Lab 3

PART-A: k-fold Cross Validation

1) Refer to Lab2 PART-B and Repeat the KNN Implementation (using built-in function only), but in this exercise after doing k-fold cross validation. Explore the 3 different types of k-fold cross validation listed below (code snippets) and observe the results. Does it vary significantly from what you obtained in Lab2?

   a) Partitioning and K-fold Cross Validation

   ```
   from sklearn.model_selection import KFold
   kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)
   # X is the feature set and y is the target
   for train_index, test_index in kf.split(X):
       print("Train:", train_index, "Validation:",test_index)
       X_train, X_test = X[train_index], X[test_index]
       y_train, y_test = y[train_index], y[test_index]
   ```

   b) Stratified k-fold cross validation:

   ```
   from sklearn.model_selection import StratifiedKFold
   skf = StratifiedKFold(n_splits=5, random_state=None)
   # X is the feature set and y is the target
   for train_index, test_index in skf.split(X,y):
      print("Train:", train_index, "Validation:", val_index)
      X_train, X_test = X[train_index], X[val_index]
      y_train, y_test = y[train_index], y[val_index]
   ```

   c) Simple k-fold cross validation with repetition.

   ```
   from sklearn.model_selection import RepeatedKFold
   rkf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)
   # X is the feature set and y is the target
   for train_index, test_index in rkf.split(X):
       print("Train:", train_index, "Validation:", val_index)
       X_train, X_test = X[train_index], X[val_index]
       y_train, y_test = y[train_index], y[val_index]
   ```

PART B: ROC AUC for KNN Models

2) Building on PART-A, generate multiple KNN models for varying values of k=1,3,5,7,9,11, 13,15.

   a) Plot their ROC Curves for various KNN models
   You can get probability estimates using the ***predict_proba()*** method of the ***KNeighborsClassifier*** in *sklearn*. This returns a numpy array with two columns for a binary classification, one each for the negative and positive class. For the ***roc_curve()*** function you want to use probability estimates of the positive class, so you can take all the rows of the second column with [:, 1] to only select the probability estimates of the positive class.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(X_train, y_train)
y_pred =classifier.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_pred[:, 1])
```

From the above, the ROC AUC score can be computed as

```
roc_auc = auc(fpr, tpr)
```

or directly using *roc_auc_score()* method of *sklearn*:

```
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test, y_pred [:, 1])
```

We can also plot the ROC curves for different models using matplotlib:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')

plt.legend(loc = 'lower right') )                    #Set Legend at lower rights
plt.plot([0, 1], [0, 1],'r--')                       #Set baseline model as diagonal
plt.xlim([0, 1])                                     # Set x axis limits
plt.ylim([0, 1])                                     # Set y axis limits
plt.ylabel('True Positive Rate')                     # y label
plt.xlabel('False Positive Rate')                    # x label

plt.plot(fpr, tpr, linestyle='--',  label = 'AUC = %0.2f' %roc_auc)   # plot roc curves
plt.show();
```

b) Interpret the results and find out which KNN model performs better?