy score of 95% on the test set, whereas we could achieve 100% accuracy score with MinMax scalar normalization. Here, the data is scaled to a fixed range of 0 to 1. This bounded range makes standard deviations smaller, which can suppress the effect of outliers.

Upon using different weighting schemes (default, manhattan, euclidean), we observed all produced same accuracy of 100%.

[32]:
```
# finding best parameter for the classifier
k_range = [1,5,10,15,20,25,30,35]
Scores = {}
Scores_list = []
best_k = 0
for k in k_range:
    knn =  KNeighborsClassifier(n_neighbors = k, metric='manhattan',␣
 ↪weights='distance')
    knn.fit(X_train_normalized, y_train.values.ravel())
    y_pred = knn.predict(X_val_normalized)
    Scores[k] = metrics.accuracy_score(y_val, y_pred)
    Scores_list.append(metrics.accuracy_score(y_val, y_pred))

plt.plot(k_range, Scores_list)

# finding the best k value
accuracy = max(Scores_list)
key_list = list(Scores.keys())
val_list = list(Scores.values())
position = val_list.index(accuracy)
best_k = key_list[position]

print('The best value of k is', best_k)
print('The accuracy of the improved KNN model on the validation set is', accuracy *␣
 ↪100, '%')
```

```
The best value of k is 1
The accuracy of the improved KNN model on the validation set is 95.23809523809523 %
```



We find that the model has the highest accuracy score for k=1. When k=1, probability estimation is based on a single sample i.e the nearest neighbor. This is very sensitive to distortions like noise, outliers etc. By using a higher value for k, the model becomes more robust against such distortions. Hence, selecting the k value with next best accuracy, we notice that all values 5,10,15,20,25 & 35 show the same accuracy score. Hence selecting k=5,as the number of neareest neighbours required to calculate the weighted value of the target parameters is less compares to the rest

[33]:
```
# using best k value, fitting the model on training set predicting the target on test␣
 ↪set
knn =  KNeighborsClassifier(n_neighbors = 5, metric='manhattan', weights='distance')
knn.fit(X_train_normalized, y_train.values.ravel())

# calculating accuracy
y_pred = knn.predict(X_test_normalized)
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the improved KNN model with best k on the test set is',␣
 ↪accuracy * 100, '%')

# calculating AUC
pred_prob = knn.predict_proba(X_test_normalized)
auc = roc_auc_score(y_test, pred_prob, average = 'macro', multi_class = 'ovr')
print('AUC:', auc)

# calculating f-score
f_score = f1_score(y_test, y_pred, average = 'weighted')
print('f-score:', f_score)
```

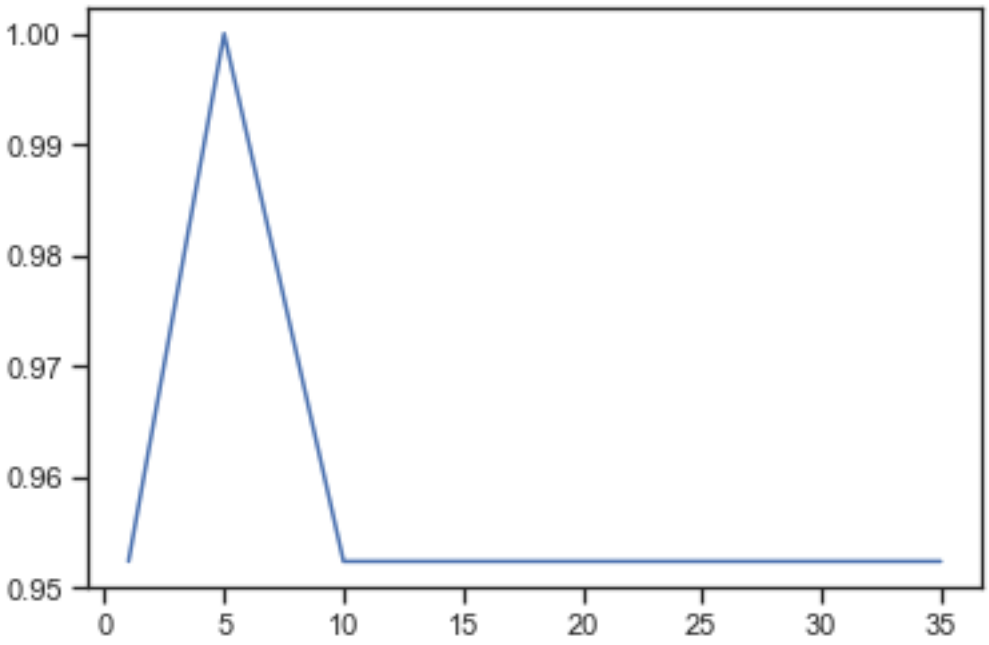
```
The accuracy of the improved KNN model with best k on the test set is 100.0 %
AUC: 1.0
f-score: 1.0
```

To verify our selection, we can try and plot the accuracy against k value for the model on the test set. We observe maximum accuracy at k=5 .

[34]:
```
# finding best parameter for the classifier
k_range = [1,5,10,15,20,25,30,35]
Scores = {}
Scores_list = []
best_k = 0
for k in k_range:
    knn =  KNeighborsClassifier(n_neighbors = k, metric='manhattan',␣
 ↪weights='distance')
    knn.fit(X_train_normalized, y_train.values.ravel())
    y_pred = knn.predict(X_test_normalized)
    Scores[k] = metrics.accuracy_score(y_test, y_pred)
    Scores_list.append(metrics.accuracy_score(y_test, y_pred))

plt.plot(k_range, Scores_list)

# finding the best k value
accuracy = max(Scores_list)
```

```
key_list = list(Scores.keys())
val_list = list(Scores.values())
position = val_list.index(accuracy)
best_k = key_list[position]

print('The best value of k is', best_k)
print('The accuracy of the improved KNN model on the test set is', accuracy * 100, '%')
```

```
The best value of k is 5
The accuracy of the improved KNN model on the test set is 100.0 %
```



Question 3: Analysis

[CM8]

1. Explain why you had to split the dataset into train, validation and test sets?

Answer:

In typical supervised machine learning, using the entire available dataset for training the model will lead to overfitting, i.e. the model simply repeats the labels of the sample data while predicting with perfect accuracy, whereas the model would fail in predicting the unseen/unknown data. This can be avoided by splitting the dataset into train, validation and test sets. The validation and test setsare used to provide an unbiased evaluation of the model fit on the training dataset.

2. Explain why you didn't evaluate directly on the test set and had to use a validation test when finding the best parameters for KNN?

Answer:

Validation set is used to provide an unbiased evaluation of a model fit on the training dataset to fine-tune the model hyperparameters. In the classifier,we used the training dataset to train the model, the validation