Used Workbook 4 as the base. Just swapped out the base for this. Just swapped out the datasets.

In [1]:
```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import datasets
from sklearn.preprocessing import StandardScaler, scale
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cr
oss_val_score, cross_val_predict
from sklearn.metrics import confusion_matrix, precision_score, recall_
score, f1_score, roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve, classification_rep
ort
```

In [2]:
```
matches = pd.read_csv('Champion_Info1.csv', sep=",")
```

In [3]:
```
matches.head()
```

Out[3]:

	gameID	platformID	seasonID	Winner	C1	C1S1	C1S2	C2	C2S1	C2S2	...	C7S2
0	3591397591	NA1	13	1	201	4	14	102	12	4	...	7
1	3653371287	NA1	13	0	120	11	6	13	12	4	...	4
2	3650198359	NA1	13	1	102	11	4	122	4	6	...	4
3	3645932094	NA1	13	0	18	4	7	99	3	4	...	4
4	3645761570	NA1	13	0	102	11	4	82	4	12	...	4

5 rows × 34 columns

In [4]:
```
matches.shape
```

Out[4]: (965, 34)

In [5]:
```
#Data Matrix
X = pd.DataFrame(matches, columns=["C1","C1S1","C1S2","C2","C2S1","C2S
2","C3","C4S1","C4S2",
          "C4","C4S1","C4S2","C5","C5S1","C5S2","C6","C6S1","C6S2",
          "C7","C7S1","C7S2","C8","C8S1","C7S2","C9","C9S1","C9S2",
          "C10","C10S1","C10S2"])

#Target Vector
y=matches['Winner']

print(X.shape)
print(y.shape)
```

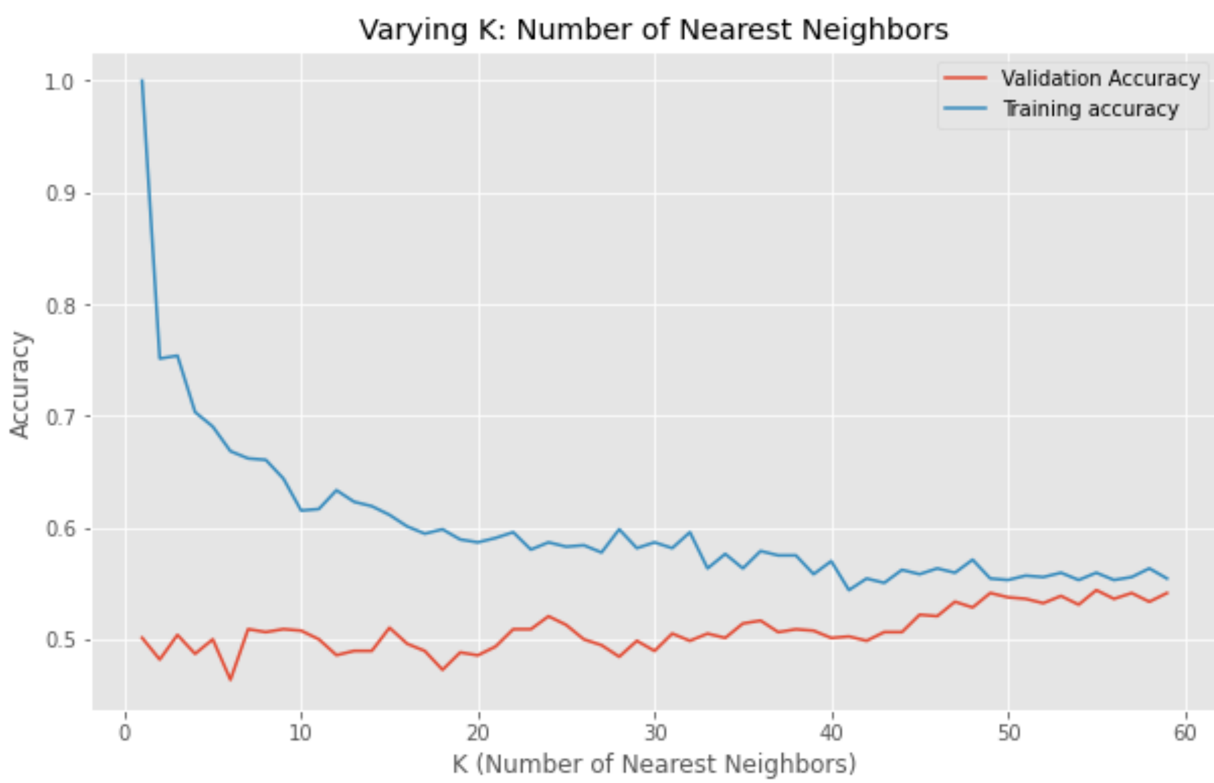
```
(965, 30)
(965,)
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
2, random_state=0)
print(y_train.shape)
```

```
(772,)
```

In [7]:
```
# Set the the range of K
neighbors = np.arange(1,60)

# Two arrays to store training and test accuracies
train_accuracy = np.empty(len(neighbors))
validation_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):

    # Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the model
    knn.fit(X_train, y_train)


    # The "score" function returns the mean accuracy on the given trai
n/test data and labels.
    # Note that "accuracy" may not be a good performance measure in a
skewed data set
    # Thus, we need to do hyperparameter tuning by using better perfor
mance measures (e.g., f1 score, presision, recall)

    # Compute training accuracy
    train_accuracy[i] = knn.score(X_train, y_train)

    # Compute validation accuracy using cross-validation

    scores = cross_val_score(knn, X_train, y_train, scoring='accuracy'
, cv=5)

    validation_accuracy[i] = scores.mean()
```

In [8]:
```
plt.style.use('ggplot')
fig = plt.figure(figsize=(10, 6))
plt.title('Varying K: Number of Nearest Neighbors')
plt.plot(neighbors, validation_accuracy, label='Validation Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('K (Number of Nearest Neighbors)')
plt.ylabel('Accuracy')
plt.show()



# Find the value of "K" that gives max validation accuracy
j = 0
max_val_accuracy = validation_accuracy[j]
max_k = 1

for i in neighbors:
    if(validation_accuracy[j] > max_val_accuracy):
        max_val_accuracy = validation_accuracy[j]
        max_k = i
    j +=1

print("Optimal K: ", max_k)
```




```
Optimal K:  55
```

In [9]:
```
%%time

warnings.filterwarnings('ignore')

# The param_grid tells Scikit-Learn to evaluate all combinations of th
e hyperparameter values
param_grid = {'n_neighbors': np.arange(1,50), 'p': [1, 2, 10, 50, 100,
500, 1000],
              'weights': ["uniform", "distance"]}

knn_clf = KNeighborsClassifier()

knn_cv = GridSearchCV(knn_clf, param_grid, scoring='f1', cv=5, verbose
=3, n_jobs=-1)
knn_cv.fit(X_train, y_train)


params_optimal_knn = knn_cv.best_params_

print("Best Score: %f" % knn_cv.best_score_)
print("Optimal Hyperparameter Values: ", params_optimal_knn)
print("\n")
```

```
Fitting 5 folds for each of 686 candidates, totalling 3430 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:   35.2s
[Parallel(n_jobs=-1)]: Done 212 tasks      | elapsed:   40.1s
[Parallel(n_jobs=-1)]: Done 852 tasks      | elapsed:   56.5s
[Parallel(n_jobs=-1)]: Done 1748 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 2900 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 3430 out of 3430 | elapsed:  2.1min fini
shed

Best Score: 0.662806
Optimal Hyperparameter Values:  {'n_neighbors': 49, 'p': 1, 'weights
': 'uniform'}


CPU times: user 3.93 s, sys: 342 ms, total: 4.27 s
Wall time: 2min 8s
```

In [10]:
```
# With the Mahalanobis distance metric only the brute force algorithm
works
#knn = KNeighborsClassifier(weights=?, algorithm='brute', n_neighbors=
?, metric = "mahalanobis", metric_params= {'V': ?})

# Minkowski distance metric based optimal model selected via hyperpara
meter tuning.
# The Minkowski distance based model (i.e., knn_cv) is already trained
with the optimal hyperparameter values.
# We can use the optimal model (knn_cv) for prediction.
# Or we can use the optimal hyperparameter values to train a new model
, as follows.

knn = KNeighborsClassifier(**params_optimal_knn)

knn.fit(X_train, y_train)

y_train_predicted = knn.predict(X_train)
print(y_train_predicted)

train_accuracy_knn = np.mean(y_train_predicted == y_train)
print("\nTraining Accuracy: ", train_accuracy_knn)
```

```
[0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0
1 1 1
 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 1 0
1 0 1
 0 1 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1
1 1 1
 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 0
1 0 1
 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1
 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 1
0 1 1
 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1
1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1
0 1 1
 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1
0 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
0 1 0
 0 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 0
1 1 1
 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1
0 1 0
 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1
1 1 0
 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 0
1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1
0 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1
1 1 1
 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0
1 1 0
 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 0 1 1]

Training Accuracy:  0.5764248704663213
```

In [11]:
```
# Scoring Parameter for Classification:
# https://scikit-learn.org/stable/modules/model_evaluation.html#scorin
g-parameter
# Note: For a skewed data set "accuracy" might not be a good choice fo
r scoring
scores = cross_val_score(knn, X_train, y_train, scoring='f1', cv=5)
print(scores)

print("F1 Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2
))
```

```
[0.66009852 0.70531401 0.63809524 0.65686275 0.65365854]
F1 Score: 0.66 (+/- 0.05)
```

In [12]:
```
y_train_pred = cross_val_predict(knn, X_train, y_train, cv=5)

confusion_matrix(y_train, y_train_pred)
```

Out[12]:
```
array([[ 84, 276],
       [ 71, 341]])
```

In [13]:
```
precision = precision_score(y_train, y_train_pred)
print("Precision = %f" % precision)

recall = recall_score(y_train, y_train_pred)
print("Recall = %f" % recall)


f1 = f1_score(y_train, y_train_pred)
print("F1 Score = %f" % f1)
```

```
Precision = 0.552674
Recall = 0.827670
F1 Score = 0.662779
```

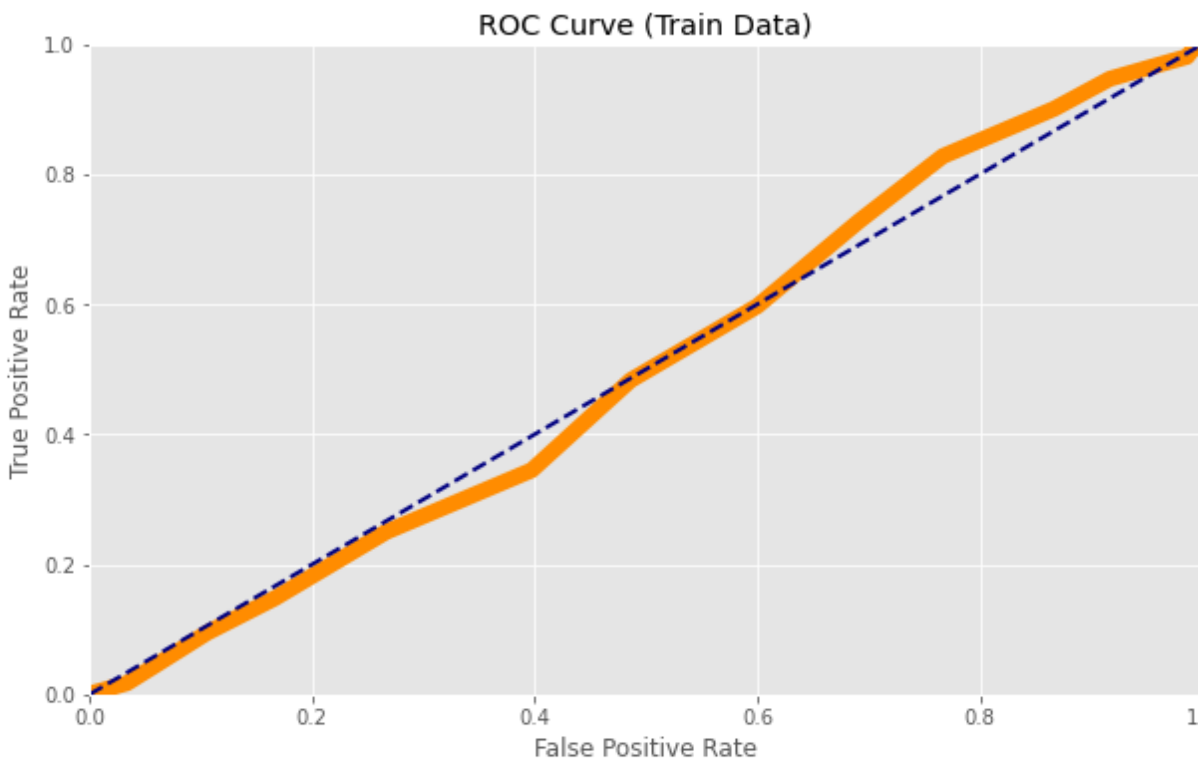
In [14]:
```
#The accuracy of the model
train_accuracy_knn = knn.score(X_train, y_train)
print("\nTest Accuracy: ", train_accuracy_knn)


# No. of Correct Predictions
y_train_predicted = knn.predict(X_train)
print("\nNo. of correct predictions (Test): %d/%d" % (np.sum(y_train_p
redicted == y_train), len(y_train)))

# Confusion Matrix
print("\nConfusion Matrix (Train Data):\n", confusion_matrix(y_train,
y_train_predicted))

precision = precision_score(y_train, y_train_predicted)
print("Train Precision = %f" % precision)

recall = recall_score(y_train, y_train_predicted)
print("Train Recall = %f" % recall)


f1 = f1_score(y_train, y_train_predicted)
print("Train F1 Score = %f" % f1)
print(classification_report(y_train,y_train_predicted))
```

```
Test Accuracy:  0.5764248704663213

No. of correct predictions (Test): 445/772

Confusion Matrix (Train Data):
 [[ 93 267]
 [ 60 352]]
Train Precision = 0.568659
Train Recall = 0.854369
Train F1 Score = 0.682832
              precision    recall  f1-score   support

           0       0.61      0.26      0.36       360
           1       0.57      0.85      0.68       412

    accuracy                           0.58       772
   macro avg       0.59      0.56      0.52       772
weighted avg       0.59      0.58      0.53       772
```

In [15]:
```
# Get the 2nd column of the matrix of predicted probabilities  for eac
h data point
#    The 2nd column stores the probabilities of the positive class
y_scores = cross_val_predict(knn, X_train, y_train, method="predict_pr
oba", cv=5)[:, 1]


fpr, tpr, thresholds = roc_curve(y_train, y_scores)

print("\nFPR FPR & TPR for Various Threshold Values:")

print("FPR: ", fpr)
print("TPR: ", tpr)
print("\nThresholds: ", thresholds)
```

```
FPR FPR & TPR for Various Threshold Values:
FPR:  [0.         0.00277778 0.00833333 0.03333333 0.05833333 0.1055
5556
 0.16666667 0.26666667 0.39722222 0.48611111 0.6        0.69166667
 0.76666667 0.86666667 0.91666667 0.96111111 0.98611111 0.99166667
 1.         1.        ]
TPR:  [0.         0.00242718 0.00485437 0.01699029 0.04368932 0.0946
6019
 0.14805825 0.25       0.34466019 0.48300971 0.59708738 0.72815534
 0.8276699  0.90048544 0.94660194 0.9684466  0.98058252 0.99271845
 0.99514563 1.        ]

Thresholds:  [1.73469388 0.73469388 0.71428571 0.69387755 0.67346939
0.65306122
 0.63265306 0.6122449  0.59183673 0.57142857 0.55102041 0.53061224
 0.51020408 0.48979592 0.46938776 0.44897959 0.42857143 0.40816327
 0.3877551  0.34693878]
```

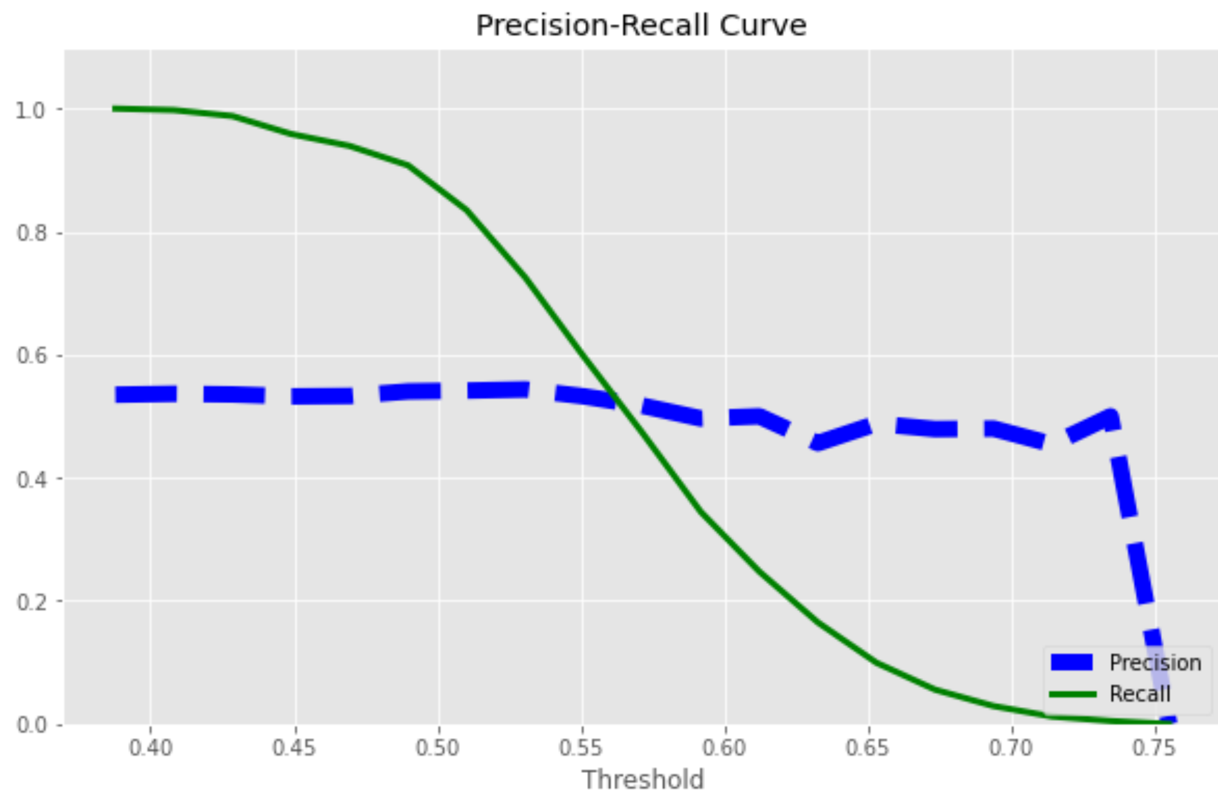
In [16]:
```
plt.style.use('ggplot')

fig = plt.figure(figsize=(10, 6))
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, color='darkorange', linewidth=8, label=label)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.axis([0, 1, 0, 1])
    plt.title('ROC Curve (Train Data)')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

plot_roc_curve(fpr, tpr)
plt.show()
```



In [17]:
```
#Area under ROC curve
roc_auc_score(y_train,y_scores)
```

Out[17]: 0.5011023462783172

In [18]:
```
plt.style.use('ggplot')

# Get the 2nd column of the matrix of predicted probabilities for each
data point
#    The 2nd column stores the probalities of the positive class
y_scores = cross_val_predict(knn, X_train, y_train, method="predict_pr
oba", cv=3)[:, 1]

precisions, recalls, thresholds = precision_recall_curve(y_train, y_sc
ores)


fig = plt.figure(figsize=(10, 6))
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds
):
    plt.plot(thresholds, precisions[:-1], "b--",  linewidth=8, label="
Precision")
    plt.plot(thresholds, recalls[:-1], "g-",  linewidth=3, label="Reca
ll")
    plt.xlabel("Threshold")
    plt.legend(loc="lower right")
    plt.title('Precision-Recall Curve')
    #plt.xlim([0, 1])
    plt.ylim([0, 1.1])

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()


threshold_optimal = -1
for i in range(len(precisions)):
    if(precisions[i] == recalls[i]):
        threshold_optimal = thresholds[i]

print("Optimal Threshold: ", threshold_optimal)
```




```
Optimal Threshold:  0.7551020408163265
```

In [19]:

```
print("Performance Measures Based on the Default Threshold:\n")


y_train_pred = cross_val_predict(knn, X_train, y_train, cv=5)

# Precision, Recall, F1 Score and Confusion Matrix for the Default Thr
eshold 0.5
precision_train = precision_score(y_train, y_train_pred)
print("Precision (Default Threshold 0.5) = %f" % precision_train)

recall_train = recall_score(y_train, y_train_pred)
print("Recall (Default Threshold 0.5) = %f" % recall_train)


f1_train = f1_score(y_train, y_train_pred)
print("F1 Score (Default Threshold 0.5) = %f" % f1_train)

print("Confusion Matrix (Default Threshold 0.5)\n", confusion_matrix(y
_train, y_train_pred))


print("\n---------------------------------------------------------\n")
print("Performance Measures Based on the Optimal Threshold (from Preci
sion-Recall Curve):")
```

```
# Precision, Recall, F1 Score and Confusion Matrix for different thres
hold

t = threshold_optimal # optimal threshold from precision-recall curve

# Compute predictions based on new t by using the following method:
#  - Get the probability of the positive class from the 2nd column [:,
1]
#  - If that probability is greater than or equal to t, then the test
data belongs to the positive class
y_train_predicted_new = (cross_val_predict(knn, X_train, y_train, meth
od="predict_proba", cv=3)[:,1] > t).astype(int)


precision = precision_score(y_train, y_train_predicted_new)
print("\nPrecision (Threshold %.2f) = %f" % (t, precision))

recall = recall_score(y_train, y_train_predicted_new)
print("Recall (Threshold %.2f) = %f" % (t, recall))


f1 = f1_score(y_train, y_train_predicted_new)
print("F1 Score = (Threshold %.2f) = %f" % (t, f1))


print("Confusion Matrix (Threshold %.2f)" %  t)
print(confusion_matrix(y_train, y_train_predicted_new))
```

```
Performance Measures Based on the Default Threshold:

Precision (Default Threshold 0.5) = 0.552674
Recall (Default Threshold 0.5) = 0.827670
F1 Score (Default Threshold 0.5) = 0.662779
Confusion Matrix (Default Threshold 0.5)
 [[ 84 276]
 [ 71 341]]

-------------------------------------------------------

Performance Measures Based on the Optimal Threshold (from Precision-
Recall Curve):

Precision (Threshold 0.76) = 0.000000
Recall (Threshold 0.76) = 0.000000
F1 Score = (Threshold 0.76) = 0.000000
Confusion Matrix (Threshold 0.76)
[[360   0]
 [412   0]]
```

In [20]:

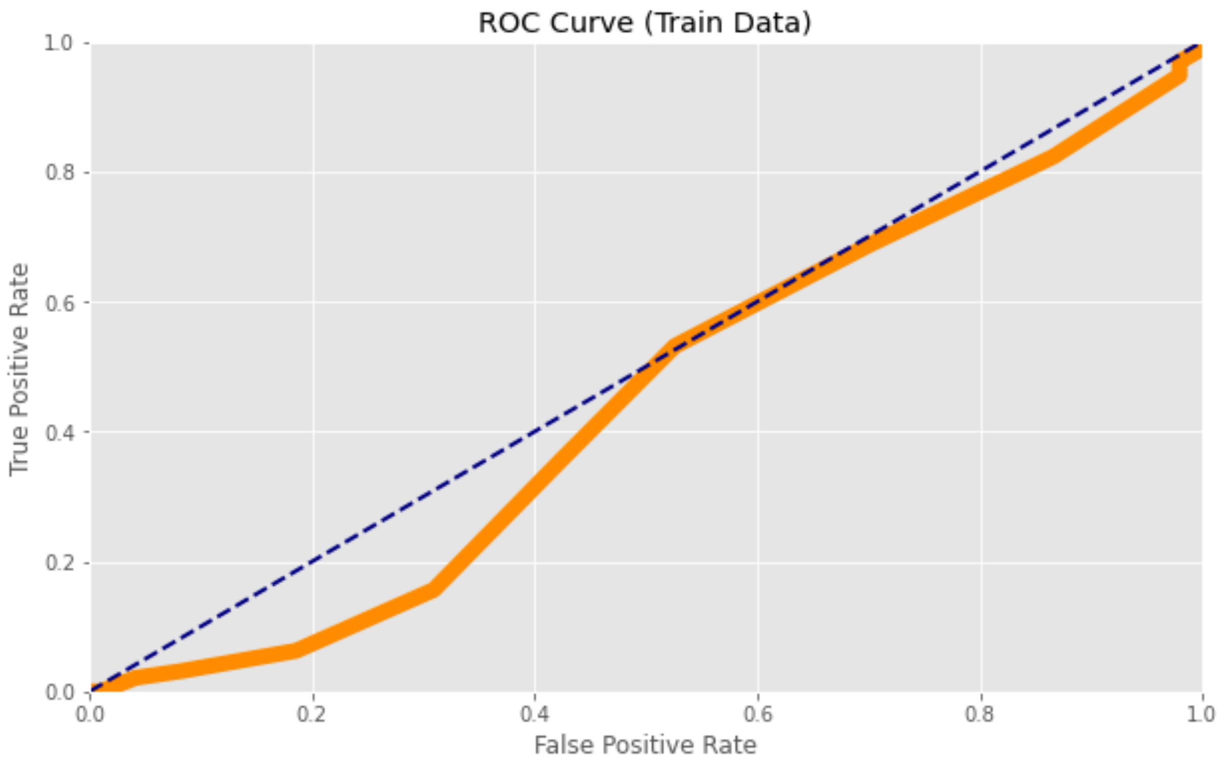
```
#The accuracy of the model
test_accuracy_knn = knn.score(X_test, y_test)
print("\nTest Accuracy: ", test_accuracy_knn)


# No. of Correct Predictions
y_test_predicted = knn.predict(X_test)
print("\nNo. of correct predictions (Test): %d/%d" % (np.sum(y_test_pr
edicted == y_test), len(y_test)))

# Confusion Matrix
print("\nConfusion Matrix (Test Data):\n", confusion_matrix(y_test, y_
test_predicted))

precision = precision_score(y_test, y_test_predicted)
print("Test Precision = %f" % precision)

recall = recall_score(y_test, y_test_predicted)
print("Test Recall = %f" % recall)


f1 = f1_score(y_test, y_test_predicted)
print("Test F1 Score = %f" % f1)
```

```
Test Accuracy:  0.48704663212435234

No. of correct predictions (Test): 94/193

Confusion Matrix (Test Data):
 [[18 79]
 [20 76]]
Test Precision = 0.490323
Test Recall = 0.791667
Test F1 Score = 0.605578
```

In [21]:
```
# Get the 2nd column of the matrix of predicted probabilities for each
data point
#    The 2nd column stores the probalities of the positive class
y_scores_test = cross_val_predict(knn, X_test, y_test, method="predict
_proba", cv=3)[:, 1]

fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_scores_test)


print("\nFPR FPR & TPR for Various Threshold Values:")
print("FPR: ", fpr_test)
print("TPR: ", tpr_test)
print("\nThresholds: ", thresholds_test)
```

```
FPR FPR & TPR for Various Threshold Values:
FPR:  [0.         0.01030928 0.04123711 0.08247423 0.18556701 0.3092
7835
 0.42268041 0.5257732  0.70103093 0.86597938 0.97938144 0.97938144
 1.         1.        ]
TPR:  [0.         0.         0.02083333 0.03125    0.0625     0.1562
5
 0.35416667 0.53125    0.6875     0.82291667 0.94791667 0.96875
 0.98958333 1.        ]

Thresholds:  [1.6122449  0.6122449  0.59183673 0.57142857 0.55102041
0.53061224
 0.51020408 0.48979592 0.46938776 0.44897959 0.42857143 0.40816327
 0.3877551  0.34693878]
```

In [22]:
```
plt.style.use('ggplot')

fig = plt.figure(figsize=(10, 6))
plot_roc_curve(fpr_test, tpr_test)
plt.show()
```



In [23]:
```
# Area under ROC curve
roc_auc_score(y_test,y_scores_test)
```

Out[23]: 0.4463058419243986

In [24]:
```
pd.crosstab(y_test, y_test_predicted, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[24]:

Predicted True	0	1	All
0	18	79	97
1	20	76	96
All	38	155	193

In [25]:
```
#The accuracy of the model
test_accuracy_knn = knn.score(X_test, y_test)
print("\nTest Accuracy: ", test_accuracy_knn)


# No. of Correct Predictions
y_test_predicted = knn.predict(X_test)
print("\nNo. of correct predictions (Test): %d/%d" % (np.sum(y_test_pr
edicted == y_test), len(y_test)))

# Confusion Matrix
print("\nConfusion Matrix (Test Data):\n", confusion_matrix(y_test, y_
test_predicted))

precision = precision_score(y_test, y_test_predicted)
print("Test Precision = %f" % precision)

recall = recall_score(y_test, y_test_predicted)
print("Test Recall = %f" % recall)


f1 = f1_score(y_test, y_test_predicted)
print("Test F1 Score = %f" % f1)
print(classification_report(y_test,y_test_predicted))
```

```
Test Accuracy:  0.48704663212435234

No. of correct predictions (Test): 94/193

Confusion Matrix (Test Data):
 [[18 79]
 [20 76]]
Test Precision = 0.490323
Test Recall = 0.791667
Test F1 Score = 0.605578
              precision    recall  f1-score   support

           0       0.47      0.19      0.27        97
           1       0.49      0.79      0.61        96

    accuracy                           0.49       193
   macro avg       0.48      0.49      0.44       193
weighted avg       0.48      0.49      0.44       193
```

In [ ]: