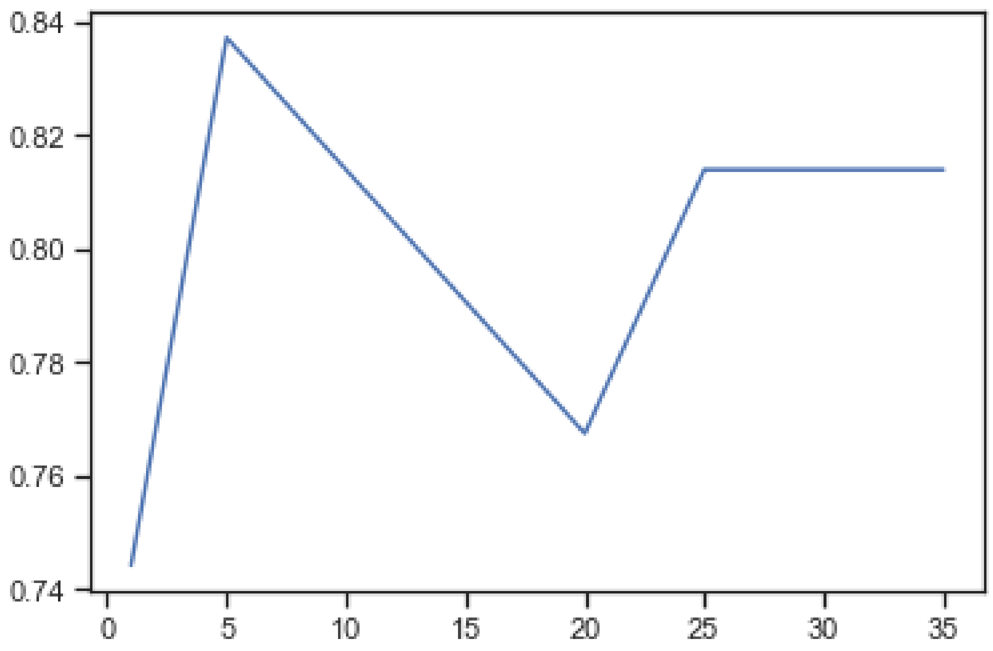


**We find that the model has the highest accuracy score for k=5**

## [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[:, 1], 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 73.80952380952381 %
AUC: 0.8901234567901234
f-score: 0.7433664667707222
```

[30]:
```
# Improved Model

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

# # 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)

# Robust Scalar Normalization
robustScaler = RobustScaler()
# Compute the mean and standard deviation based on the training data
robustScaler.fit(X_train)
# Scale the training data to be of mean 0 and of unit variance
X_train_normalized = robustScaler.transform(X_train)
# Scale the test data to be of mean 0 and of unit variance
X_test_normalized = robustScaler.transform(X_test)
# Scale the validation data to be of mean 0 and of unit variance
X_val_normalized = robustScaler.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='euclidean', 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
78.57142857142857 %
```

Upon trying different Normalization methods (Standard Scalar , MinMax Scalar, Robust Scalar), we observed better accuracy with Robust Scalar. it scales using median and quantiles consists of subtracting the median from all the observations and then dividing by the interquartile difference. In case of non-numeric features, it is better to use median for scaling. Robust scalar scales features using statistics that are robust to outliers.

Upon using different weighting schemes (default, manhattan, euclidean), we observed better model performance and accuracy with 'euclidean'.

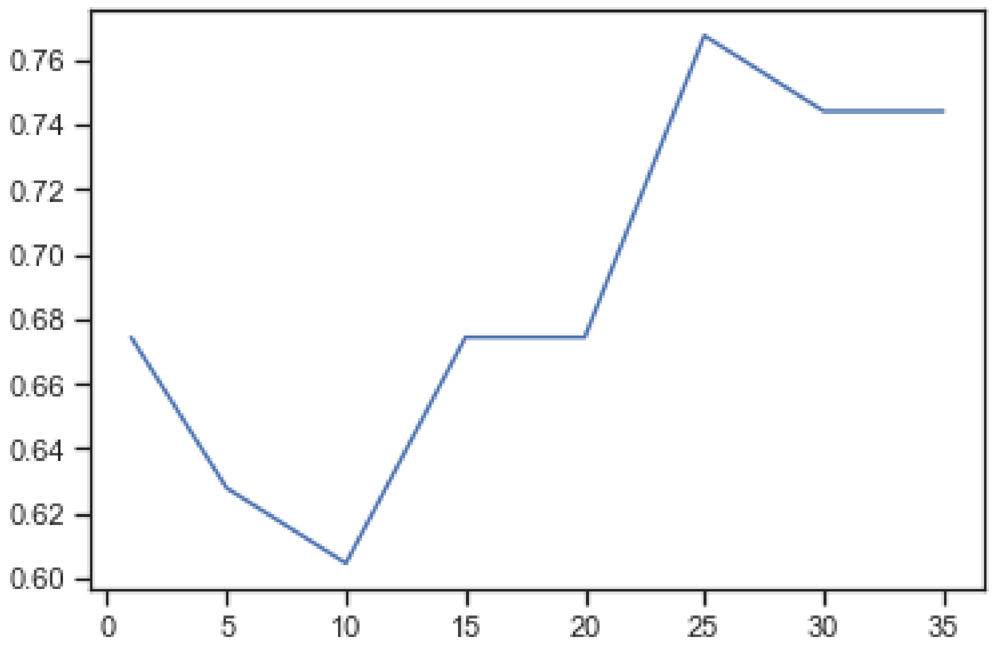
[31]:
```
# finding best parameter for the classifier
k_range = [1,5,10,15,20,25,30,35]
Scores = {}
Scores_list = []
best_k = 0
for k in k_range:
    knn =  KNeighborsClassifier(n_neighbors = k, metric='euclidean',␣
↪weights='distance')
    knn.fit(X_train_normalized, y_train.values.ravel())
    y_pred = 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 25
The accuracy of the improved KNN model on the validation set is 76.74418604651163 %
```



[32]:
```
# using best k value, fitting the model on training set predicting the target on test␣
↪set
knn =  KNeighborsClassifier(n_neighbors = 25, metric='euclidean', 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[:, 1], 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 83.33333333333334 %
AUC: 0.9382716049382716
f-score: 0.8367346938775511
```