Silas_hw3_submission

March 5, 2023

1 Homework 3

[1]:
```
# import necessary packages
import pandas as pd
from pandas import read_csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings("ignore")

# reading csv data file
df = read_csv("AMZN.csv")
dfSonar = read_csv("sonar.all-data.csv")

# data prep
adjClose = df[['Adj Close']].values
arraySonar = dfSonar.values
dataSonar = arraySonar[:,:-1]
targetSonar = arraySonar[:,-1]
```

[2]:
```
# series_to_supervised function
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
```

```
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()

    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]

    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names

    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

1.1 Problem 1

1.1.1 (a)

[3]:
```
# create supervised learning set from closing data
supervisedDF = series_to_supervised(adjClose, 10)
print(supervisedDF)
```

```
       var1(t-10)    var1(t-9)    var1(t-8)    var1(t-7)    var1(t-6)  \
10       1.958333     1.729167     1.708333     1.635417     1.427083
11       1.729167     1.708333     1.635417     1.427083     1.395833
12       1.708333     1.635417     1.427083     1.395833     1.500000
13       1.635417     1.427083     1.395833     1.500000     1.583333
14       1.427083     1.395833     1.500000     1.583333     1.531250
…             …            …            …            …            …
5753  1676.609985  1785.000000  1689.150024  1807.839966  1830.000000
```

```
5754  1785.000000  1689.150024  1807.839966  1830.000000  1880.930054
5755  1689.150024  1807.839966  1830.000000  1880.930054  1846.089966
5756  1807.839966  1830.000000  1880.930054  1846.089966  1902.829956
5757  1830.000000  1880.930054  1846.089966  1902.829956  1940.099976

        var1(t-5)    var1(t-4)    var1(t-3)    var1(t-2)    var1(t-1)  \
10       1.395833     1.500000     1.583333     1.531250     1.505208
11       1.500000     1.583333     1.531250     1.505208     1.500000
12       1.583333     1.531250     1.505208     1.500000     1.510417
13       1.531250     1.505208     1.500000     1.510417     1.479167
14       1.505208     1.500000     1.510417     1.479167     1.416667
…             …           …           …           …           …
5753  1880.930054  1846.089966  1902.829956  1940.099976  1885.839966
5754  1846.089966  1902.829956  1940.099976  1885.839966  1955.489990
5755  1902.829956  1940.099976  1885.839966  1955.489990  1900.099976
5756  1940.099976  1885.839966  1955.489990  1900.099976  1963.949951
5757  1885.839966  1955.489990  1900.099976  1963.949951  1949.719971

          var1(t)
10       1.500000
11       1.510417
12       1.479167
13       1.416667
14       1.541667
…             …
5753  1955.489990
5754  1900.099976
5755  1963.949951
5756  1949.719971
5757  1907.699951

[5748 rows x 11 columns]
```

1.2 (b)

[4]:
```
# df to array and scaling
array = supervisedDF.values
scaler = MinMaxScaler()
scaler.fit_transform(array)
```

[4]:
```
array([[2.59357128e-04, 1.53693076e-04, 1.44087293e-04, …,
        5.28325334e-05, 4.08251620e-05, 3.84236657e-05],
       [1.53693076e-04, 1.44087293e-04, 1.10466888e-04, …,
        4.08251620e-05, 3.84236657e-05, 4.32266033e-05],
       [1.44087293e-04, 1.10466888e-04, 1.44087293e-05, …,
        3.84236657e-05, 4.32266033e-05, 2.88177355e-05],
       …,
```

```
        [7.78188587e-01, 8.32914067e-01, 8.43131601e-01, …,
         9.00991491e-01, 8.75452055e-01, 9.04892242e-01],
        [8.32914067e-01, 8.43131601e-01, 8.66614396e-01, …,
         8.75452055e-01, 9.04892242e-01, 8.98331029e-01],
        [8.43131601e-01, 8.66614396e-01, 8.50550352e-01, …,
         9.04892242e-01, 8.98331029e-01, 8.78956280e-01]])
```

1.2.1 (c)

[5]:
```
# splitting data set
data = array[:,0:10]
target = array[:,10]

# Adding bias and reshaping
shape = data.shape[0]
data = np.append(np.ones((shape,1)), data, axis=1)
target = target.reshape(shape,1)

# Normal Equation
theta = np.dot(np.linalg.inv(np.dot(data.T,data)), np.dot(data.T,target))

# train test split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(data, target, train_size =␣
 ↪0.7, random_state=1)
```

1.2.2 (d)

[6]:
```
# prediction with MSE & R2
prediction = np.dot(X_Test, theta)
print("MSE: %.3f" % mean_squared_error(Y_Test, prediction))
print("R2: %.3f" % r2_score(Y_Test, prediction))
```

```
MSE: 110.595
R2: 1.000
```

1.2.3 (e)

[7]:
```
# functions for gradient descent
def predict(row, coefficients):
    prediction = coefficients[0]
    for i in range (len(row)):
        prediction = prediction + coefficients[i+1] * row[i]
    return prediction

def coefficients_sgd(X_Train, Y_Train, learning_rate, iterations):
    coef = [0.0 for i in range(len(X_Train[0])+1)]
```

```
    for epoch in range(iterations):
        sum_error = 0
        for i in range(X_Train.shape[0]):
            gradientPrediction = predict(X_Train[i,:], coef)
            error = gradientPrediction - Y_Train[i]
            sum_error += error**2
            coef[0] = coef[0] - learning_rate * error
            for j in range(len(coef)-1):
                coef[j+1] = coef[j+1] - learning_rate * error * X_Train[i,j]
        print( ' >epoch=%d, lrate=%.3f, error=%.3f ' % (epoch, learning_rate,␣
↪sum_error))
    return coef
```

1.2.4 (f)

[8]:

```
# apply functions and predict
learning_rate = 0.01
iterations = 200
coef = coefficients_sgd(X_Train, Y_Train, learning_rate, iterations)

Y_Prediction = np.array(predict(X_Test[0,:], coef))
for i in range (X_Test.shape[0]-1):
    Y_Prediction = np.append(Y_Prediction, predict(X_Test[i+1,:], coef))

print("MSE: %.3f" % mean_squared_error(Y_Test, Y_Prediction))
print("R2: %.3f" % r2_score(Y_Test, Y_Prediction))
```

```
>epoch=0, lrate=0.010, error=nan
>epoch=1, lrate=0.010, error=nan
>epoch=2, lrate=0.010, error=nan
>epoch=3, lrate=0.010, error=nan
>epoch=4, lrate=0.010, error=nan
>epoch=5, lrate=0.010, error=nan
>epoch=6, lrate=0.010, error=nan
>epoch=7, lrate=0.010, error=nan
>epoch=8, lrate=0.010, error=nan
>epoch=9, lrate=0.010, error=nan
>epoch=10, lrate=0.010, error=nan
>epoch=11, lrate=0.010, error=nan
>epoch=12, lrate=0.010, error=nan
>epoch=13, lrate=0.010, error=nan
>epoch=14, lrate=0.010, error=nan
>epoch=15, lrate=0.010, error=nan
>epoch=16, lrate=0.010, error=nan
>epoch=17, lrate=0.010, error=nan
>epoch=18, lrate=0.010, error=nan
>epoch=19, lrate=0.010, error=nan
```

```
>epoch=20, lrate=0.010, error=nan
>epoch=21, lrate=0.010, error=nan
>epoch=22, lrate=0.010, error=nan
>epoch=23, lrate=0.010, error=nan
>epoch=24, lrate=0.010, error=nan
>epoch=25, lrate=0.010, error=nan
>epoch=26, lrate=0.010, error=nan
>epoch=27, lrate=0.010, error=nan
>epoch=28, lrate=0.010, error=nan
>epoch=29, lrate=0.010, error=nan
>epoch=30, lrate=0.010, error=nan
>epoch=31, lrate=0.010, error=nan
>epoch=32, lrate=0.010, error=nan
>epoch=33, lrate=0.010, error=nan
>epoch=34, lrate=0.010, error=nan
>epoch=35, lrate=0.010, error=nan
>epoch=36, lrate=0.010, error=nan
>epoch=37, lrate=0.010, error=nan
>epoch=38, lrate=0.010, error=nan
>epoch=39, lrate=0.010, error=nan
>epoch=40, lrate=0.010, error=nan
>epoch=41, lrate=0.010, error=nan
>epoch=42, lrate=0.010, error=nan
>epoch=43, lrate=0.010, error=nan
>epoch=44, lrate=0.010, error=nan
>epoch=45, lrate=0.010, error=nan
>epoch=46, lrate=0.010, error=nan
>epoch=47, lrate=0.010, error=nan
>epoch=48, lrate=0.010, error=nan
>epoch=49, lrate=0.010, error=nan
>epoch=50, lrate=0.010, error=nan
>epoch=51, lrate=0.010, error=nan
>epoch=52, lrate=0.010, error=nan
>epoch=53, lrate=0.010, error=nan
>epoch=54, lrate=0.010, error=nan
>epoch=55, lrate=0.010, error=nan
>epoch=56, lrate=0.010, error=nan
>epoch=57, lrate=0.010, error=nan
>epoch=58, lrate=0.010, error=nan
>epoch=59, lrate=0.010, error=nan
>epoch=60, lrate=0.010, error=nan
>epoch=61, lrate=0.010, error=nan
>epoch=62, lrate=0.010, error=nan
>epoch=63, lrate=0.010, error=nan
>epoch=64, lrate=0.010, error=nan
>epoch=65, lrate=0.010, error=nan
>epoch=66, lrate=0.010, error=nan
>epoch=67, lrate=0.010, error=nan
```

```
>epoch=68, lrate=0.010, error=nan
>epoch=69, lrate=0.010, error=nan
>epoch=70, lrate=0.010, error=nan
>epoch=71, lrate=0.010, error=nan
>epoch=72, lrate=0.010, error=nan
>epoch=73, lrate=0.010, error=nan
>epoch=74, lrate=0.010, error=nan
>epoch=75, lrate=0.010, error=nan
>epoch=76, lrate=0.010, error=nan
>epoch=77, lrate=0.010, error=nan
>epoch=78, lrate=0.010, error=nan
>epoch=79, lrate=0.010, error=nan
>epoch=80, lrate=0.010, error=nan
>epoch=81, lrate=0.010, error=nan
>epoch=82, lrate=0.010, error=nan
>epoch=83, lrate=0.010, error=nan
>epoch=84, lrate=0.010, error=nan
>epoch=85, lrate=0.010, error=nan
>epoch=86, lrate=0.010, error=nan
>epoch=87, lrate=0.010, error=nan
>epoch=88, lrate=0.010, error=nan
>epoch=89, lrate=0.010, error=nan
>epoch=90, lrate=0.010, error=nan
>epoch=91, lrate=0.010, error=nan
>epoch=92, lrate=0.010, error=nan
>epoch=93, lrate=0.010, error=nan
>epoch=94, lrate=0.010, error=nan
>epoch=95, lrate=0.010, error=nan
>epoch=96, lrate=0.010, error=nan
>epoch=97, lrate=0.010, error=nan
>epoch=98, lrate=0.010, error=nan
>epoch=99, lrate=0.010, error=nan
>epoch=100, lrate=0.010, error=nan
>epoch=101, lrate=0.010, error=nan
>epoch=102, lrate=0.010, error=nan
>epoch=103, lrate=0.010, error=nan
>epoch=104, lrate=0.010, error=nan
>epoch=105, lrate=0.010, error=nan
>epoch=106, lrate=0.010, error=nan
>epoch=107, lrate=0.010, error=nan
>epoch=108, lrate=0.010, error=nan
>epoch=109, lrate=0.010, error=nan
>epoch=110, lrate=0.010, error=nan
>epoch=111, lrate=0.010, error=nan
>epoch=112, lrate=0.010, error=nan
>epoch=113, lrate=0.010, error=nan
>epoch=114, lrate=0.010, error=nan
>epoch=115, lrate=0.010, error=nan
```

```
>epoch=116, lrate=0.010, error=nan
>epoch=117, lrate=0.010, error=nan
>epoch=118, lrate=0.010, error=nan
>epoch=119, lrate=0.010, error=nan
>epoch=120, lrate=0.010, error=nan
>epoch=121, lrate=0.010, error=nan
>epoch=122, lrate=0.010, error=nan
>epoch=123, lrate=0.010, error=nan
>epoch=124, lrate=0.010, error=nan
>epoch=125, lrate=0.010, error=nan
>epoch=126, lrate=0.010, error=nan
>epoch=127, lrate=0.010, error=nan
>epoch=128, lrate=0.010, error=nan
>epoch=129, lrate=0.010, error=nan
>epoch=130, lrate=0.010, error=nan
>epoch=131, lrate=0.010, error=nan
>epoch=132, lrate=0.010, error=nan
>epoch=133, lrate=0.010, error=nan
>epoch=134, lrate=0.010, error=nan
>epoch=135, lrate=0.010, error=nan
>epoch=136, lrate=0.010, error=nan
>epoch=137, lrate=0.010, error=nan
>epoch=138, lrate=0.010, error=nan
>epoch=139, lrate=0.010, error=nan
>epoch=140, lrate=0.010, error=nan
>epoch=141, lrate=0.010, error=nan
>epoch=142, lrate=0.010, error=nan
>epoch=143, lrate=0.010, error=nan
>epoch=144, lrate=0.010, error=nan
>epoch=145, lrate=0.010, error=nan
>epoch=146, lrate=0.010, error=nan
>epoch=147, lrate=0.010, error=nan
>epoch=148, lrate=0.010, error=nan
>epoch=149, lrate=0.010, error=nan
>epoch=150, lrate=0.010, error=nan
>epoch=151, lrate=0.010, error=nan
>epoch=152, lrate=0.010, error=nan
>epoch=153, lrate=0.010, error=nan
>epoch=154, lrate=0.010, error=nan
>epoch=155, lrate=0.010, error=nan
>epoch=156, lrate=0.010, error=nan
>epoch=157, lrate=0.010, error=nan
>epoch=158, lrate=0.010, error=nan
>epoch=159, lrate=0.010, error=nan
>epoch=160, lrate=0.010, error=nan
>epoch=161, lrate=0.010, error=nan
>epoch=162, lrate=0.010, error=nan
>epoch=163, lrate=0.010, error=nan
```

```
>epoch=164, lrate=0.010, error=nan
>epoch=165, lrate=0.010, error=nan
>epoch=166, lrate=0.010, error=nan
>epoch=167, lrate=0.010, error=nan
>epoch=168, lrate=0.010, error=nan
>epoch=169, lrate=0.010, error=nan
>epoch=170, lrate=0.010, error=nan
>epoch=171, lrate=0.010, error=nan
>epoch=172, lrate=0.010, error=nan
>epoch=173, lrate=0.010, error=nan
>epoch=174, lrate=0.010, error=nan
>epoch=175, lrate=0.010, error=nan
>epoch=176, lrate=0.010, error=nan
>epoch=177, lrate=0.010, error=nan
>epoch=178, lrate=0.010, error=nan
>epoch=179, lrate=0.010, error=nan
>epoch=180, lrate=0.010, error=nan
>epoch=181, lrate=0.010, error=nan
>epoch=182, lrate=0.010, error=nan
>epoch=183, lrate=0.010, error=nan
>epoch=184, lrate=0.010, error=nan
>epoch=185, lrate=0.010, error=nan
>epoch=186, lrate=0.010, error=nan
>epoch=187, lrate=0.010, error=nan
>epoch=188, lrate=0.010, error=nan
>epoch=189, lrate=0.010, error=nan
>epoch=190, lrate=0.010, error=nan
>epoch=191, lrate=0.010, error=nan
>epoch=192, lrate=0.010, error=nan
>epoch=193, lrate=0.010, error=nan
>epoch=194, lrate=0.010, error=nan
>epoch=195, lrate=0.010, error=nan
>epoch=196, lrate=0.010, error=nan
>epoch=197, lrate=0.010, error=nan
>epoch=198, lrate=0.010, error=nan
>epoch=199, lrate=0.010, error=nan
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13908\3526729411.py in <module>
      8     Y_Prediction = np.append(Y_Prediction, predict(X_Test[i+1,:], coef)
      9 
---> 10 print("MSE: %.3f" % mean_squared_error(Y_Test, Y_Prediction))
     11 print("R2: %.3f" % r2_score(Y_Test, Y_Prediction))

~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py in
  mean_squared_error(y_true, y_pred, sample_weight, multioutput, squared)
```

```
    436     0.825…
    437     """
--> 438     y_type, y_true, y_pred, multioutput = _check_reg_targets(

    439         y_true, y_pred, multioutput
    440     )

~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py in␣
 ↪_check_reg_targets(y_true, y_pred, multioutput, dtype)
     94     check_consistent_length(y_true, y_pred)
     95     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
---> 96     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
     97
     98     if y_true.ndim == 1:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array,␣
 ↪accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite,␣
 ↪ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite ==␣
 ↪"allow-nan")
    801
    802     if ensure_min_samples > 0:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in␣
 ↪_assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114             raise ValueError(

    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None else X
 ↪dtype

ValueError: Input contains NaN, infinity or a value too large for␣
 ↪dtype('float64').
```

I can not for the life of me figure out how I'm getting nan for error and have tried everything. Please correct my likely obvious and blind mistake because I am at a loss.

1.3 Problem 2

[9]:
```
# train test split
XTrain, XTest, YTrain, YTest = train_test_split(dataSonar, targetSonar,␣
 ↪test_size = 0.3, random_state=3)


# Perceptron model w/ RepeatedStratifiedKFold
```

```
model = Perceptron()
model.fit(XTrain,YTrain)
repStratKFold = RepeatedStratifiedKFold(n_splits=10, n_repeats=5,
 ↪random_state=1)

# creating grid and applying on Perceptron
grid = dict()
grid["alpha"] = [0.0001, 0.001, 0.01, 0.1]
gridSearch = GridSearchCV(model, grid, scoring="accuracy",cv=repStratKFold,
 ↪n_jobs=1)

# reporting score
results = gridSearch.fit(XTrain,YTrain)
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)

# Perceptron model with best value
bestModel = Perceptron(alpha=0.0001)
bestModel.fit(XTrain,YTrain)
bestPrediction = bestModel.predict(XTest)
print('Accuracy of Perceptron Model w/ best alpha: %3f' %
 ↪accuracy_score(YTest,bestPrediction))
```

```
Mean Accuracy: 0.664
Config: {'alpha': 0.0001}
Accuracy of Perceptron Model w/ best alpha: 0.650794
```

1.4 Problem 3

1.4.1 (a)

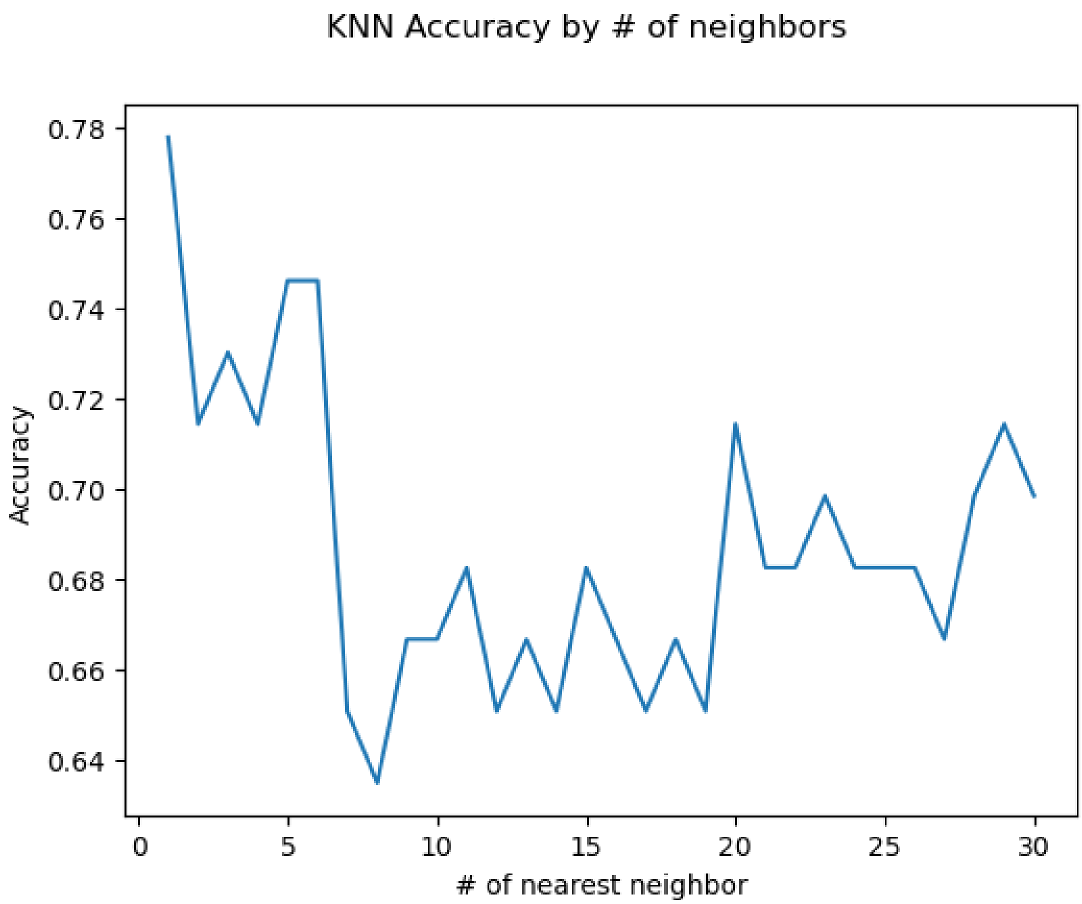
[10]:
```
# train test split
XTrain, XTest, YTrain, YTest = train_test_split(dataSonar, targetSonar,
 ↪test_size = 0.3, random_state=5)

# KNN model
kResults = {}
for k in range(1,31):
    kNeighbors = KNeighborsClassifier(n_neighbors=k)
    kNeighbors.fit(XTrain,YTrain)
    kPrediction = kNeighbors.predict(XTest)
    kResults[k] = accuracy_score(kPrediction,YTest)
kResults
```

[10]:
```
{1: 0.7777777777777778,
 2: 0.7142857142857143,
 3: 0.7301587301587301,
 4: 0.7142857142857143,
```

```
5: 0.746031746031746,
6: 0.746031746031746,
7: 0.6507936507936508,
8: 0.6349206349206349,
9: 0.6666666666666666,
10: 0.6666666666666666,
11: 0.6825396825396826,
12: 0.6507936507936508,
13: 0.6666666666666666,
14: 0.6507936507936508,
15: 0.6825396825396826,
16: 0.6666666666666666,
17: 0.6507936507936508,
18: 0.6666666666666666,
19: 0.6507936507936508,
20: 0.7142857142857143,
21: 0.6825396825396826,
22: 0.6825396825396826,
23: 0.6984126984126984,
24: 0.6825396825396826,
25: 0.6825396825396826,
26: 0.6825396825396826,
27: 0.6666666666666666,
28: 0.6984126984126984,
29: 0.7142857142857143,
30: 0.6984126984126984}
```

1.4.2 (b)

[11]:
```
# plotting KNN model
plt.plot(list(kResults.keys()),list(kResults.values()))
plt.suptitle('KNN Accuracy by # of neighbors')
plt.xlabel('# of nearest neighbor')
plt.ylabel('Accuracy')
plt.show()
```



1.4.3 (c)

[12]:
```
## KNN model using best value
kBest = KNeighborsClassifier(n_neighbors=1)
kBest.fit(XTrain,YTrain)
kBestPrediction = bestModel.predict(XTest)
print('Accuracy of KNN Model w/ best # of neighbors: %3f' %␣
↪accuracy_score(YTest,kBestPrediction))
```

```
Accuracy of KNN Model w/ best # of neighbors: 0.682540
```