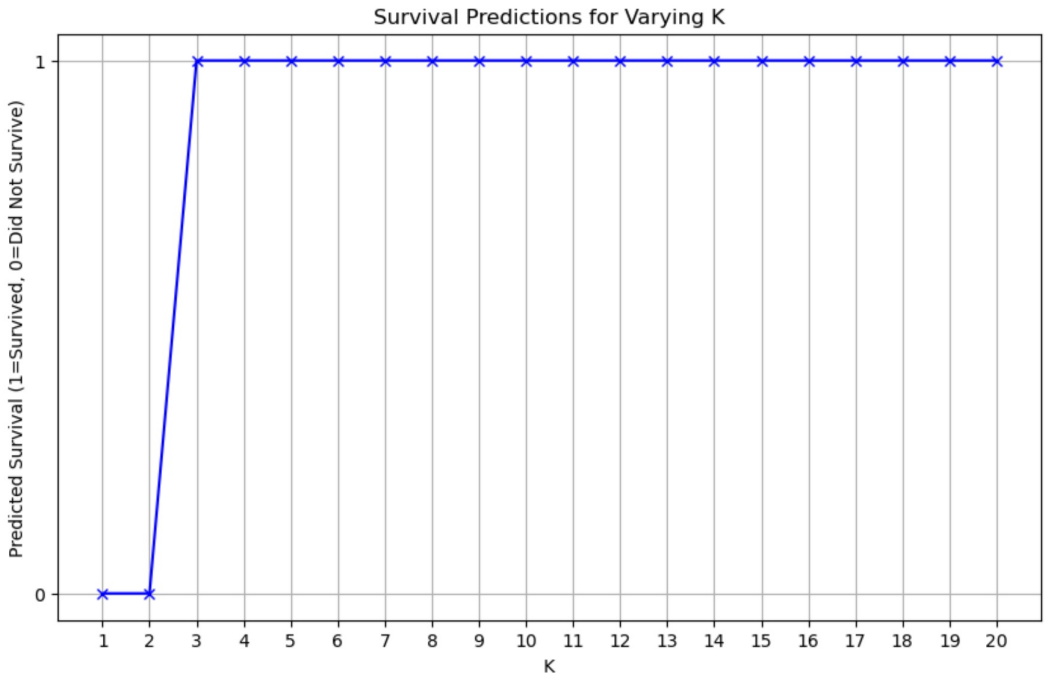
HW5

March 10, 2024

[116]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt

datasetPath = 'titanic_data.csv'
titanic_data = pd.read_csv(dataPath)

features = titanic_data.drop('Survived', axis=1)
labels = titanic_data['Survived']
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.2, random_state=42)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

def knnPredict_range(X_train_scaled, y_train, test_instance_scaled, max_k):
    allPredictions = []

    # Calculate Euclidean distances and make predictions for each k
    for k in range(1, max_k + 1):
        distances = np.sqrt(((X_train_scaled - test_instance_scaled) ** 2).
 ↪sum(axis=1))
        nearest_neighbors = y_train.iloc[distances.nsmallest(k).index]
        vote = nearest_neighbors.mode()[0]
        allPredictions.append(vote)

    return allPredictions

x = [2, 1, 24, 1, 0, 45]
x_scaled = scaler.transform([x])
N = 20
```

```
predictions = knnPredict_range(X_train_scaled, y_train, x_scaled[0], N)

plt.figure(figsize=(10, 6))
plt.plot(range(1, N + 1), predictions, marker='x', linestyle='-', color='blue')
plt.title('Survival Predictions for Varying K')
plt.xlabel('K')
plt.ylabel('Predicted Survival (1=Survived, 0=Did Not Survive)')
plt.xticks(range(1, N + 1))  # Fix the x-axis ticks
plt.yticks([0, 1])
plt.grid(True)
plt.show()
```

```
/home/flash/anaconda3/envs/760/lib/python3.11/site-packages/sklearn/base.py:464:
UserWarning: X does not have valid feature names, but MinMaxScaler was fitted
with feature names
  warnings.warn(
```



[120]:
```
titanic_dataset = pd.read_csv(datasetPath)

feature_columns = titanic_dataset.drop('Survived', axis=1)
target_column = titanic_dataset['Survived']
X_training, X_testing, y_training, y_testing =␣
 ↪train_test_split(feature_columns, target_column, test_size=0.2,␣
 ↪random_state=42)
```

```
class GNaiveBayes:
    def train(self, features, target):
        self.class_labels = np.unique(target)
        self.classFeatures = {}
        for label in self.class_labels:
            features_of_class = features[target == label]
            self.classFeatures[label] = {
                'means': features_of_class.mean(axis=0),
                'vars': features_of_class.var(axis=0),
                'priProb': features_of_class.shape[0] / features.shape[0]
            }

    def gProb(self, mean, var, featureVal):
        exponent_term = np.exp(-((featureVal - mean) ** 2) / (2 * var))
        return exponent_term / np.sqrt(2 * np.pi * var)

    def computePost(self, feature_set):
        classPosts = []
        for label in self.class_labels:
            logPri = np.log(self.classFeatures[label]['priProb'])
            class_likelihoods = self.gProb(self.classFeatures[label]['means'],␣
 ↪self.classFeatures[label]['vars'], feature_set)
            total_likelihood = np.sum(np.log(class_likelihoods))
            post = logPri + total_likelihood
            classPosts.append(post)
        return self.class_labels[np.argmax(classPosts)]

    def classify(self, features_set):
        predicted_labels = [self.computePost(features) for features in␣
 ↪features_set.to_numpy()]
        return np.array(predicted_labels)

gnbModel = GNaiveBayes()

gnbModel.train(X_training, y_training)

yPredictions = gnbModel.classify(X_testing)

model_accuracy = (y_testing == yPredictions).mean()
print(f'Model Accuracy: {model_accuracy}')
```

```
Model Accuracy: 0.7359550561797753
```

[121]:
```
featureVector = np.array([2, 1, 24, 1, 0, 45])
prediction = gnbModel.computePost(featureVector)
if prediction == 1:
```

```
    print("I survive the Titanic.")
else:
    print("I dont survive Titanic.")
```

```
I survive the Titanic.
```

[122]:
```
def gPdf(value, mean, var):
    return (1 / np.sqrt(2 * np.pi * var)) * np.exp(- (value - mean) ** 2 / (2 *␣
 ↪var))

fMeans = [38, 168, 6.75]
fVars = [2, 50, 0.125]
evidence = [42, 180, 5.5]

likelihoodsF = [
    gPdf(evidence[0], fMeans[0], fVars[0]),
    gPdf(evidence[1], fMeans[1], fVars[1]),
    gPdf(evidence[2], fMeans[2], fVars[2])]

priF = 1/3

postF = priF
for likelihood in likelihoodsF:
    postF *= likelihood

postM = 0.0006471370319999999

postM, postF, postM > postF
```

[122]:
```
(0.0006471370319999999, 5.0147748922449326e-08, True)
```

[ ]:
```

```