Supervised Learning and K Nearest Neighbors Exercises

Learning Objectives:

- Explain supervised learning and how it can be applied to regression and classification problems
- Apply K-Nearest Neighbor (KNN) algorithm for classification
- Apply Intel® Extension for Scikit-learn* to leverage underlying compute capabilities of hardware

scikit-learn*

Frameworks provide structure that Data Scientists use to build code. Frameworks are more than just libraries, because in addition to callable code, frameworks influence how code is written.

A main virtue of using an optimized framework is that code runs faster. Code that runs faster is just generally more convenient but when we begin looking at applied data science and AI models, we can see more material benefits. Here you will see how optimization, particularly hyperparameter optimization can benefit more than just speed.

These exercises will demonstrate how to apply **the Intel® Extension for Scikit-learn*,** a seamless way to speed up your Scikit-learn application. The acceleration is achieved through the use of the Intel® oneAPI Data Analytics Library (oneDAL). Patching is the term used to extend scikit-learn with Intel optimizations and makes it a well-suited machine learning framework for dealing with real-life problems.

To get optimized versions of many Scikit-learn algorithms using a patch() approach consisting of adding these lines of code Prior to importing sklearn:

- **from sklearnex import patch_sklearn**
- **patch_sklearn()**

This exercise relies on installation of Intel® Extension for Scikit-learn*

If you have not already done so, follow the instructions from Week 1 for instructions

```
from __future__ import print_function
import os
data_path = ['../data']

from sklearnex import patch_sklearn
patch_sklearn()  # patch should be called PRIOR to importing Scikit-learn algorithms

from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
```

Question 1

- Begin by importing the data. Examine the columns and data.
- Notice that the data contains a state, area code, and phone number. Do you think these are good features to use when building a machine learning model? Why or why not?

We will not be using them, so they can be dropped from the data.

```
df = pd.read_csv('resources/Orange_Telecom_Churn_Data.csv')

print("Columns in the dataset:")
print(df.columns)
```

```
Columns in the dataset:
Index(['state', 'account_length', 'area_code', 'phone_number', 'intl_plan',
       'voice_mail_plan', 'number_vmail_messages', 'total_day_minutes',
       'total_day_calls', 'total_day_charge', 'total_eve_minutes',
       'total_eve_calls', 'total_eve_charge', 'total_night_minutes',
       'total_night_calls', 'total_night_charge', 'total_intl_minutes',
       'total_intl_calls', 'total_intl_charge',
```

```
       'number_customer_service_calls', 'churned'],
      dtype='object')
```

```
print("\nFirst few rows of the dataset:")
print(df.head())
```

```
First few rows of the dataset:
  state  account_length  area_code phone_number intl_plan voice_mail_plan  \
0    KS             128        415     382-4657        no             yes
1    OH             107        415     371-7191        no             yes
2    NJ             137        415     358-1921        no              no
3    OH              84        408     375-9999       yes              no
4    OK              75        415     330-6626       yes              no

   number_vmail_messages  total_day_minutes  total_day_calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   total_day_charge  ...  total_eve_calls  total_eve_charge  \
0             45.07  ...               99             16.78
1             27.47  ...              103             16.62
2             41.38  ...              110             10.30
3             50.90  ...               88              5.26
4             28.34  ...              122             12.61

   total_night_minutes  total_night_calls  total_night_charge  \
0                244.7                 91                11.01
1                254.4                103                11.45
2                162.6                104                 7.32
3                196.9                 89                 8.86
4                186.9                121                 8.41

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   number_customer_service_calls  churned
0                              1    False
1                              1    False
2                              0    False
3                              2    False
4                              3    False

[5 rows x 21 columns]
```

```
columns_to_drop = ['state', 'area_code', 'phone_number']
df = df.drop(columns=columns_to_drop)
```

```
print("\nDataset after dropping 'state', 'area_code', and 'phone_number' columns:")
print(df.head())
```

```
Dataset after dropping 'state', 'area_code', and 'phone_number' columns:
   account_length intl_plan voice_mail_plan  number_vmail_messages  \
0             128        no             yes                     25
1             107        no             yes                     26
2             137        no              no                      0
3              84       yes              no                      0
4              75       yes              no                      0

   total_day_minutes  total_day_calls  total_day_charge  total_eve_minutes  \
0              265.1              110             45.07              197.4
1              161.6              123             27.47              195.5
2              243.4              114             41.38              121.2
3              299.4               71             50.90               61.9
4              166.7              113             28.34              148.3

   total_eve_calls  total_eve_charge  total_night_minutes  total_night_calls  \
0               99             16.78                244.7                 91
1              103             16.62                254.4                103
2              110             10.30                162.6                104
3               88              5.26                196.9                 89
4              122             12.61                186.9                121

   total_night_charge  total_intl_minutes  total_intl_calls  \
0               11.01                10.0                 3
1               11.45                13.7                 3
2                7.32                12.2                 5
```

```
3               8.86                6.6                 7
4               8.41               10.1                 3

   total_intl_charge  number_customer_service_calls  churned
0               2.70                              1    False
1               3.70                              1    False
2               3.29                              0    False
3               1.78                              2    False
4               2.73                              3    False
```

Question 2

- Notice that some of the columns are categorical data and some are floats. These features will need to be numerically encoded using one of the methods from the lecture.
- Finally, remember from the lecture that K-nearest neighbors requires scaled data. Scale the data using one of the scaling methods discussed in the lecture.

```
label_binarizer = LabelBinarizer()

categorical_columns = ['intl_plan', 'voice_mail_plan', 'churned']
for column in categorical_columns:
    df[column] = label_binarizer.fit_transform(df[column])

print("\nDataset after encoding categorical features:")
print(df.head())
```

```
Dataset after encoding categorical features:
   account_length  intl_plan  voice_mail_plan  number_vmail_messages  \
0             128          0                1                     25
1             107          0                1                     26
2             137          0                0                      0
3              84          1                0                      0
4              75          1                0                      0

   total_day_minutes  total_day_calls  total_day_charge  total_eve_minutes  \
0              265.1              110             45.07              197.4
1              161.6              123             27.47              195.5
2              243.4              114             41.38              121.2
3              299.4               71             50.90               61.9
4              166.7              113             28.34              148.3

   total_eve_calls  total_eve_charge  total_night_minutes  total_night_calls  \
0               99             16.78                244.7                 91
1              103             16.62                254.4                103
2              110             10.30                162.6                104
3               88              5.26                196.9                 89
4              122             12.61                186.9                121

   total_night_charge  total_intl_minutes  total_intl_calls  \
0               11.01                10.0                 3
1               11.45                13.7                 3
2                7.32                12.2                 5
3                8.86                 6.6                 7
4                8.41                10.1                 3

   total_intl_charge  number_customer_service_calls  churned
0               2.70                              1        0
1               3.70                              1        0
2               3.29                              0        0
3               1.78                              2        0
4               2.73                              3        0
```

```
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(df.drop(columns=['churned']))

scaled_df = pd.DataFrame(scaled_features, columns=df.columns[:-1])  # Exclude the target variable
scaled_df['churned'] = df['churned']

print("\nDataset after scaling features:")
print(scaled_df.head())
```

```
Dataset after scaling features:
   account_length  intl_plan  voice_mail_plan  number_vmail_messages  \
0        0.524793        0.0              1.0               0.480769
1        0.438017        0.0              1.0               0.500000
2        0.561983        0.0              0.0               0.000000
3        0.342975        1.0              0.0               0.000000
4        0.305785        1.0              0.0               0.000000
```

```
   total_day_minutes  total_day_calls  total_day_charge  total_eve_minutes  \
0           0.754196         0.666667          0.754183           0.542755
1           0.459744         0.745455          0.459672           0.537531
2           0.692461         0.690909          0.692436           0.333242
3           0.851778         0.430303          0.851740           0.170195
4           0.474253         0.684848          0.474230           0.407754

   total_eve_calls  total_eve_charge  total_night_minutes  total_night_calls  \
0         0.582353          0.542866             0.619494           0.520000
1         0.605882          0.537690             0.644051           0.588571
2         0.647059          0.333225             0.411646           0.594286
3         0.517647          0.170171             0.498481           0.508571
4         0.717647          0.407959             0.473165           0.691429

   total_night_charge  total_intl_minutes  total_intl_calls  \
0            0.619584               0.500              0.15
1            0.644344               0.685              0.15
2            0.411930               0.610              0.25
3            0.498593               0.330              0.35
4            0.473270               0.505              0.15

   total_intl_charge  number_customer_service_calls  churned
0           0.500000                       0.111111        0
1           0.685185                       0.111111        0
2           0.609259                       0.000000        0
3           0.329630                       0.222222        0
4           0.505556                       0.333333        0
```

Question 3

- Separate the feature columns (everything except `churned`) from the label (`churned`). This will create two tables.
- Fit a K-nearest neighbors model with a value of `k=3` to this data and predict the outcome on the same data.

```
X = scaled_df.drop(columns=['churned'])
y = scaled_df['churned']


knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X, y)

y_pred = knn_model.predict(X)

predicted_df = pd.DataFrame({'Actual Churned': y, 'Predicted Churned': y_pred})
print("\nPredicted outcomes on the same data:")
print(predicted_df.head())
```

```
Predicted outcomes on the same data:
   Actual Churned  Predicted Churned
0               0                  0
1               0                  0
2               0                  0
3               0                  0
4               0                  0
```

Question 4

Ways to measure error haven't been discussed in class yet, but accuracy is an easy one to understand--it is simply the percent of labels that were correctly predicted (either true or false).

- Write a function to calculate accuracy using the actual and predicted labels.
- Using the function, calculate the accuracy of this K-nearest neighbors model on the data.

```
def calculate_accuracy(actual, predicted):
    """
    Calculate accuracy given the actual and predicted labels.

    Parameters:
    - actual: Actual labels
    - predicted: Predicted labels

    Returns:
    - accuracy: Accuracy score
    """
    correct_predictions = np.sum(actual == predicted)
    total_predictions = len(actual)
    accuracy = correct_predictions / total_predictions
    return accuracy
```

```
accuracy = calculate_accuracy(y, y_pred)

print(f"\nAccuracy of the K-nearest neighbors model: {accuracy:.2%}")
```

```
Accuracy of the K-nearest neighbors model: 94.22%
```

Question 5

- Fit the K-nearest neighbors model again with `n_neighbors=3` but this time use distance for the weights. Calculate the accuracy using the function you created above.
- Fit another K-nearest neighbors model. This time use uniform weights but set the power parameter for the Minkowski distance metric to be 1 (`p=1`) i.e. Manhattan Distance.

When weighted distances are used for part 1 of this question, a value of 1.0 should be returned for the accuracy. Why do you think this is? *Hint:* we are predicting on the data and with KNN the model *is* the data. We will learn how to avoid this pitfall in the next lecture.

```
knn_distance_model = KNeighborsClassifier(n_neighbors=3, weights='distance')
knn_distance_model.fit(X, y)
y_pred_distance = knn_distance_model.predict(X)

accuracy_distance = calculate_accuracy(y, y_pred_distance)
print(f"\nAccuracy with distance weights: {accuracy_distance:.2%}")
```

```
Accuracy with distance weights: 96.48%
```

```
knn_uniform_manhattan_model = KNeighborsClassifier(n_neighbors=3, weights='uniform', p=1)
knn_uniform_manhattan_model.fit(X, y)
y_pred_uniform_manhattan = knn_uniform_manhattan_model.predict(X)

accuracy_uniform_manhattan = calculate_accuracy(y, y_pred_uniform_manhattan)
print(f"Accuracy with uniform weights and Manhattan distance: {accuracy_uniform_manhattan:.2%}")
```

```
Accuracy with uniform weights and Manhattan distance: 94.56%
```

Question 6

- Fit a K-nearest neighbors model using values of `k` (`n_neighbors`) ranging from 1 to 20. Use uniform weights (the default). The coefficient for the Minkowski distance (`p`) can be set to either 1 or 2--just be consistent. Store the accuracy and the value of `k` used from each of these fits in a list or dictionary.
- Plot (or view the table of) the `accuracy` vs `k`. What do you notice happens when `k=1`? Why do you think this is? *Hint:* it's for the same reason discussed above.

```
import matplotlib.pyplot as plt

accuracy_list = []
k_values = list(range(1, 21))

for k in k_values:
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X, y)
    y_pred_k = knn_model.predict(X)
    accuracy_k = calculate_accuracy(y, y_pred_k)
    accuracy_list.append(accuracy_k)


# Plot accuracy vs k
plt.plot(k_values, accuracy_list, marker='o')
plt.title('Accuracy vs k in K-near')
plt.xlabel('k (Neighbors)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 # Plot accuracy vs k
----> 2 plt.plot(k_values, accuracy_list, marker='o')
      3 plt.title('Accuracy vs k in K-near')
      4 plt.xlabel('k (Neighbors)')

NameError: name 'plt' is not defined
```