CSE572-Lab4-key

October 3, 2022

1 CSE 572: Lab 4

In this lab, you will practice implementing k nearest neighbors, decision trees, and random forests.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things: 1. a link to your Colab notebook, 2. the .ipynb file, and 3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

Acknowledgment: Much of the content in this notebook was adapted from Introduction to Data Mining, 2nd Edition by Tan, Steinbach, Karpatne, Kumar.

1.1 Vertebrate Dataset

The vertebrate dataset we will use in this lab consists of samples containing information about vertebrates. Each vertebrate is classified into one of 5 categories: mammals, reptiles, birds, fishes, and amphibians, based on a set of explanatory attributes (predictor variables). Except for "name", the rest of the attributes have been converted into a binary representation. To illustrate this, we will first load the data into a Pandas DataFrame object and display its content.

[1]:
```
import pandas as pd

data = pd.read_csv('https://docs.google.com/uc?
  ↪export=download&id=1DrqbYx-0E8qdHexxO7m9fo11444pz5v5', header='infer')
data
```

[1]:

	Name	Warm-blooded	Gives Birth	Aquatic Creature	\
0	human	1	1	0	
1	python	0	0	0	
2	salmon	0	0	1	
3	whale	1	1	1	
4	frog	0	0	1	
5	komodo	0	0	0	
6	bat	1	1	0	

7	pigeon	1	0	0
8	cat	1	1	0
9	leopard shark	0	1	1
10	turtle	0	0	1
11	penguin	1	0	1
12	porcupine	1	1	0
13	eel	0	0	1
14	salamander	0	0	1

	Aerial Creature	Has Legs	Hibernates	Class
0	0	1	0	mammals
1	0	0	1	reptiles
2	0	0	0	fishes
3	0	0	0	mammals
4	0	1	1	amphibians
5	0	1	0	reptiles
6	1	1	1	mammals
7	1	1	0	birds
8	0	1	0	mammals
9	0	0	0	fishes
10	0	1	0	reptiles
11	0	1	0	birds
12	0	1	1	mammals
13	0	0	0	fishes
14	0	1	1	amphibians

Given the limited number of training examples (15), suppose we convert the problem into a binary classification task (mammals versus non-mammals). We can do so by replacing the class labels of the instances to *non-mammals* except for those that belong to the *mammals* class.

[2]:
```
data['Class'] = data['Class'].
 ↪replace(['fishes','birds','amphibians','reptiles'], 'non-mammals')
data
```

[2]:

	Name	Warm-blooded	Gives Birth	Aquatic Creature	\
0	human	1	1	0	
1	python	0	0	0	
2	salmon	0	0	1	
3	whale	1	1	1	
4	frog	0	0	1	
5	komodo	0	0	0	
6	bat	1	1	0	
7	pigeon	1	0	0	
8	cat	1	1	0	
9	leopard shark	0	1	1	
10	turtle	0	0	1	
11	penguin	1	0	1	

```
12      porcupine              1             1                 0
13            eel              0             0                 1
14     salamander              0             0                 1

    Aerial Creature  Has Legs  Hibernates        Class
0                 0         1           0      mammals
1                 0         0           1  non-mammals
2                 0         0           0  non-mammals
3                 0         0           0      mammals
4                 0         1           1  non-mammals
5                 0         1           0  non-mammals
6                 1         1           1      mammals
7                 1         1           0  non-mammals
8                 0         1           0      mammals
9                 0         0           0  non-mammals
10                0         1           0  non-mammals
11                0         1           0  non-mammals
12                0         1           1      mammals
13                0         0           0  non-mammals
14                0         1           1  non-mammals
```

We can use the Pandas cross-tabulation function to examine the relationship between the Warm-blooded and Gives Birth attributes with respect to the class. This cross-tabulation gives the counts of mammals and non-mammals associated with each combination of Warm-blooded and Gives Birth values.

[3]:
```
pd.crosstab([data['Warm-blooded'], data['Gives Birth']], data['Class'])
```

[3]:
```
Class                      mammals  non-mammals
Warm-blooded Gives Birth
0            0                   0            7
             1                   0            1
1            0                   0            2
             1                   5            0
```

The results above show that it is possible to distinguish mammals from non-mammals using these two attributes alone since each combination of their attribute values would yield only instances that belong to the same class. For example, mammals can be identified as warm-blooded vertebrates that give birth to their young. Such a relationship can also be derived using a decision tree classifier, as shown by the example given in the next subsection.

1.2 Decision Tree Classifier

In this section, we apply a decision tree classifier to the vertebrate dataset described in the previous subsection.

[4]:
```
from sklearn import tree
```

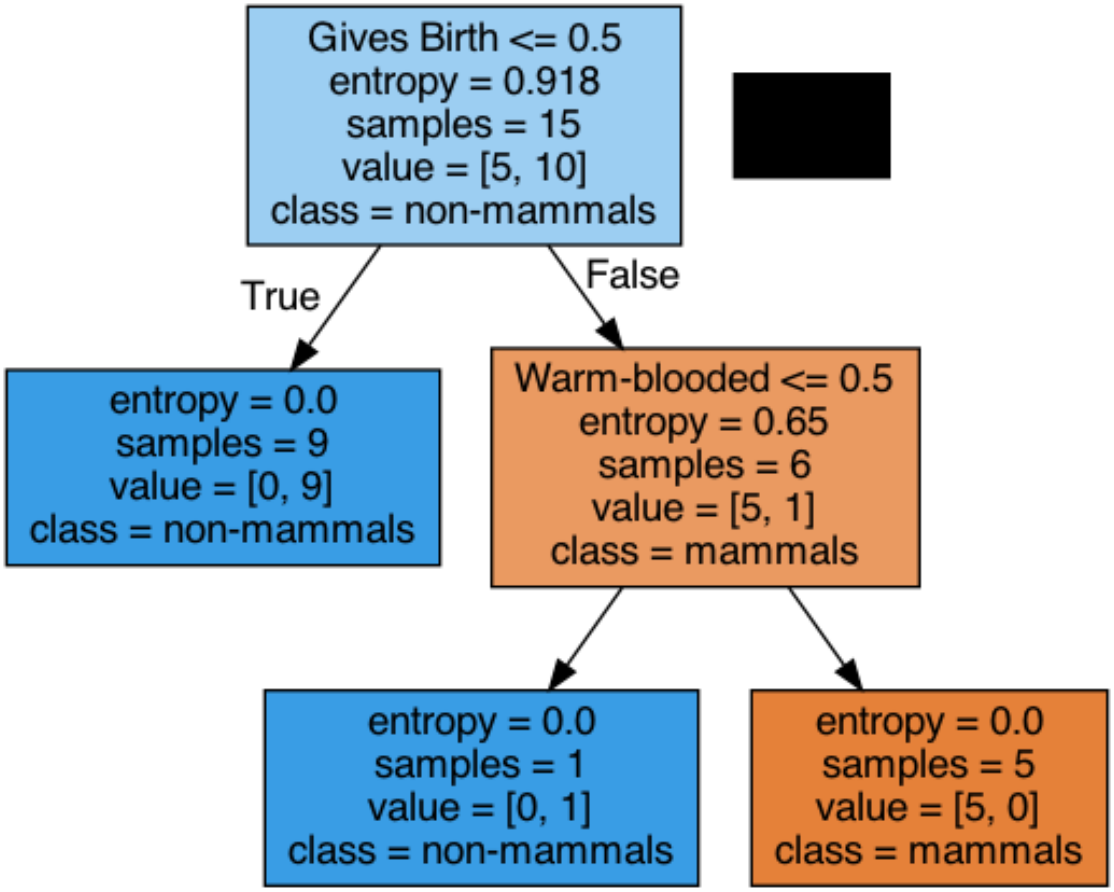
```
Y = data['Class']
X = data.drop(['Name','Class'], axis=1)

clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3)
clf = clf.fit(X, Y)
```

The preceding commands will extract the predictor (X) and target class (Y) attributes from the vertebrate dataset and create a decision tree classifier object using entropy as its impurity measure for splitting criterion. The decision tree class in Python sklearn library also supports using 'gini' as impurity measure. The classifier above is also constrained to generate trees with a maximum depth equals to 3. Next, the classifier is trained on the labeled data using the fit() function.

We can plot the resulting decision tree obtained after training the classifier. To do this, you must first install both graphviz (http://www.graphviz.org) and its Python interface called pydotplus (http://pydotplus.readthedocs.io/).

[5]:
```
import pydotplus
from IPython.display import Image

dot_data = tree.export_graphviz(clf, feature_names=X.columns,␣
 ↪class_names=['mammals','non-mammals'], filled=True,
                                out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[5]:



Next, suppose we apply the decision tree to classify the following test examples.

[6]:
```
testData = [['gila monster',0,0,0,0,1,1,'non-mammals'],
            ['platypus',1,0,0,0,1,1,'mammals'],
            ['owl',1,0,0,1,1,0,'non-mammals'],
            ['dolphin',1,1,1,0,0,0,'mammals']]

testData = pd.DataFrame(testData, columns=data.columns)
testData
```

[6]:

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	gila monster	0	0	0	0	1	1	non-mammals
1	platypus	1	0	0	0	1	1	mammals
2	owl	1	0	0	1	1	0	non-mammals
3	dolphin	1	1	1	0	0	0	mammals

```
2          1           0  non-mammals
3          0           0      mammals
```

We first extract the predictor and target class attributes from the test data and then apply the decision tree classifier to predict their classes.

[7]:
```
testY = testData['Class']
testX = testData.drop(['Name','Class'],axis=1)

predY = clf.predict(testX)
predictions = pd.concat([testData['Name'], pd.Series(predY,name='Predicted␣
 ↪Class')], axis=1)
predictions
```

[7]:
```
            Name Predicted Class
0  gila monster     non-mammals
1      platypus     non-mammals
2           owl     non-mammals
3       dolphin         mammals
```

Except for platypus, which is an egg-laying mammal, the classifier correctly predicts the class label of the test examples. We can calculate the accuracy of the classifier on the test data as shown by the example given below.

[8]:
```
from sklearn.metrics import accuracy_score

print('Accuracy on test data is %.2f' % (accuracy_score(testY, predY)))
```

```
Accuracy on test data is 0.75
```

1.3 Model Overfitting

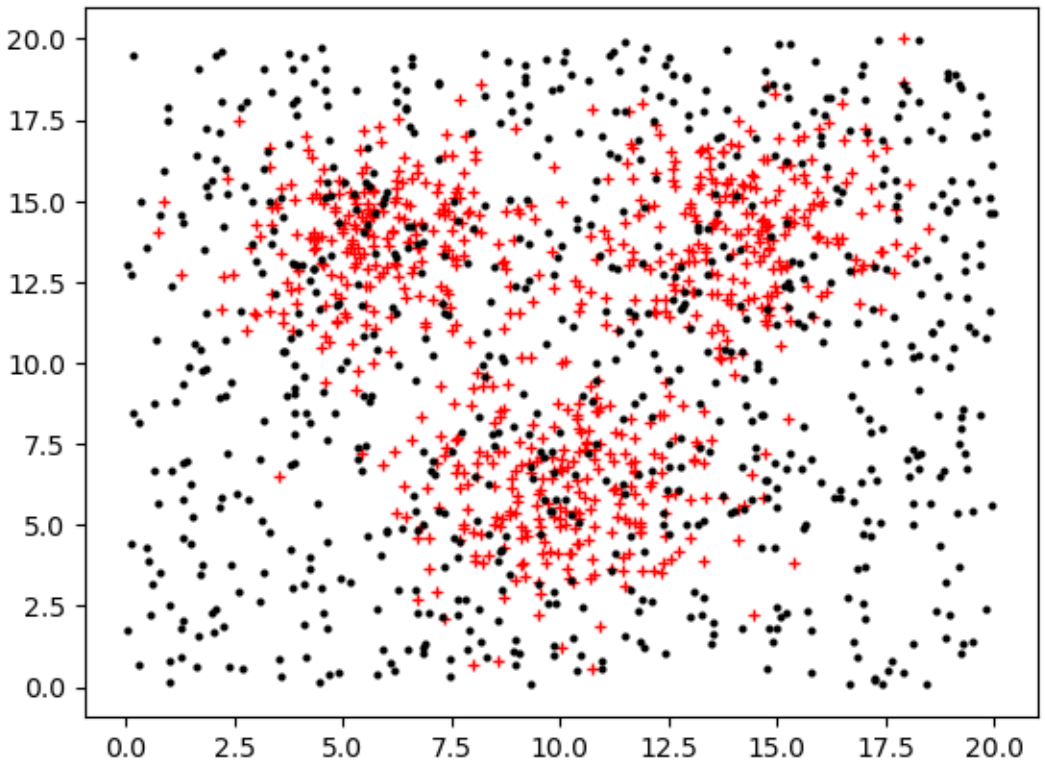
To illustrate the problem of model overfitting, we consider a synthetic two-dimensional dataset containing 1500 labeled instances, each of which is assigned to one of two classes, 0 or 1. Instances from each class are generated as follows: 1. Instances from class 1 are generated from a mixture of 3 two-dimensional Gaussian distributions, centered at [6, 14], [10, 6], and [14, 14], respectively. 2. Instances from class 0 are generated from a uniform distribution in a two-dimensional square region, whose sides have a length equal to 20.

For simplicity, both classes have equal number of labeled instances. The code for generating and plotting the data is shown below. All instances from class 1 are shown in red while those from class 0 are shown in black.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import random

%matplotlib inline
```

```
N = 1500

mean1 = [6, 14]
mean2 = [10, 6]
mean3 = [14, 14]
cov = [[3.5, 0], [0, 3.5]]  # diagonal covariance

np.random.seed(50)
# Generate the dataset from class 1
X = np.random.multivariate_normal(mean1, cov, int(N/6))
X = np.concatenate((X, np.random.multivariate_normal(mean2, cov, int(N/6))))
X = np.concatenate((X, np.random.multivariate_normal(mean3, cov, int(N/6))))

# Generate the dataset from class 0 and concatenate to the first dataset
X = np.concatenate((X, 20*np.random.rand(int(N/2), 2)))

# Assign the labels to classes 0 and 1
Y = np.concatenate((np.ones(int(N/2)), np.zeros(int(N/2))))

plt.plot(X[:int(N/2),0],X[:int(N/2),1],'r+',X[int(N/2):,0],X[int(N/2):,1],'k.
 ↪',ms=4)
```

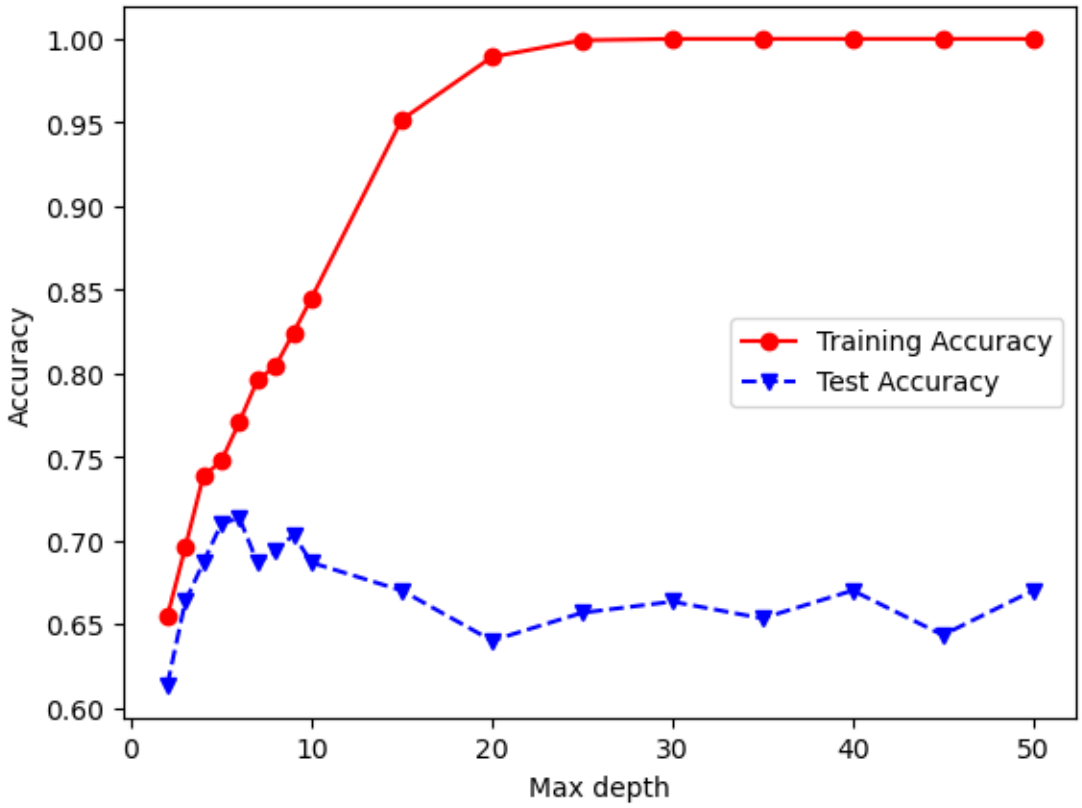
[9]:
```
[<matplotlib.lines.Line2D at 0x19eea29a0>,
 <matplotlib.lines.Line2D at 0x19eea2a30>]
```



In this example, we reserve 80% of the labeled data for training and the remaining 20% for testing. We then fit decision trees of different maximum depths (from 2 to 50) to the training set and plot their respective accuracies when applied to the training and test sets.

[10]:
```
########################################
# Training and Test set creation
########################################

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=1)

from sklearn import tree
from sklearn.metrics import accuracy_score

########################################
# Model fitting and evaluation
########################################

maxdepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]

trainAcc = np.zeros(len(maxdepths))
testAcc = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    clf = clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc[index] = accuracy_score(Y_train, Y_predTrain)
    testAcc[index] = accuracy_score(Y_test, Y_predTest)
    index += 1

########################################
# Plot of training and test accuracies
########################################

plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

[10]:
```
Text(0, 0.5, 'Accuracy')
```



Question 1: What happens to the training accuracy as the model becomes more complex (maximum depth of tree increases)? What happens to the test accuracy?

Answer:

The plot above shows that training accuracy will continue to improve as the maximum depth of the tree increases (i.e., as the model becomes more complex). However, the test accuracy initially improves up to a maximum depth of 5, before it gradually decreases due to model overfitting.

Question 2: The model begins to overfit when the test accuracy starts to decrease while the training accuracy is still increasing. What is the approximate maximum depth at which the model starts to overfit?

Answer:

The model starts to overfit after a maximum depth of about 5 trees (accept any answer between 4-6).

1.4 Alternative Classification Techniques

Besides decision tree classifier, the Python sklearn library also supports other classification techniques. In this section, we provide examples to illustrate how to apply the k-nearest neighbor and random forest classifiers to the 2-dimensional data given in the previous section.

1.4.1 K-Nearest neighbor classifier

In this approach, the class label of a test instance is predicted based on the majority class of its k closest training instances. The number of nearest neighbors, k, is a hyperparameter that must be provided by the user, along with the distance metric. By default, we can use Euclidean distance (which is equivalent to Minkowski distance with an exponent factor equals to p=2):

$$\text{Minkowski distance}(x, y) = \left[\sum_{i=1}^{N} |x_i - y_i|^p\right]^{\frac{1}{p}}$$

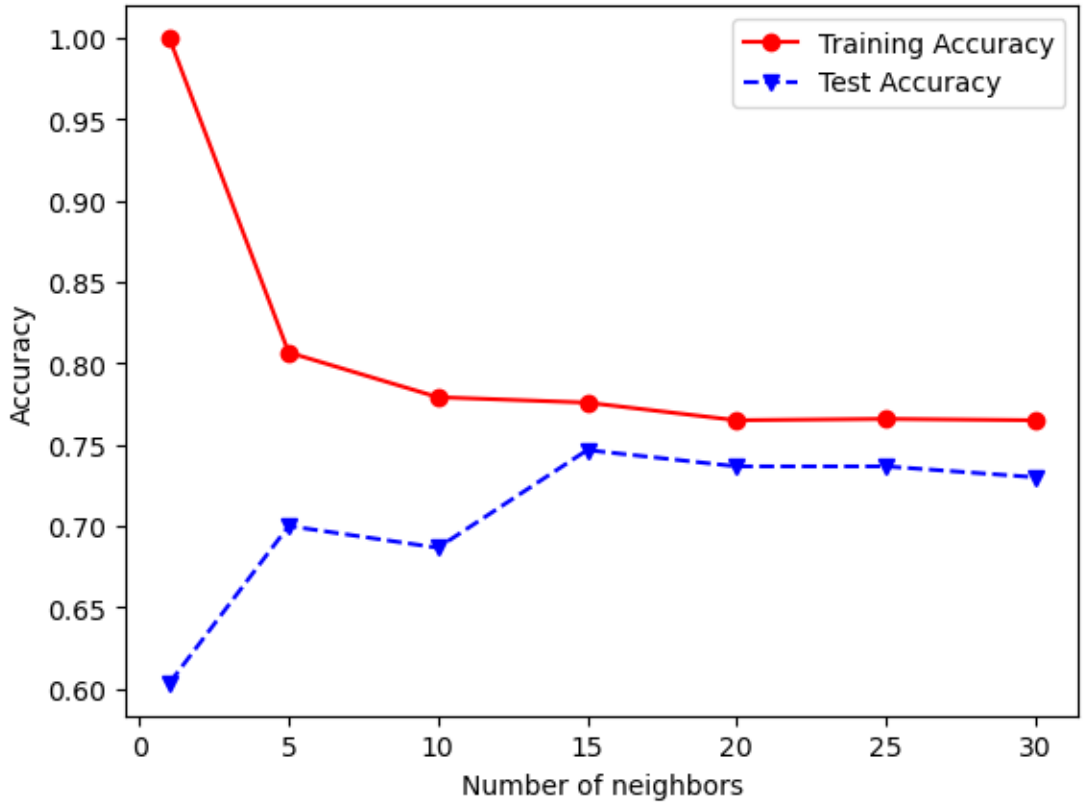
[11]:
```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
%matplotlib inline

numNeighbors = [1, 5, 10, 15, 20, 25, 30]
trainAcc = []
testAcc = []

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[11]: Text(0, 0.5, 'Accuracy')



In class, we discussed other distance metrics that could be used besides Euclidean distance, such as absolute distance (Minkowski distance with order = 1) and cosine distance. Implement kNN and create the same plot as above, but using 1) absolute distance and 2) cosine distance. You will need to consult the scikit-learn documentation to find how to change the distance metric.

[12]:
```
### Absolute distance ###
# YOUR CODE HERE

trainAcc = []
testAcc = []

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=1)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))
```

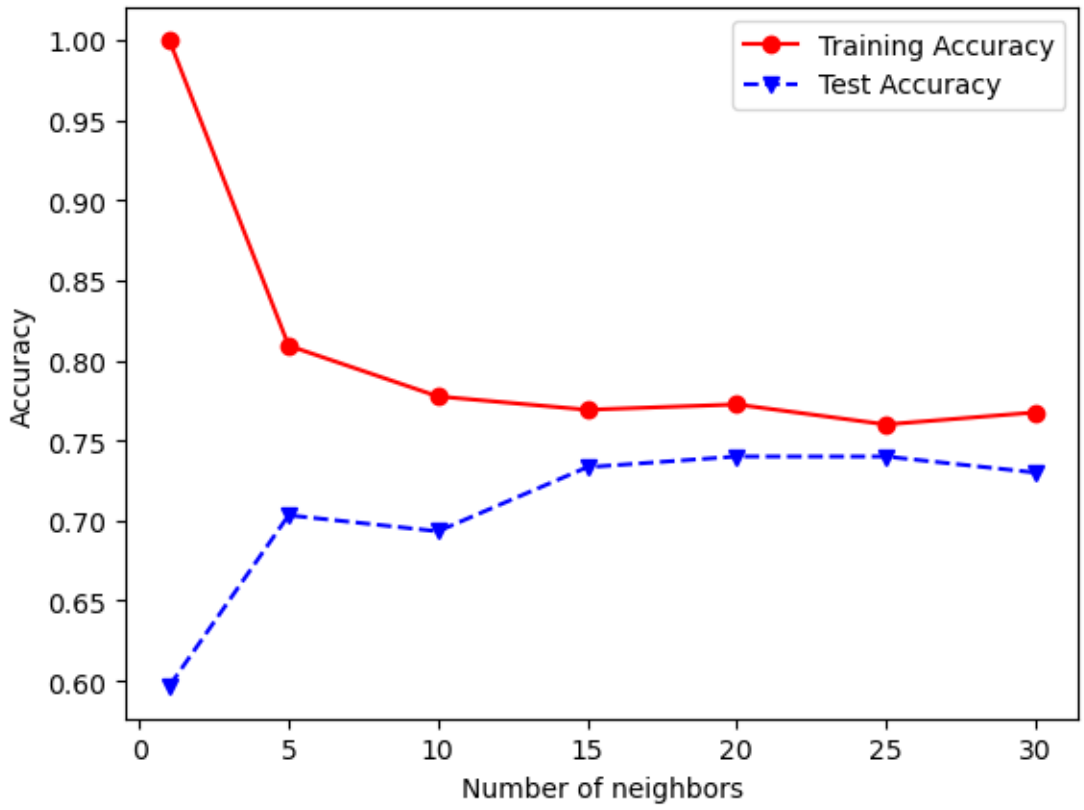
```python
plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
```

```
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
```

```
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[12]: `Text(0, 0.5, 'Accuracy')`



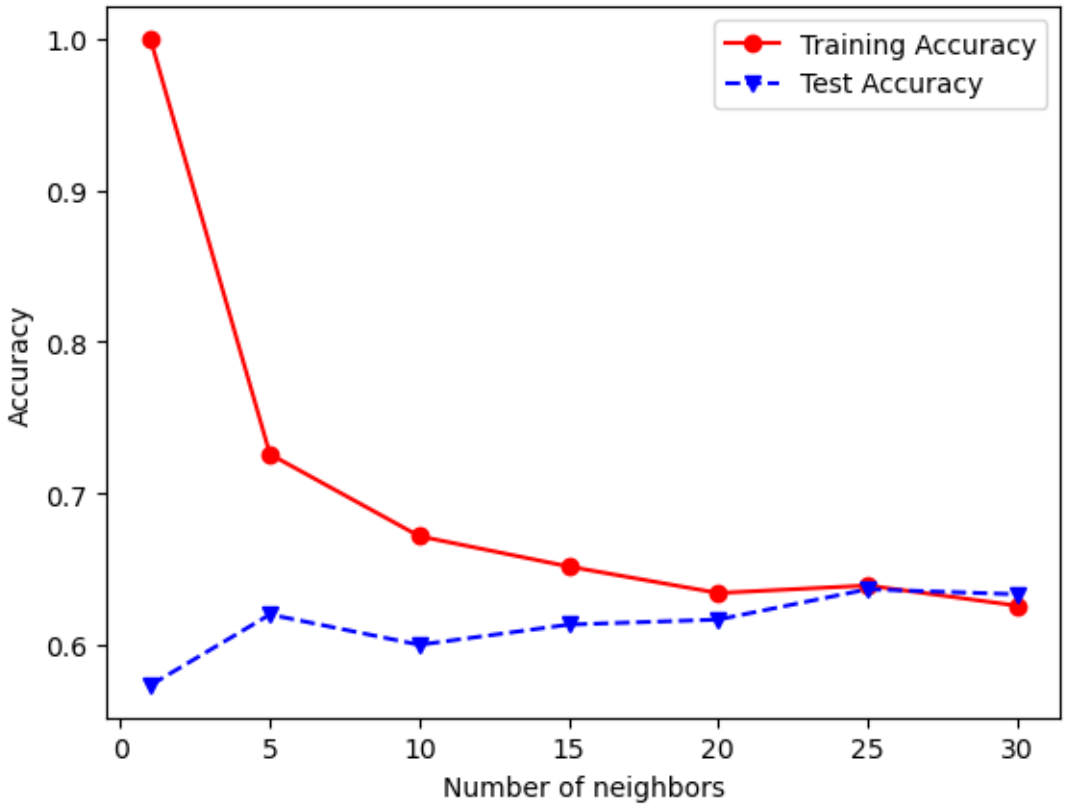
[13]:
```python
### Cosine distance ###
# YOUR CODE HERE

trainAcc = []
testAcc = []

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='cosine')
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
```

```
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
```

```
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[13]:
```
Text(0, 0.5, 'Accuracy')
```



Question 3: Which distance(s) give(s) the best overall accuracy on the test set? Which is the worst?

Answer:

Euclidean and absolute distance (Minkowski with orders 2 and 1 respectively) both have the best overall accuracy on the test set (accept answers that state either or both). Cosine distance has the worst accuracy.

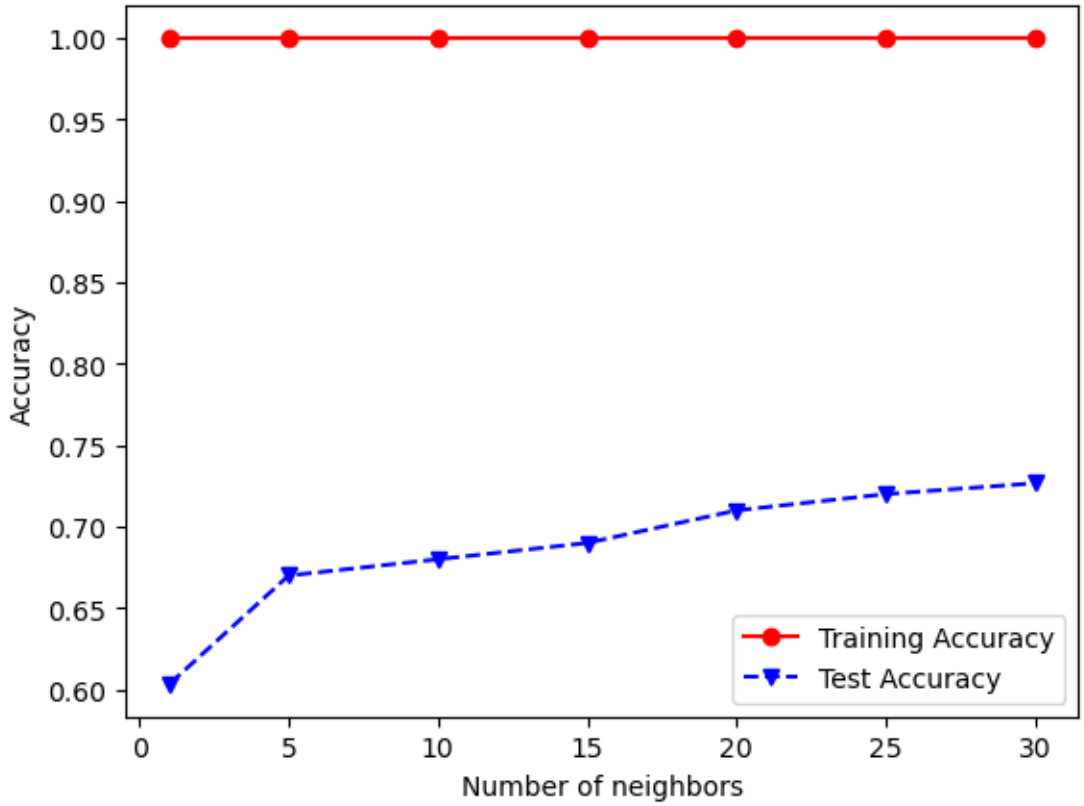
1.4.2 Weighted kNN

By default, kNN classifier in sklearn uses uniform weights—i.e., each neighbor is weighted equally when determining the class label based on the k nearest neighbors. Alternatively, we could weight the decision based on the distance of each neighbor from the test instance. Consult the documentation to figure out how to weight neighbors by their distance during prediction, then implement weighted kNN and generate the same plot as in the previous cells. Use Euclidean distance as the distance metric.

[14]:
```
### Weighted kNN ###
# YOUR CODE HERE

trainAcc = []
testAcc = []
```

```
for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2,␣
 ↪weights='distance')
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

[14]: `Text(0, 0.5, 'Accuracy')`



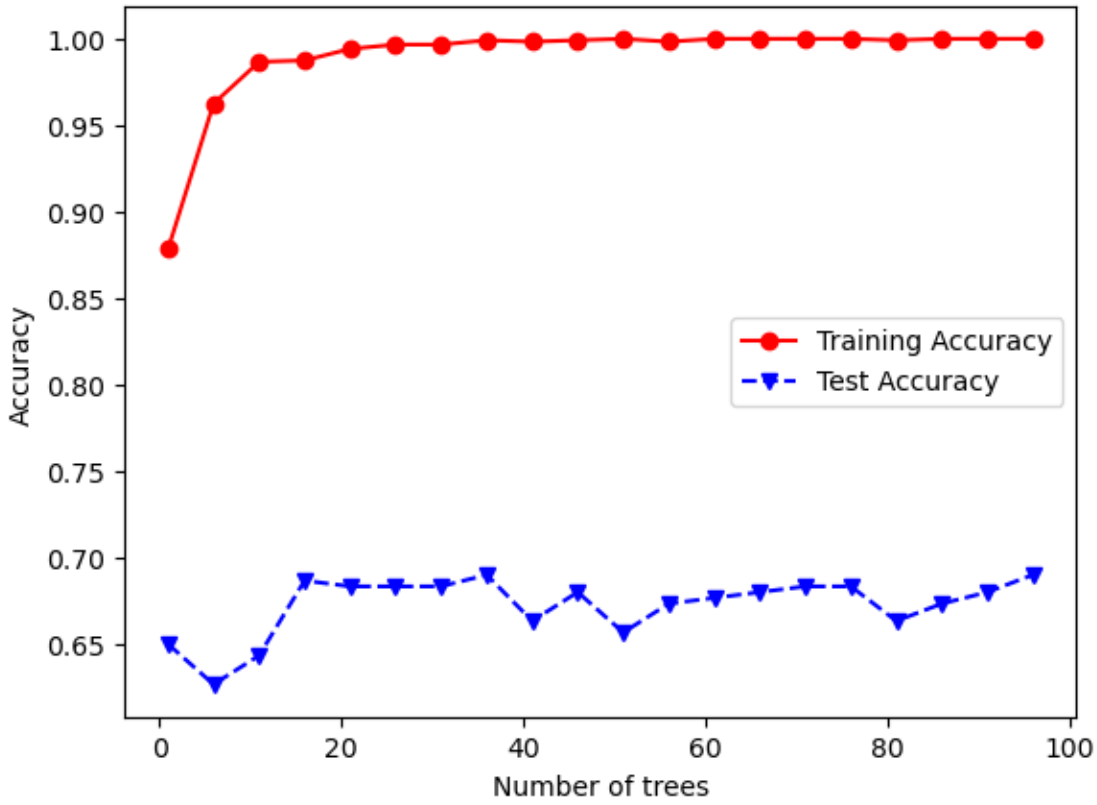
1.4.3 Random Forest

A random forest is an ensemble of decision trees designed to improve generalization to unseen test data.

In the example below, we fit a random forest with varying numbers of decision trees to the 2-dimensional dataset using each ensemble method.

[15]:
```
from sklearn import ensemble

maxdepth = 3
n_trees = range(1, 101, 5)

trainAcc = []
testAcc = []

for n in n_trees:
    clf = ensemble.RandomForestClassifier(n_estimators=n)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(n_trees, trainAcc, 'ro-', n_trees, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of trees')
plt.ylabel('Accuracy')
```

[15]:
```
Text(0, 0.5, 'Accuracy')
```



1.5 Iris dataset

Load the iris dataset used in Lab 2. Then perform the following steps:

1. Split the dataset into 80% train and 20% test (use random state = 50).
2. Train a decision tree with **max depth = 3 and using Gini index criterion** and print the resulting training and test accuracy.
3. Plot the resulting decision tree obtained after training the classifier using graphviz as in the vertebrate example.

[16]:
```
# Load the dataset
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪iris/iris.data', header=None)
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width',␣
 ↪'class']

data
```

[16]:

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

```
3            4.6          3.1           1.5          0.2     Iris-setosa
4            5.0          3.6           1.4          0.2     Iris-setosa
..           …            …             …            …       …
145          6.7          3.0           5.2          2.3  Iris-virginica
146          6.3          2.5           5.0          1.9  Iris-virginica
147          6.5          3.0           5.2          2.0  Iris-virginica
148          6.2          3.4           5.4          2.3  Iris-virginica
149          5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

[17]:
```
# Split the dataset into 80% train and 20% test. Use a random state of 50 for␣
↪reproducibility.
# YOUR CODE HERE

train = data.sample(frac=0.8, random_state=50)
test = data.drop(train.index)
```

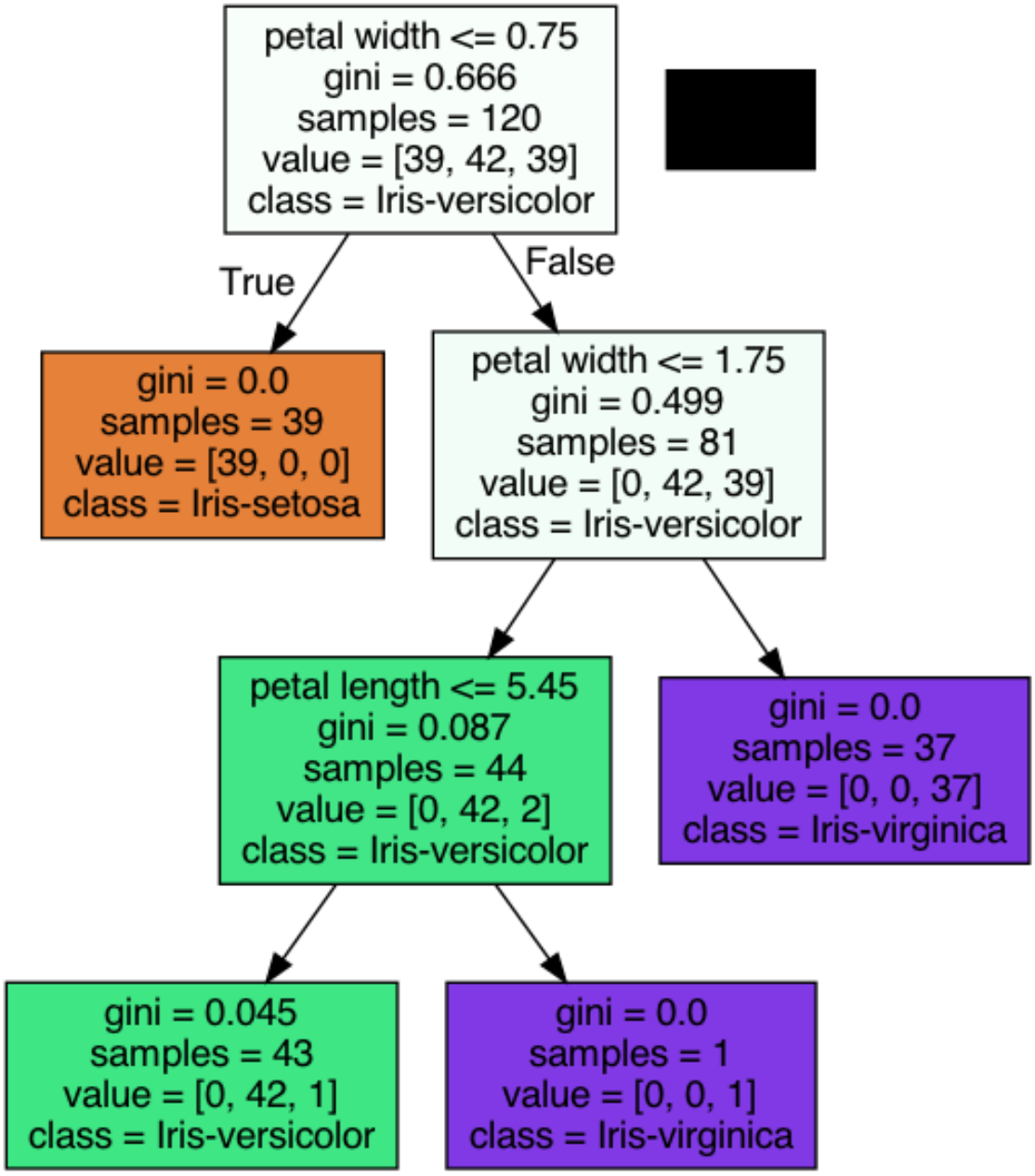
[18]:
```
# Train decision tree and print train and test accuracy
np.random.seed(50)

# YOUR CODE HERE

clf = tree.DecisionTreeClassifier(max_depth=3, criterion='gini')
clf = clf.fit(train[train.columns[:-1]], train[train.columns[-1]])
Y_predTrain = clf.predict(train[train.columns[:-1]])
Y_predTest = clf.predict(test[test.columns[:-1]])

print('Train accuracy = %.2f' % accuracy_score(train[train.columns[-1]],␣
↪Y_predTrain))
print('Test accuracy = %.2f' % accuracy_score(test[test.columns[-1]],␣
↪Y_predTest))
```

```
Train accuracy = 0.99
Test accuracy = 0.90
```

[19]:
```
# Plot resulting decision tree
# YOUR CODE HERE

dot_data = tree.export_graphviz(clf,
                                feature_names=data.columns[:-1],
                                ␣
↪class_names=['Iris-setosa','Iris-versicolor','Iris-virginica'],
                                filled=True,
                                out_file=None)

graph = pydotplus.graph_from_dot_data(dot_data)
```

```
Image(graph.create_png())
```

[19]:



Question 4: In Lab 2, we came up with rules that could be used to separate instances from the different iris classes, such as: - setosa if petal length < 2.5, virginica if petal length > 4.8, and versicolor otherwise - setosa if petal width < 1, virginica if petal width > 1.4, and versicolor otherwise

How do these rules compare to the splits learned by the decision tree?

Answer:

The decision tree also used splits on the petal length and petal width attributes, though slightly different thresholds were chosen (any answer that talks about these attributes is acceptable)

[20]:
```
X_train, X_test, Y_train, Y_test = train_test_split(data[data.columns[:-1]],␣
↪data[data.columns[-1]], test_size=0.2, random_state=50)
```

[21]:
```
# Train decision tree and train and test accuracy
np.random.seed(50)

# YOUR CODE HERE

clf = tree.DecisionTreeClassifier(max_depth=3, criterion='gini')
clf = clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)

print('Train accuracy = %.2f' % accuracy_score(Y_train, Y_predTrain))
print('Test accuracy = %.2f' % accuracy_score(Y_test, Y_predTest))
```

```
Train accuracy = 0.98
Test accuracy = 0.97
```