April 14, 2023

1 Lab 3: Training Decision Tree & KNN Classifiers

[1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
pd.options.mode.chained_assignment = None


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In this Lab session, you will implement the following steps:

1. Load the Airbnb "listings" data set
2. Convert categorical features to one-hot encoded values
3. Split the data into training and test sets
4. Fit a Decision Tree classifier and evaluate the accuracy

- Plot the accuracy of the DT model as a function of hyperparameter max depth

5. Fit a KNN classifier and evaluate the accuracy

- Plot the accuracy of the KNN model as a function of hyperparameter k

1.1 Part 1. Load the Dataset

We will work with a preprocessed version of the Airbnb NYC "listings" data set.

Task: load the data set into a Pandas DataFrame variable named `df`:

[2]:
```
# Do not remove or edit the line below:
filename = os.path.join(os.getcwd(), "data", "airbnb.csv.gz")

# YOUR CODE HERE
### Solution:
```

```
df = pd.read_csv(filename, header=0)
```

[3]:
```
df.shape
```

[3]:
```
(28022, 44)
```

[4]:
```
df.head(10)
```

[4]:
```
   host_response_rate  host_acceptance_rate  host_is_superhost  \
0            0.800000              0.170000              False
1            0.090000              0.690000              False
2            1.000000              0.250000              False
3            1.000000              1.000000              False
4            0.890731              0.768297              False
5            1.000000              1.000000               True
6            1.000000              1.000000              False
7            1.000000              1.000000              False
8            1.000000              0.000000              False
9            1.000000              0.990000               True

   host_listings_count  host_total_listings_count  host_has_profile_pic  \
0                  8.0                        8.0                  True
1                  1.0                        1.0                  True
2                  1.0                        1.0                  True
3                  1.0                        1.0                  True
4                  1.0                        1.0                  True
5                  3.0                        3.0                  True
6                  1.0                        1.0                  True
7                  3.0                        3.0                  True
8                  2.0                        2.0                  True
9                  1.0                        1.0                  True

   host_identity_verified neighbourhood_group_cleansed        room_type  \
0                    True                    Manhattan  Entire home/apt
1                    True                     Brooklyn  Entire home/apt
2                    True                     Brooklyn  Entire home/apt
3                   False                    Manhattan     Private room
4                    True                    Manhattan     Private room
5                    True                     Brooklyn     Private room
6                    True                     Brooklyn  Entire home/apt
7                    True                    Manhattan     Private room
8                    True                     Brooklyn     Private room
9                    True                     Brooklyn  Entire home/apt

   accommodates  ...  review_scores_communication  review_scores_location  \
0             1  ...                         4.79                    4.86
1             3  ...                         4.80                    4.71
2             4  ...                         5.00                    4.50
3             2  ...                         4.42                    4.87
```

```
4          1  ...                         4.95                     4.94
5          2  ...                         4.82                     4.87
6          3  ...                         4.80                     4.67
7          1  ...                         4.95                     4.84
8          1  ...                         5.00                     5.00
9          4  ...                         4.91                     4.93

   review_scores_value instant_bookable  calculated_host_listings_count  \
0                 4.41            False                               3
1                 4.64            False                               1
2                 5.00            False                               1
3                 4.36            False                               1
4                 4.92            False                               1
5                 4.73            False                               3
6                 4.57             True                               1
7                 4.84             True                               1
8                 5.00            False                               2
9                 4.78             True                               2

   calculated_host_listings_count_entire_homes  \
0                                            3
1                                            1
2                                            1
3                                            0
4                                            0
5                                            1
6                                            1
7                                            0
8                                            0
9                                            1

   calculated_host_listings_count_private_rooms  \
0                                             0
1                                             0
2                                             0
3                                             1
4                                             1
5                                             2
6                                             0
7                                             1
8                                             2
9                                             1

   calculated_host_listings_count_shared_rooms  reviews_per_month  \
0                                            0               0.33
1                                            0               4.86
2                                            0               0.02
```

```
3                                            0           3.68
4                                            0           0.87
5                                            0           1.48
6                                            0           1.24
7                                            0           1.82
8                                            0           0.07
9                                            0           3.05

   n_host_verifications
0                     9
1                     6
2                     3
3                     4
4                     7
5                     7
6                     7
7                     5
8                     5
9                     8

[10 rows x 44 columns]
```

[5]:
```
df.columns
```

[5]:
```
Index(['host_response_rate', 'host_acceptance_rate', 'host_is_superhost',
       'host_listings_count', 'host_total_listings_count',
       'host_has_profile_pic', 'host_identity_verified',
       'neighbourhood_group_cleansed', 'room_type', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'amenities', 'price', 'minimum_nights',
       'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_nights',
       'minimum_maximum_nights', 'maximum_maximum_nights',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'number_of_reviews', 'number_of_reviews_ltm',
       'number_of_reviews_l30d', 'review_scores_rating',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
       'n_host_verifications'],
      dtype='object')
```

1.2 Part 2. One-Hot Encode Categorical Values

Transform the string-valued categorical features into numerical boolean values using one-hot encoding.

1.2.1 a. Find the Columns Containing String Values

First, let us identify all features that need to be one-hot encoded:

[6]:
```
df.dtypes
```

[6]:
```
host_response_rate                         float64
host_acceptance_rate                       float64
host_is_superhost                             bool
host_listings_count                        float64
host_total_listings_count                  float64
host_has_profile_pic                          bool
host_identity_verified                        bool
neighbourhood_group_cleansed                object
room_type                                   object
accommodates                                 int64
bathrooms                                  float64
bedrooms                                   float64
beds                                       float64
amenities                                   object
price                                      float64
minimum_nights                               int64
maximum_nights                               int64
minimum_minimum_nights                     float64
maximum_minimum_nights                     float64
minimum_maximum_nights                     float64
maximum_maximum_nights                     float64
minimum_nights_avg_ntm                     float64
maximum_nights_avg_ntm                     float64
has_availability                              bool
availability_30                              int64
availability_60                              int64
availability_90                              int64
availability_365                             int64
number_of_reviews                            int64
number_of_reviews_ltm                        int64
number_of_reviews_l30d                       int64
review_scores_rating                       float64
review_scores_cleanliness                  float64
review_scores_checkin                      float64
review_scores_communication                float64
review_scores_location                     float64
review_scores_value                        float64
instant_bookable                              bool
```

```
calculated_host_listings_count                          int64
calculated_host_listings_count_entire_homes             int64
calculated_host_listings_count_private_rooms            int64
calculated_host_listings_count_shared_rooms             int64
reviews_per_month                                     float64
n_host_verifications                                    int64
dtype: object
```

Task: add all of the column names of variables of type 'object' to a list named `to_encode`

[7]:
```python
# YOUR CODE HERE

### Solution:
to_encode = list(df.select_dtypes(include=['object']).columns)
to_encode
```

[7]:
```
['neighbourhood_group_cleansed', 'room_type', 'amenities']
```

Let's take a closer look at the candidates for one-hot encoding

[8]:
```python
df[to_encode].nunique()
```

[8]:
```
neighbourhood_group_cleansed        5
room_type                           4
amenities                       25020
dtype: int64
```

Notice that one column stands out as containing two many values for us to attempt to transform. For this exercise, the best choice is to simply remove this column. Of course, this means losing potentially useful information. In a real-life situation, you would want to retain all of the information in a column, or you could selectively keep information in.

In the code cell below, drop this column from Dataframe `df` and from the `to_encode` list.

[9]:
```python
# YOUR SOLUTION HERE

# solution
df.drop(columns = ['amenities'], inplace=True)
to_encode.remove('amenities')
```

1.2.2 b. One-Hot Encode all Unique Values

All of the other columns in `to_encode` have reasonably small numbers of unique values, so we are going to simply one-hot encode every unique value of those columns.

Task: complete the code below to create one-hot encoded columns Tip: Use the sklearn `OneHotEncoder` class

[10]:
```python
from sklearn.preprocessing import OneHotEncoder

# Create the encoder:
#encoder = # YOUR CODE HERE

### Solution:
```

```
encoder = OneHotEncoder(handle_unknown="error", sparse=False)

# Apply the encoder:
#df_enc = # YOUR CODE HERE

### Solution:
df_enc = pd.DataFrame(encoder.fit_transform(df[to_encode]))

# Reinstate the original column names:
#df_enc.columns = # YOUR CODE HERE

### Solution:
df_enc.columns = encoder.get_feature_names(to_encode)
```

[11]:
```
df_enc.head()
```

[11]:
```
   neighbourhood_group_cleansed_Bronx  neighbourhood_group_cleansed_Brooklyn  \
0                                 0.0                                    0.0
1                                 0.0                                    1.0
2                                 0.0                                    1.0
3                                 0.0                                    0.0
4                                 0.0                                    0.0

   neighbourhood_group_cleansed_Manhattan  \
0                                     1.0
1                                     0.0
2                                     0.0
3                                     1.0
4                                     1.0

   neighbourhood_group_cleansed_Queens  \
0                                  0.0
1                                  0.0
2                                  0.0
3                                  0.0
4                                  0.0

   neighbourhood_group_cleansed_Staten Island  room_type_Entire home/apt  \
0                                         0.0                        1.0
1                                         0.0                        1.0
2                                         0.0                        1.0
3                                         0.0                        0.0
4                                         0.0                        0.0

   room_type_Hotel room  room_type_Private room  room_type_Shared room
0                   0.0                     0.0                    0.0
1                   0.0                     0.0                    0.0
2                   0.0                     0.0                    0.0
```

```
3                    0.0                      1.0                     0.0
4                    0.0                      1.0                     0.0
```

Task: You can now remove the original columns that we have just transformed from DataFrame `df`.

[12]:
```
# YOUR CODE HERE

### Solution:
df.drop(columns = to_encode, inplace=True)
```

[13]:
```
df.head()
```

[13]:
```
   host_response_rate  host_acceptance_rate  host_is_superhost  \
0            0.800000              0.170000              False
1            0.090000              0.690000              False
2            1.000000              0.250000              False
3            1.000000              1.000000              False
4            0.890731              0.768297              False

   host_listings_count  host_total_listings_count  host_has_profile_pic  \
0                  8.0                        8.0                  True
1                  1.0                        1.0                  True
2                  1.0                        1.0                  True
3                  1.0                        1.0                  True
4                  1.0                        1.0                  True

   host_identity_verified  accommodates  bathrooms  bedrooms  ...  \
0                    True             1        1.0  1.323567  ...
1                    True             3        1.0  1.000000  ...
2                    True             4        1.5  2.000000  ...
3                   False             2        1.0  1.000000  ...
4                    True             1        1.0  1.000000  ...

   review_scores_communication  review_scores_location  review_scores_value  \
0                         4.79                    4.86                 4.41
1                         4.80                    4.71                 4.64
2                         5.00                    4.50                 5.00
3                         4.42                    4.87                 4.36
4                         4.95                    4.94                 4.92

   instant_bookable  calculated_host_listings_count  \
0             False                               3
1             False                               1
2             False                               1
3             False                               1
4             False                               1

   calculated_host_listings_count_entire_homes  \
```

```
0                                            3
1                                            1
2                                            1
3                                            0
4                                            0

   calculated_host_listings_count_private_rooms  \
0                                             0
1                                             0
2                                             0
3                                             1
4                                             1

   calculated_host_listings_count_shared_rooms  reviews_per_month  \
0                                            0               0.33
1                                            0               4.86
2                                            0               0.02
3                                            0               3.68
4                                            0               0.87

   n_host_verifications
0                     9
1                     6
2                     3
3                     4
4                     7

[5 rows x 41 columns]
```

Task: You can now join the transformed categorical features contained in `df_enc` with DataFrame `df`

[14]:
```python
# YOUR CODE HERE

### Solution:
df = df.join(df_enc)
```

Glance at the resulting column names:

[15]:
```python
df.columns
```

[15]:
```
Index(['host_response_rate', 'host_acceptance_rate', 'host_is_superhost',
       'host_listings_count', 'host_total_listings_count',
       'host_has_profile_pic', 'host_identity_verified', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'price', 'minimum_nights',
       'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_nights',
       'minimum_maximum_nights', 'maximum_maximum_nights',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'number_of_reviews', 'number_of_reviews_ltm',
```

```
       'number_of_reviews_l30d', 'review_scores_rating',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
       'n_host_verifications', 'neighbourhood_group_cleansed_Bronx',
       'neighbourhood_group_cleansed_Brooklyn',
       'neighbourhood_group_cleansed_Manhattan',
       'neighbourhood_group_cleansed_Queens',
       'neighbourhood_group_cleansed_Staten Island',
       'room_type_Entire home/apt', 'room_type_Hotel room',
       'room_type_Private room', 'room_type_Shared room'],
      dtype='object')
```

Check for missing values.

[16]:
```
# YOUR CODE HERE

# Solution
df.isnull().values.any()
```

[16]:
```
False
```

1.3 Part 3. Create Training and Test Data Sets

1.3.1 a. Create Labeled Examples

Task: Choose columns from our data set to create labeled examples.

In the airbnb dataset, we will choose column host_is_superhost to be the label. The remaining columns will be the features.

Obtain the features from DataFrame df and assign to X. Obtain the label from DataFrame df and assign to Y

[17]:
```
# YOUR CODE HERE

### Solution:
y = df['host_is_superhost']
X = df.drop(columns = 'host_is_superhost', axis=1)
```

[18]:
```
print("Number of examples: " + str(X.shape[0]))
print("\nNumber of Features:" + str(X.shape[1]))
print(str(list(X.columns)))
```

```
Number of examples: 28022

Number of Features:49
['host_response_rate', 'host_acceptance_rate', 'host_listings_count',
```

```
'host_total_listings_count', 'host_has_profile_pic', 'host_identity_verified',
'accommodates', 'bathrooms', 'bedrooms', 'beds', 'price', 'minimum_nights',
'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_nights',
'minimum_maximum_nights', 'maximum_maximum_nights', 'minimum_nights_avg_ntm',
'maximum_nights_avg_ntm', 'has_availability', 'availability_30',
'availability_60', 'availability_90', 'availability_365', 'number_of_reviews',
'number_of_reviews_ltm', 'number_of_reviews_l30d', 'review_scores_rating',
'review_scores_cleanliness', 'review_scores_checkin',
'review_scores_communication', 'review_scores_location', 'review_scores_value',
'instant_bookable', 'calculated_host_listings_count',
'calculated_host_listings_count_entire_homes',
'calculated_host_listings_count_private_rooms',
'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
'n_host_verifications', 'neighbourhood_group_cleansed_Bronx',
'neighbourhood_group_cleansed_Brooklyn',
'neighbourhood_group_cleansed_Manhattan', 'neighbourhood_group_cleansed_Queens',
'neighbourhood_group_cleansed_Staten Island', 'room_type_Entire home/apt',
'room_type_Hotel room', 'room_type_Private room', 'room_type_Shared room']
```

1.3.2 b. Split Examples into Training and Test Sets

Task: In the code cell below create training and test sets out of the labeled examples using Scikit-learn's `train_test_split()` function.

Specify: * A test set that is one third (.33) of the size of the data set. * A seed value of '123'.

[19]:
```
# YOUR CODE HERE

### Solution:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=123)
```

Check that the dimensions of the training and test datasets are what you expected

[20]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(18774, 49)
(9248, 49)
```

1.4 Part 4. Implement a Decision Tree Classifier

The code cell below contains a shell of a function named `train_test_DT()`. This function should train a Decision Tree classifier on the training data, test the resulting model on the test data, and compute and return the accuracy score of the resulting predicted class labels on the test data. Remember to use `DecisionTreeClassifier()` to create a model object.

Task: Complete the function to make it work.

[21]:
```
def train_test_DT(X_train, X_test, y_train, y_test, leaf, depth,␣
↪crit='entropy'):
    '''
```

```
    Fit a Decision Tree classifier to the training data X_train, y_train.
    Return the accuracy of resulting predictions on the test set.
    Parameters:
        leaf := The minimum number of samples required to be at a leaf node
        depth := The maximum depth of the tree
        crit := The function to be used to measure the quality of a split.␣
↪Default: gini.
    '''

      # YOUR CODE HERE

    # 1. Create the  DecisionTreeClassifier model object below and assign to␣
↪variable 'model'

    ### SOLUTION
    model = DecisionTreeClassifier(criterion = crit, max_depth = depth,␣
↪min_samples_leaf = leaf)


    # 2. Fit the model to the training data below
     # YOUR CODE HERE
    # SOLUTION
    model.fit(X_train, y_train)


    # 3. Make predictions on the test data below and assign the result to the␣
↪variable 'class_label_predictions'
     # YOUR CODE HERE

    #SOLUTION
    class_label_predictions = model.predict(X_test)


    # 4. Compute the accuracy here and save the result to the variable␣
↪'acc_score'
     # YOUR CODE HERE

     #SOLUTION
    acc_score = accuracy_score(y_test, class_label_predictions)



    return acc_score
```

Visualization The cell below contains a function that you will use to compare the accuracy results of training multiple models with different hyperparameter values.

Function `visualize_accuracy()` accepts two arguments: 1. a list of hyperparamter values 2. a list of accuracy scores

Both lists must be of the same size.

[22]:
```
# Do not remove or edit the code below

def visualize_accuracy(hyperparam_range, acc):

    fig = plt.figure()
    ax = fig.add_subplot(111)
    p = sns.lineplot(x=hyperparam_range, y=acc, marker='o', label = 'Full␣
 ↪training set')

    plt.title('Test set accuracy of the model predictions, for ' + ','.
 ↪join([str(h) for h in hyperparam_range]))
    ax.set_xlabel('Hyperparameter value')
    ax.set_ylabel('Accuracy')
    plt.show()
```

Train on Different Values of Hyperparameter Max Depth Task:

Complete function `train_multiple_trees()` in the code cell below. The function should train multiple decision trees and return a list of accuracy scores.

The function will:

1. accept list `max_depth_range` and `leaf` as parameters; list `max_depth_range` will contain multiple values for hyperparameter max depth.

2. loop over list `max_depth_range` and at each iteration:

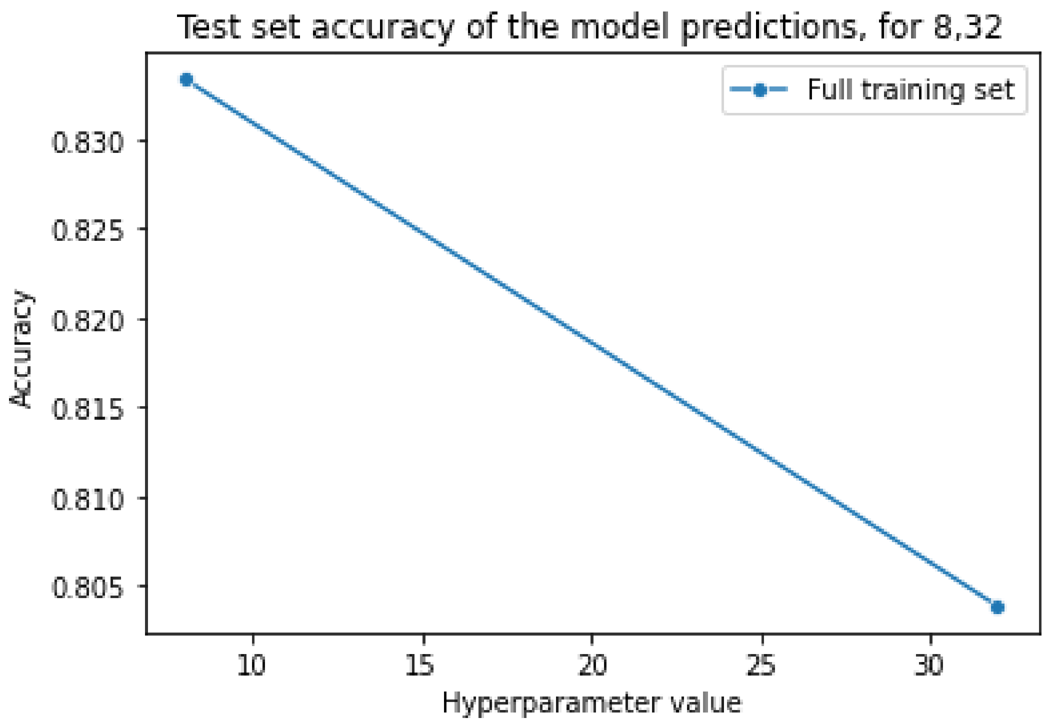
    a. index into list `max_depth_range` to obtain a value for max depth
    b. call `train_test_DT` with the training and test set, the value of max depth, and the value of `leaf`
    c. print the resulting accuracy score
    d. append the accuracy score to list `accuracy_list`

[23]:
```
def train_multiple_trees(max_depth_range, leaf):

    accuracy_list = []

    # YOUR CODE HERE
     # SOLUTION
    for md in max_depth_range:
        score = train_test_DT(X_train, X_test, y_train, y_test, leaf, md)
        print('Max Depth=' + str(md) + ', accuracy score: ' + str(score))
        accuracy_list.append(float(score))


    return accuracy_list
```

The code cell below tests function `train_multiple_trees()` and calls function `visualize_accuracy()` to visualize the results.

[24]:
```
max_depth_range = [8, 32]
leaf = 1

acc = train_multiple_trees(max_depth_range, leaf)

visualize_accuracy(max_depth_range, acc)
```

```
Max Depth=8, accuracy score: 0.8333693771626297
Max Depth=32, accuracy score: 0.8038494809688581
```



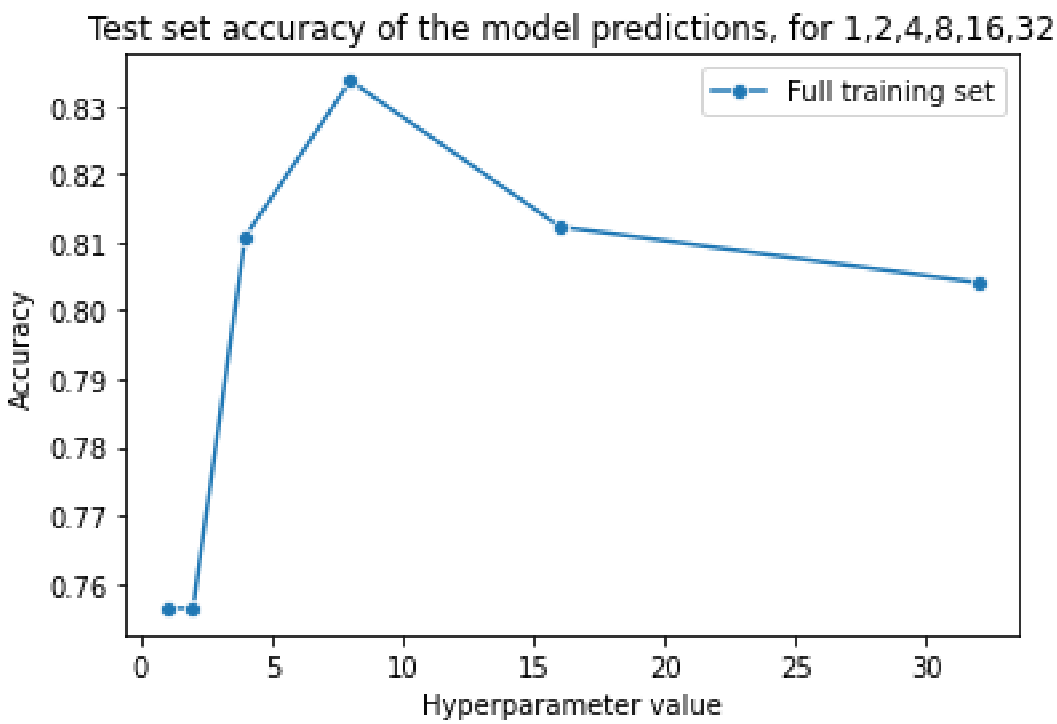
Analysis: Is this graph conclusive for determining a good value of max depth?
Task: Let's train on more values for max depth.
In the code cell below:

1. call `train_multiple_trees()` with arguments `max_depth_range` and `leaf`
2. call `visualize_accuracy()` with arguments `max_depth_range` and `acc`

[25]:
```
max_depth_range = [2**i for i in range(6)]
leaf = 1

# Solution
```

```
acc = train_multiple_trees(max_depth_range, leaf)# call train_multiple_trees()␣
↪here

# call visualize_accuracy() here
#solution
visualize_accuracy(max_depth_range, acc)
```

```
Max Depth=1, accuracy score: 0.7563797577854672
Max Depth=2, accuracy score: 0.7563797577854672
Max Depth=4, accuracy score: 0.810878027681661
Max Depth=8, accuracy score: 0.8336937716262975
Max Depth=16, accuracy score: 0.8122837370242214
Max Depth=32, accuracy score: 0.8040657439446367
```



Analysis: Analyze this graph. Keep in mind that this is the performance on the test set, and pay attention to the scale of the y-axis. Answer the following questions in the cell below. How would you go about choosing the best model based on this plot? Is it conclusive? What other hyperparameters of interest would you want to vary to make sure you are finding the best model fit?

1.5 Part 5. Implement a KNN Classifier

Note: In this section you will train KNN classifiers using the same training and test data.

The code cell below contains a shell of a function named `train_test_knn()`. This function should train a KNN classifier on the training data, test the resulting model on the test data, and compute and return the accuracy score of the resulting predicted class labels on the test data.

Remember to use `KNeighborsClassifier()` to create a model object and call the method with one parameter: `n_neighbors = k`.

Task: Complete the function to make it work.

[26]:
```python
def train_test_knn(X_train, X_test, y_train, y_test, k):
    '''
    Fit a k Nearest Neighbors classifier to the training data X_train, y_train.
    Return the accuracy of resulting predictions on the test data.
    '''

    # YOUR CODE HERE

    # 1. Create the  KNeighborsClassifier model object below and assign to␣
↪variable 'model'
    ### BEGIN SOLUTION
    model = KNeighborsClassifier(n_neighbors = k)
    ### END SOLUTION

    # 2. Fit the model to the training data below
    ### BEGIN SOLUTION
    model.fit(X_train, y_train)
    ### END SOLUTION

    # 3. Make predictions on the test data below and assign the result to the␣
↪variable 'class_label_predictions'
    ### BEGIN SOLUTION
    class_label_predictions = model.predict(X_test)
    ### END SOLUTION

    # 4. Compute the accuracy here and save the result to the variable␣
↪'acc_score'
    ### BEGIN SOLUTION
    acc_score = accuracy_score(y_test, class_label_predictions)
    ### END SOLUTION



    return acc_score
```

Train on Different Values of Hyperparameter K Task:

Just as you did above, complete function `train_multiple_knns()` in the code cell below. The function should train multiple KNN models and return a list of accuracy scores.

The function will:

1. accept list `k_range` as a parameter; this list will contain multiple values for hyperparameter

k

2. loop over list `k_range` and at each iteration:

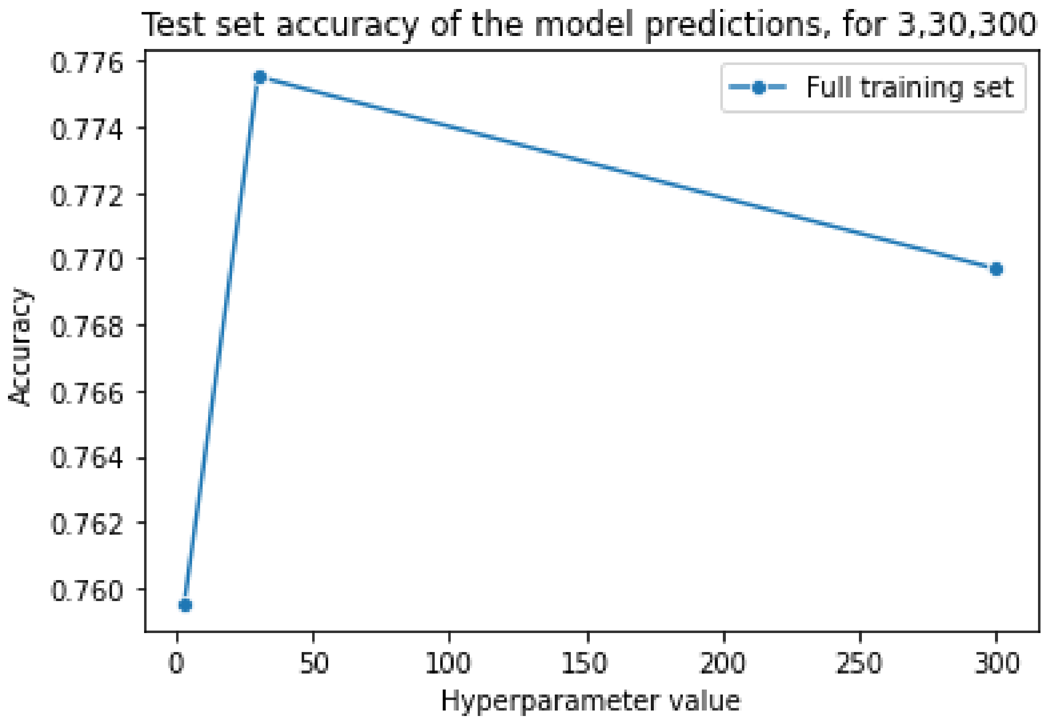
   a. index into list `k_range` to obtain a value for k
   b. call `train_test_knn` with the training and test set, and the value of k
   c. print the resulting accuracy score
   d. append the accuracy score to list `accuracy_list`

[27]:
```python
def train_multiple_knns(k_range):

    accuracy_list = []

    # YOUR CODE HERE

     # SOLUTION
    for k in k_range:
        score = train_test_knn(X_train, X_test, y_train, y_test, k)
        print('k=' + str(k) + ', accuracy score: ' + str(score))
        accuracy_list.append(float(score))


    return accuracy_list
```

The code cell below uses your `train_multiple_knn()` function to train 3 KNN models, specifying three values for k: $3, 30$, and 300. It calls function `visualize_accuracy()` to visualize the results. Note: this make take a second.

[28]:
```python
k_range = [3, 30, 300]
acc = train_multiple_knns(k_range)

visualize_accuracy(k_range, acc)
```

```
k=3, accuracy score: 0.759515570934256
k=30, accuracy score: 0.7755190311418685
k=300, accuracy score: 0.7696799307958477
```



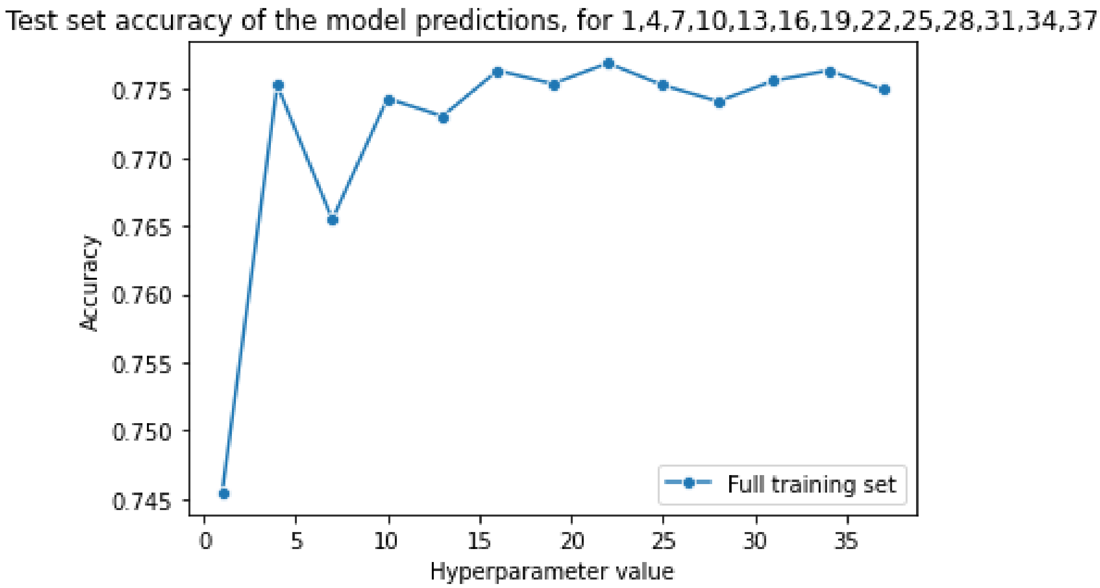
Task: Let's train on more than values for k
In the code cell below:

1. call `train_multiple_knns()` with argument `k_range`
2. call `visualize_accuracy()` with arguments `k_range` and the resulting accuracy list obtained from `train_multiple_knns()`

[29]:
```
k_range = np.arange(1, 40, step = 3)

# YOUR CODE HERE

acc = train_multiple_knns(k_range)

visualize_accuracy(k_range, acc)
```

```
k=1, accuracy score: 0.7454584775086506
k=4, accuracy score: 0.77530276816609
k=7, accuracy score: 0.7654628027681661
k=10, accuracy score: 0.7743295847750865
k=13, accuracy score: 0.7730320069204152
k=16, accuracy score: 0.7763840830449827
k=19, accuracy score: 0.7754108996539792
k=22, accuracy score: 0.776924740484429
k=25, accuracy score: 0.77530276816609
```

```
k=28, accuracy score: 0.7741133217993079
k=31, accuracy score: 0.7756271626297578
k=34, accuracy score: 0.7763840830449827
k=37, accuracy score: 0.7749783737024222
```



Analysis: Compare the performance of the KNN model relative to the Decision Tree model, with various hyperparameter values and record your findings in the cell below.