btt004_week3_lab-solution

May 1, 2024

1 Lab 3: ML Life Cycle: Modeling

[1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

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
```

Decision Trees (DTs) and KNNs have many similarities. They are models that are fairly simple and intuitive to understand, can be used to solve both classification and regression problems, and are non-parametric models, meaning that they don't assume a particular relationship between the features and the label prior to training. However, KNNs and DTs each have their own advantages and disadvantages. In addition, one model may be better suited than the other for a particular machine learning problem based on multiple factors, such as the size and quality of the data, the problem-type and the hyperparameter configuration. For example, KNNs require feature values to be scaled, whereas DTs do not. DTs are also able to handle noisy data better than KNNs.

Often times, it is beneficial to train multiple models on your training data to find the one that performs the best on the test data.

In this lab, you will continue practicing the modeling phase of the machine learning life cycle. You will train Decision Trees and KNN models to solve a classification problem. You will experiment training multiple variations of the models with different hyperparameter values to find the best performing model for your predictive problem. You will complete the following tasks:

1. Build your DataFrame and define your ML problem:
    - Load the Airbnb "listings" data set
    - Define the label - what are you predicting?
    - Identify the features
2. Prepare your data:
    - Perform feature engineering by converting categorical features to one-hot encoded values

3. Create labeled examples from the data set
4. Split the data into training and test data sets
5. Train multiple decision trees and evaluate their performances:
    - Fit Decision Tree classifiers to the training data using different hyperparameter values per classifier
    - Evaluate the accuracy of the models' predictions
    - Plot the accuracy of each DT model as a function of hyperparameter max depth
6. Train multiple KNN classifiers and evaluate their performances:
    - Fit KNN classifiers to the training data using different hyperparameter values per classifier
    - Evaluate the accuracy of the models' predictions
    - Plot the accuracy of each KNN model as a function of hyperparameter k
7. Analysis:

- Determine which is the best performing model
- Experiment with other factors that can help determine the best performing model

1.1 Part 1. Build Your DataFrame and Define Your ML Problem

Load a Data Set and Save it as a Pandas DataFrame We will work with a new preprocessed, slimmed down version of the Airbnb NYC "listings" data set. This version is almost ready for modeling, with missing values and outliers taken care of. Also note that unstructured fields have been removed.

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

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

[3]:
```
# YOUR CODE HERE
### Solution:
df = pd.read_csv(filename)
```

Inspect the Data Task: In the code cell below, inspect the data in DataFrame `df` by printing the number of rows and columns, the column names, and the first ten rows. You may perform any other techniques you'd like to inspect the data.

[4]:
```
# YOUR CODE HERE

# solution
print(df.shape)
print(list(df.columns))
df.head(10)
```

```
(28022, 43)
['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', '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']
```

[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	8	True
1	1	1	True
2	1	1	True
3	1	1	True
4	1	1	True
5	3	3	True
6	1	1	True
7	3	3	True
8	2	2	True
9	1	1	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 43 columns]
```

Define the Label Assume that your goal is to train a machine learning model that predicts whether an Airbnb host is a 'super host'. This is an example of supervised learning and is a binary classification problem. In our dataset, our label will be the `host_is_superhost` column and the label will either contain the value `True` or `False`. Let's inspect the values in the `host_is_superhost` column.

[5]:
```
df['host_is_superhost']
```

[5]:
```
0        False
1        False
2        False
3        False
4        False
         ...
28017    False
28018    False
28019     True
28020     True
```

```
28021    False
Name: host_is_superhost, Length: 28022, dtype: bool
```

Identify Features Our features will be all of the remaining columns in the dataset.

Task: Create a list of the feature names.

[6]:
```
# YOUR CODE HERE

# Solution
list(df.loc[:, df.columns != 'host_is_superhost'])
```

[6]:
```
['host_response_rate',
 'host_acceptance_rate',
 'host_listings_count',
 'host_total_listings_count',
 'host_has_profile_pic',
 'host_identity_verified',
 'neighbourhood_group_cleansed',
 'room_type',
 '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']
```

1.2 Part 2. Prepare Your Data

Many of the data preparation techniques that you practiced in Unit two have already been performed and the data is almost ready for modeling. The one exception is that a few string-valued categorical features remain. Let's perform one-hot encoding to transform these features into numerical boolean values. This will result in a data set that we can use for modeling.

Identify the Features that Should be One-Hot Encoded **Task**: Find all of the columns whose values are of type 'object' and add the column names to a list named `to_encode`.

[7]:
```
# YOUR CODE HERE

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

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

Task: Find the number of unique values each column in `to_encode` has:

[8]:
```
# YOUR CODE HERE

### Solution:

df[to_encode].nunique()
```

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

One-Hot Encode the Features Instead of one-hot encoding each column using the NumPy `np.where()` or Pandas `pd.get_dummies()` functions, we can use the more robust `OneHotEncoder` transformation class from `sklearn`. For more information, consult the online documentation.

Note: We are working with `sklearn` version 0.22.2. You can find documentation for the `OneHotEncoder` class that that corresponds to our version of `sklearn` here. When choosing which features of the `OneHotEncoder` class to use, do not use features that have been introduced in newer versions of `sklearn`. For example, you should specify the parameter `sparse=False` when calling `OneHotEncoder()` to create an encoder object. The documentation notes that the latest version of `sklearn` uses the `sparse_ouput` parameter instead of `sparse`, but you should stick with `sparse`.

Task: Refer to the documenation and follow the instructions in the code cell below to create one-hot encoded features.

[9]:
```
from sklearn.preprocessing import OneHotEncoder  # Import OneHotEncoder
```

```
# Create the encoder:
# Create the  Scikit-learn OneHotEncoder object below and assign to variable
 ↪'enc'.
# When calling OneHotEncoder(), specify that the 'sparse' parameter is False
# enc = # YOUR CODE HERE

### Solution:
encoder = OneHotEncoder(sparse=False)

# Apply the encoder:
# Use the method 'enc.fit_transform() to fit the encoder to the data (the two
 ↪columns) and transform the data into
# one-hot encoded values
# Convert the results to a DataFrame and save it to variable 'df_enc'
#df_enc = # YOUR CODE HERE

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

Let's inspect our new DataFrame `df_enc` that contains the one-hot encoded columns.

[10]:
```
df_enc.head()
```

[10]:

	0	1	2	3	4	5	6	7	8
0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0

Notice that the column names are numerical.
Task: Complete the code below to reinstate the original column names.

[11]:
```
# Use the method enc.get_feature_names() to resintate the original column names.
 ↪
# Call the function with the original two column names as arguments.
# Save the results to 'df_enc.columns'

#df_enc.columns = # YOUR CODE HERE

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

Let's inspect our new DataFrame `df_enc` once again.

[12]:
```
df_enc.head(10)
```

[12]:

	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
5                                   0.0                                   1.0
6                                   0.0                                   1.0
7                                   0.0                                   0.0
8                                   0.0                                   1.0
9                                   0.0                                   1.0

   neighbourhood_group_cleansed_Manhattan  \
0                                     1.0
1                                     0.0
2                                     0.0
3                                     1.0
4                                     1.0
5                                     0.0
6                                     0.0
7                                     1.0
8                                     0.0
9                                     0.0

   neighbourhood_group_cleansed_Queens  \
0                                  0.0
1                                  0.0
2                                  0.0
3                                  0.0
4                                  0.0
5                                  0.0
6                                  0.0
7                                  0.0
8                                  0.0
9                                  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
5                                         0.0                        0.0
6                                         0.0                        1.0
7                                         0.0                        0.0
8                                         0.0                        0.0
9                                         0.0                        1.0

   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
5	0.0	1.0	0.0
6	0.0	0.0	0.0
7	0.0	1.0	0.0
8	0.0	1.0	0.0
9	0.0	0.0	0.0

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

[13]:
```
# YOUR CODE HERE

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

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

[14]:
```
# YOUR CODE HERE

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

Glance at the resulting column names:

[15]:
```
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')
```

1.3 Part 3: Create Labeled Examples from the Data Set

Task: Obtain the feature columns from DataFrame `df` and assign to `X`. Obtain the label column from DataFrame `df` and assign to `y`.

[16]:
```
# YOUR CODE HERE

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

[17]:
```
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']
```

1.4 Part 4: Create Training and Test Data 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. Save the results to variables `X_train, X_test, y_train, y_test`.

Specify: 1. A test set that is one third (.33) of the size of the data set. 2. A seed value of '123'.

[18]:
```
# YOUR CODE HERE

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

Task: Check the dimensions of the training and test datasets.

[19]:
```
# YOUR CODE HERE

### Solution:

print(X_train.shape)
print(X_test.shape)
```

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

1.5 Part 5. Train Decision Tree Classifers and Evaluate their Performances

The code cell below contains a function definition named `train_test_DT()`. This function should: 1. train a Decision Tree classifier on the training data (Remember to use `DecisionTreeClassifier()` to create a model object). 2. test the resulting model on the test data 3. compute and return the accuracy score of the resulting predicted class labels on the test data.

Task: Complete the function to make it work.

[20]:
```
def train_test_DT(X_train, X_test, y_train, y_test, depth, leaf=1,␣
↪crit='entropy'):

    # YOUR CODE HERE

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

    model.fit(X_train, y_train)

    class_label_predictions = model.predict(X_test)

    acc_score = accuracy_score(y_test, class_label_predictions)



    return acc_score
```

Train Two Decision Trees and Evaluate Their Performances Task: Use your function to train two different decision trees, one with a max depth of 8 and one with a max depth of 32. Print the max depth and corresponding accuracy score.

[21]:
```
# YOUR CODE HERE

# SOLUTION (solutions may vary)
accuracy_list = []
max_depth_range = [8,32]
for md in max_depth_range:
    score = train_test_DT(X_train, X_test, y_train, y_test, md)
    print('Max Depth=' + str(md) + ', accuracy score: ' + str(score))
    accuracy_list.append(float(score))
```

```
Max Depth=8, accuracy score: 0.833044982698962
Max Depth=32, accuracy score: 0.804606401384083
```

Visualize Accuracy We will be creating multiple visualizations that plot a specific model's hyperparameter value (such as max depth) and the resulting accuracy score of the model.

To create more clean and maintainable code, we will create one visualization function that can be called every time a plot is needed.

Task: In the code cell below, create a function called `visualize_accuracy()` that accepts two arguments:

1. a list of hyperparamter values
2. a list of accuracy scores

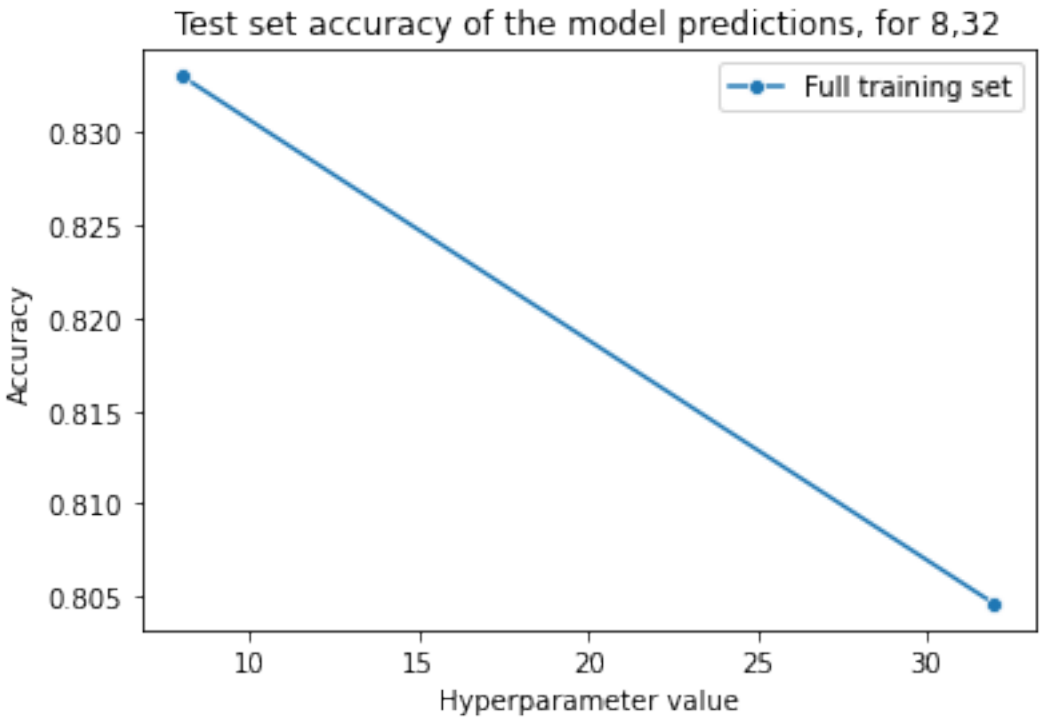
Both lists must be of the same size.

Inside the function, implement a `seaborn` lineplot in which hyperparameter values will be on the x-axis and accuracy scores will be on the y-axis. Hint: You implemented a lineplot in this week's assignment.

[22]:
```
# YOUR CODE HERE

# Solution (solutions may vary slightly)
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()
```

Task: Test your visualization function below by calling the function to plot the max depth values and accuracy scores of the two decision trees that you just trained.

[23]:
```
# YOUR CODE HERE

# SOLUTION (solutions may vary)
visualize_accuracy(max_depth_range, accuracy_list)
```



Analysis: Does this graph provide a sufficient visualization for determining a value of max depth that produces a high performing model?

Solution: Yes, we can clearly see the hyperparameter value and corresponding accuracy

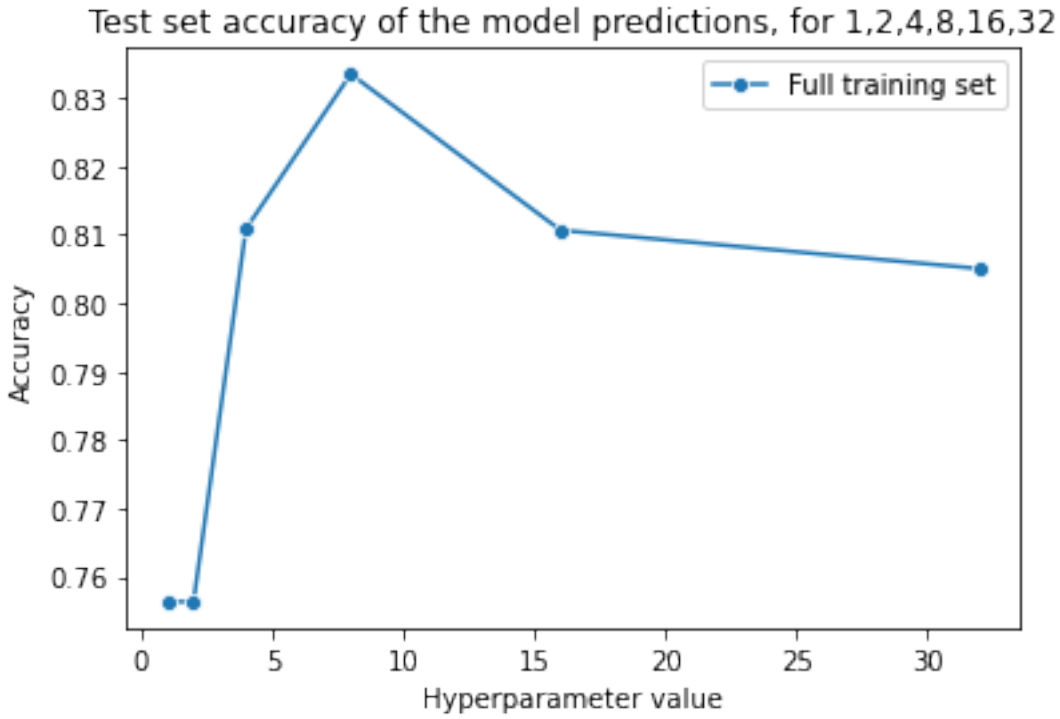
Train Multiple Decision Trees Using Different Hyperparameter Values and Evaluate Their Performances Task: Let's train on more values for max depth.

1. Train six different decision trees, using the following values for max depth: $1, 2, 4, 8, 16, 32$
2. Use your visualization function to plot the values of max depth and each model's resulting accuracy score.

[24]:
```
# YOUR CODE HERE


# Solution (solutions will vary)
accuracy_list = []
max_depth_range = [2**i for i in range(6)]
for md in max_depth_range:
    score = train_test_DT(X_train, X_test, y_train, y_test, md)
```

```
    print('Max Depth=' + str(md) + ', accuracy score: ' + str(score))
    accuracy_list.append(float(score))

visualize_accuracy(max_depth_range, accuracy_list)
```

```
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: 0.8334775086505191
Max Depth=16, accuracy score: 0.8106617647058824
Max Depth=32, accuracy score: 0.8050389273356401
```



Analysis: Analyze this graph. Pay attention to the accuracy scores. Answer the following questions in the cell below.

How would you go about choosing the best model configuration based on this plot? What other hyperparameters of interest would you want to tune to make sure you are finding the best performing model?

Solution: The best model configuration would be the one with the value of max depth that produces the highest accuracy score. In this case it would be a max depth of 8. We can also try experimenting with different values of the leaf hyperparameter.

1.6 Part 6. Train KNN Classifiers and Evaluate their Performances

The code cell below contains a function definition named `train_test_knn()`. This function should: 1. train a KNN classifier on the training data (Remember to use `KNeighborsClassifier()`

to create a model object) 2. test the resulting model on the test data 3. compute and return the accuracy score of the resulting predicted class labels on the test data.

Note: You will train KNN classifiers using the same training and test data that you used to train decision trees.

Task: Complete the function to make it work.

[25]:
```
def train_test_knn(X_train, X_test, y_train, y_test, k):

    # YOUR CODE HERE
    ### SOLUTION (variable names may vary)
    model = KNeighborsClassifier(n_neighbors = k)
    model.fit(X_train, y_train)
    class_label_predictions = model.predict(X_test)
    acc_score = accuracy_score(y_test, class_label_predictions)

    return acc_score
```

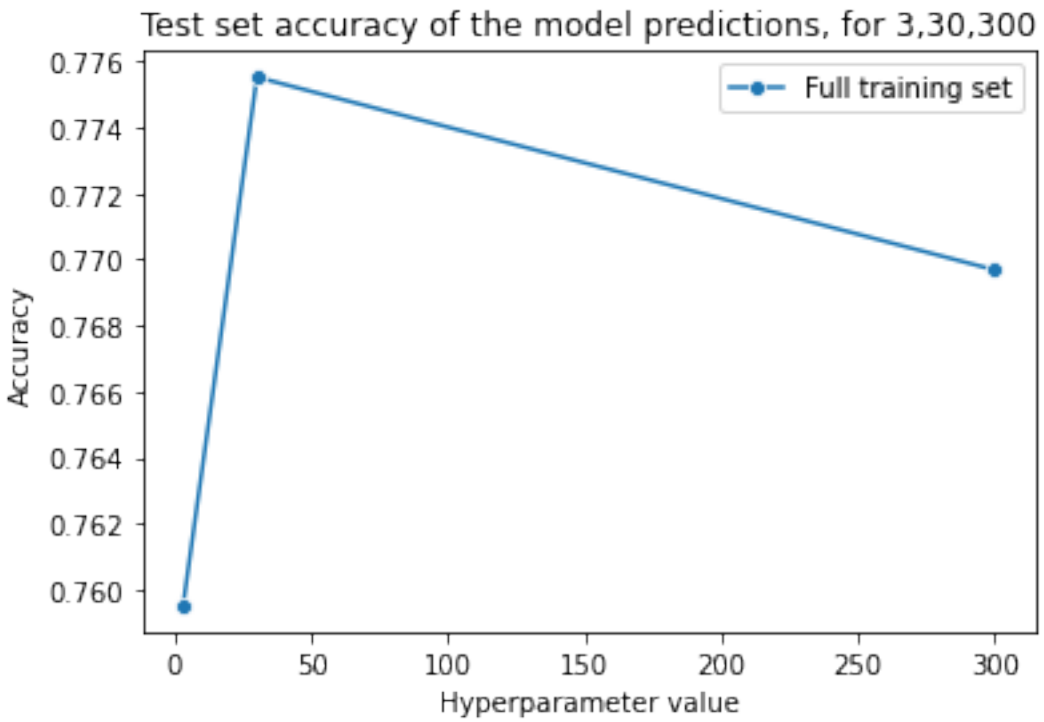
Train Three KNN Classifiers and Evaluate Their Performances Task: Use your function to train three different KNN classifiers, each with a different value for hyperparameter k: 3, 30, and 300. Note: This make take a second.

[26]:
```
# YOUR CODE HERE

# SOLUTION (solutions may vary)
accuracy_list = []
k_range = [3, 30, 300]
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))
```

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

Task: Now call the function `visualize_accuracy()` with the appropriate arguments to plot the results.

[27]:
```
# YOUR CODE HERE

# SOLUTION (solutions may vary slighly)
visualize_accuracy(k_range, accuracy_list)
```



Train Multiple KNN Classifiers Using Different Hyperparameter Values and Evaluate Their Performances Task: Let's train on more values for k.

1. Array `k_range` contains multiple values for hyperparameter k. Train one KNN model per value of k
2. Use your visualization function to plot the values of k and each model's resulting accuracy score.

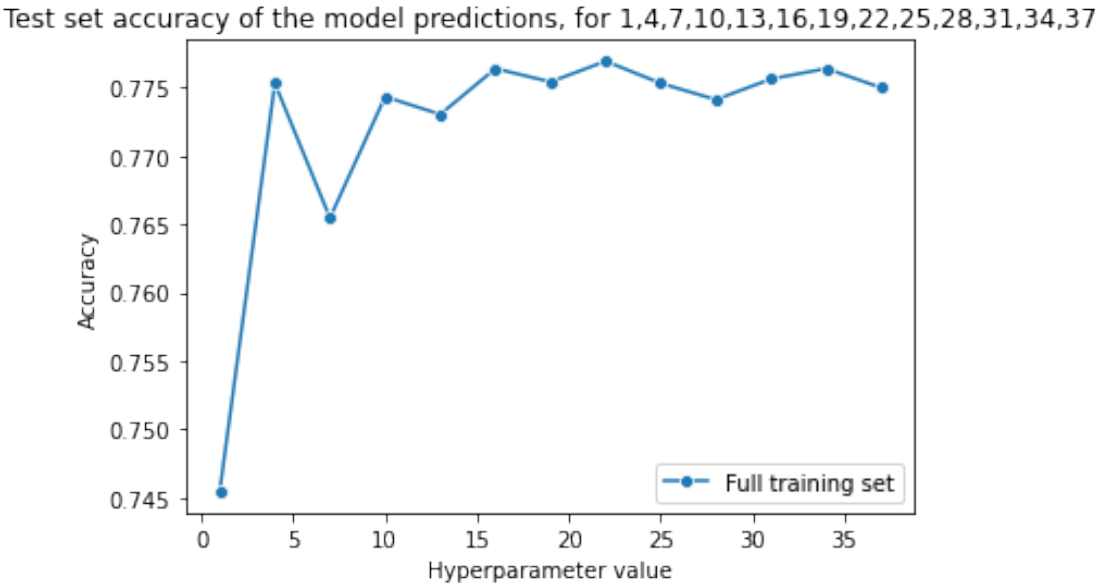
Note: This make take a second.

[28]:
```
k_range = np.arange(1, 40, step = 3)
k_range
```

[28]:
```
array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37])
```

[29]:
```
# YOUR CODE HERE

# SOLUTION (solutions may vary slightly)
accuracy_list = []
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))

visualize_accuracy(k_range, accuracy_list)
```

```
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
```



1.7 Part 7: Analysis

1. Compare the performance of the KNN model relative to the Decision Tree model, with various hyperparameter values. Which model performed the best (yielded the highest accuracy score)? Record your findings in the cell below.

2. We tuned hyperparameter k for KNNs and hyperparamter max depth for DTs. Consider other hyperparameters that can be tuned in an attempt to find the best performing model. Try a different combination of hyperparamters for both KNNs and DTs, retrain the models, obtain the accuracy scores and record your findings below.

   Note: You can consult Scikit-learn documentation for both the `KNeighborsClassifier` class and the `DecisionTreeClassifier` class to see how specific hyperparameters are passed as parameters to the model object.

Solution: 1. Decision tree with max depth of 8 performed the best. 2. Students can try different values for leaf (DT) and the distance function (KNN).