Module 2 Python Exercises

July 12, 2021

1 Module 2 Python Exerises: KNN and Perceptron

1.1 Filipp Krasovsky, 7-12-2021

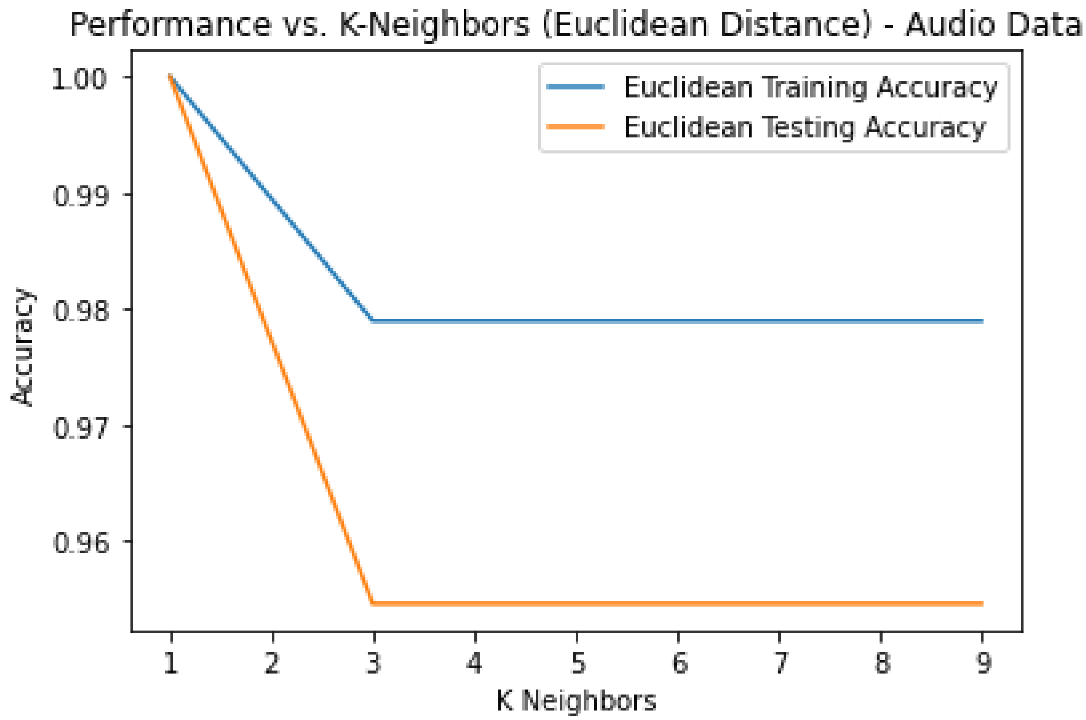
[30]:
```python
import seaborn as sns
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.preprocessing import OrdinalEncoder
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder
```

1.2 Drum Sounds Data

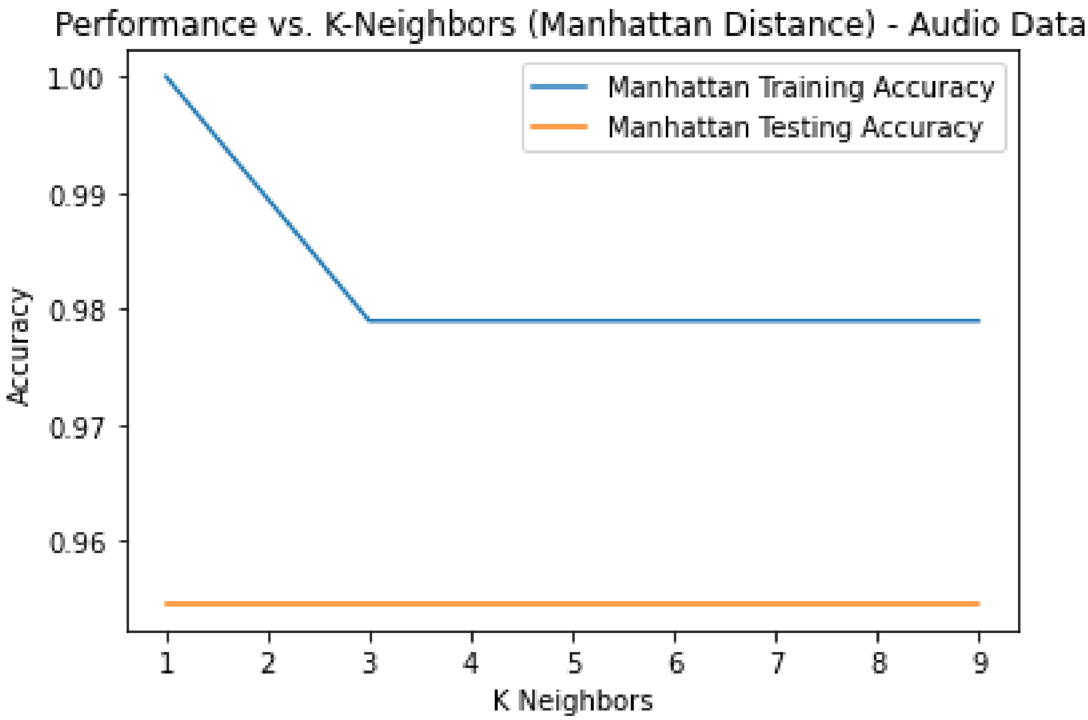
[91]:
```python
audio_data = pd.read_csv('audio_data.csv',index_col=0).
 ↪drop(['label','filename'],axis=1)

X = audio_data.drop('label_text',axis=1)

le = preprocessing.LabelEncoder()
labels = audio_data['label_text']
le.fit(labels)
y=le.transform(labels)

u_labels = le.classes_

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
 ↪random_state=42)
```

[92]:
```
#train a knn model across k = 1 to 9 and get the manhattan and euclidean␣
↪performance(s).
k_val = []
euclidean_test = []
euclidean_train= []
manhattan_test = []
manhattan_train= []

for k in range(1,10):
    if k%2==0:
        continue
    k_val.append(k)
    #train using euclidean distance
    this_model = KNeighborsClassifier(n_neighbors=k,metric="euclidean")
    this_model = this_model.fit(X_train,y_train)
    #get training accuracy
    this_train = this_model.score(X_train,y_train)
    #get testing accuracy
    this_pred  = this_model.predict(X_test)
    test_pred  = accuracy_score(this_pred,y_test)
    #append
    euclidean_test.append(test_pred)
    euclidean_train.append(this_train)

    #train using manhattan distance
    this_model_manhattan =␣
↪KNeighborsClassifier(n_neighbors=k,metric="manhattan")
    this_model_manhattan = this_model_manhattan.fit(X_train,y_train)

    #get training accuracy
    manhattan_train_accuracy = this_model_manhattan.score(X_train,y_train)

    #get testing accuracy
    this_pred_manhattan  = this_model_manhattan.predict(X_test)
    this_pred_manhattan  = accuracy_score(this_pred_manhattan,y_test)
    #append
    manhattan_test.append(this_pred_manhattan)
    manhattan_train.append(manhattan_train_accuracy)
```

[93]:
```
#plot euclidean test/train error
plt.plot(k_val, euclidean_train, label = "Euclidean Training Accuracy")
plt.plot(k_val, euclidean_test,  label = "Euclidean Testing Accuracy")
plt.legend()
plt.xlabel("K Neighbors")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Neighbors (Euclidean Distance) - Audio Data")
plt.show()
```



[94]:
```
#plot manhattan test/train error
plt.plot(k_val, manhattan_train, label = "Manhattan Training Accuracy")
plt.plot(k_val, manhattan_test,  label = "Manhattan Testing Accuracy")
plt.legend()
plt.xlabel("K Neighbors")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Neighbors (Manhattan Distance) - Audio Data")
plt.show()
```



1.3 Animal Shelter Data

[27]:
```
shelter_data = pd.read_csv('shelter_data.csv')
shelter_data.head()
```

[27]:
```
  AnimalID     Name             DateTime      OutcomeType OutcomeSubtype  \
0  A671945  Hambone  2014-02-12 18:22:00  Return_to_owner            NaN
1  A656520    Emily  2013-10-13 12:44:00       Euthanasia      Suffering
2  A686464   Pearce  2015-01-31 12:28:00         Adoption         Foster
3  A683430      NaN  2014-07-11 19:09:00         Transfer        Partner
4  A667013      NaN  2013-11-15 12:52:00         Transfer        Partner

  AnimalType SexuponOutcome AgeuponOutcome                        Breed  \
0        Dog  Neutered Male         1 year        Shetland Sheepdog Mix
1        Cat  Spayed Female         1 year       Domestic Shorthair Mix
2        Dog  Neutered Male        2 years                 Pit Bull Mix
3        Cat    Intact Male        3 weeks       Domestic Shorthair Mix
4        Dog  Neutered Male        2 years  Lhasa Apso/Miniature Poodle

         Color
0  Brown/White
1  Cream Tabby
2   Blue/White
```

```
3   Blue Cream
4          Tan
```

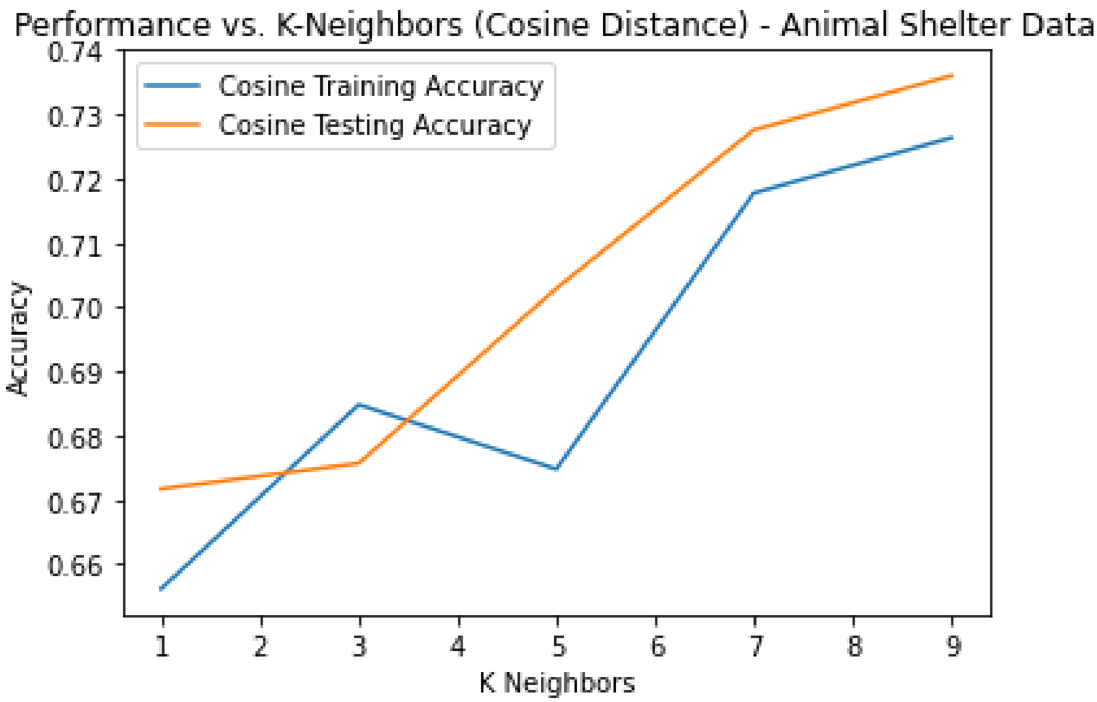
[110]:
```python
# this line drops any rows with missing data
cleaned_shelter_data = shelter_data.dropna()

# here we grab the data we want from pandas
X_data = cleaned_shelter_data[['AnimalType','SexuponOutcome','AgeuponOutcome']]
y_data = cleaned_shelter_data[['OutcomeType']]

#use the oneHotEncoder to transform our variables and get label names
ohe = OneHotEncoder(sparse='False')
feature_array = ohe.fit_transform(X_data).toarray()
feature_labels= ohe.categories_
#convert labels into one array
feature_labels = np.concatenate(feature_labels)
#combine labels and data
features = pd.DataFrame(feature_array,columns = feature_labels)

le = preprocessing.LabelEncoder()
le.fit(y_data)
y = le.transform(y_data)

X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.
 ↪10, random_state=42)
```

```
C:\Users\13234\Miniconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

[111]:
```python
#train a knn model across k = 1 to 9 and get the cosine performance for␣
 ↪training and testing.
k_val = []
cosine_train = []
cosine_test  = []
for k in range(1,10):
    if k%2==0:
        continue
    k_val.append(k)
    #train using cosine distance
    this_model = KNeighborsClassifier(n_neighbors=k,metric="cosine")
    this_model = this_model.fit(X_train,y_train)
    #get training accuracy
    this_train = this_model.score(X_train,y_train)
    #get testing accuracy
```

```
    this_pred  = this_model.predict(X_test)
    test_pred  = accuracy_score(this_pred,y_test)
    #append
    cosine_train.append(test_pred)
    cosine_test.append(this_train)
```

[112]:
```
#plot cosine test/train error
plt.plot(k_val, cosine_train, label = "Cosine Training Accuracy")
plt.plot(k_val, cosine_test,  label = "Cosine Testing Accuracy")
plt.legend()
plt.xlabel("K Neighbors")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Neighbors (Cosine Distance) - Animal Shelter Data")
plt.show()
```



2 Text Data

[113]:
```
text_data = pd.read_csv('text_data.csv',index_col=0).drop('meta_title',axis=1)
display(text_data)
```

```
  meta_author  000  10  11  13  136  13th  1648  1683  1685  …  yielding  \
0    hamilton    0   0   0   0    0     0     0     0     0  …         0
1         jay    0   0   0   0    0     0     0     0     0  …         0
2         jay    0   0   0   0    0     0     0     0     1  …         0
```

```
3          jay    0   0   0   0    0     0     0     0     0  …          0
4          jay    0   0   0   0    0     0     0     0     0  …          0
..         …  …  ..  ..  ..  …   …   …   …   …  …        …
80    hamilton    0   0   0   0    0     0     0     0     0  …          0
81    hamilton    0   0   0   0    0     0     0     0     0  …          0
82    hamilton    0   0   0   0    0     0     0     0     0  …          0
83    hamilton    0   0   0   0    1     0     0     0     0  …          0
84    hamilton    0   0   0   0    0     0     0     0     0  …          0

    yoke  yokes  york  young  yourselves  zaleucus  zeal  zealand  zealous
0      0      0     1      0           0         0     3        0        0
1      0      0     1      0           0         0     0        0        0
2      0      0     1      0           0         0     0        0        0
3      0      0     1      0           0         0     0        0        0
4      0      0     1      1           1         0     0        0        0
..   …    …    …    …         …       …   …      …     …
80     0      0     2      0           0         0     0        0        0
81     0      0     2      0           0         0     0        0        0
82     0      0     5      0           0         0     0        0        0
83     0      0     3      0           0         0     2        0        0
84     0      0     1      0           0         0     1        0        2

[85 rows x 8561 columns]
```

[115]:

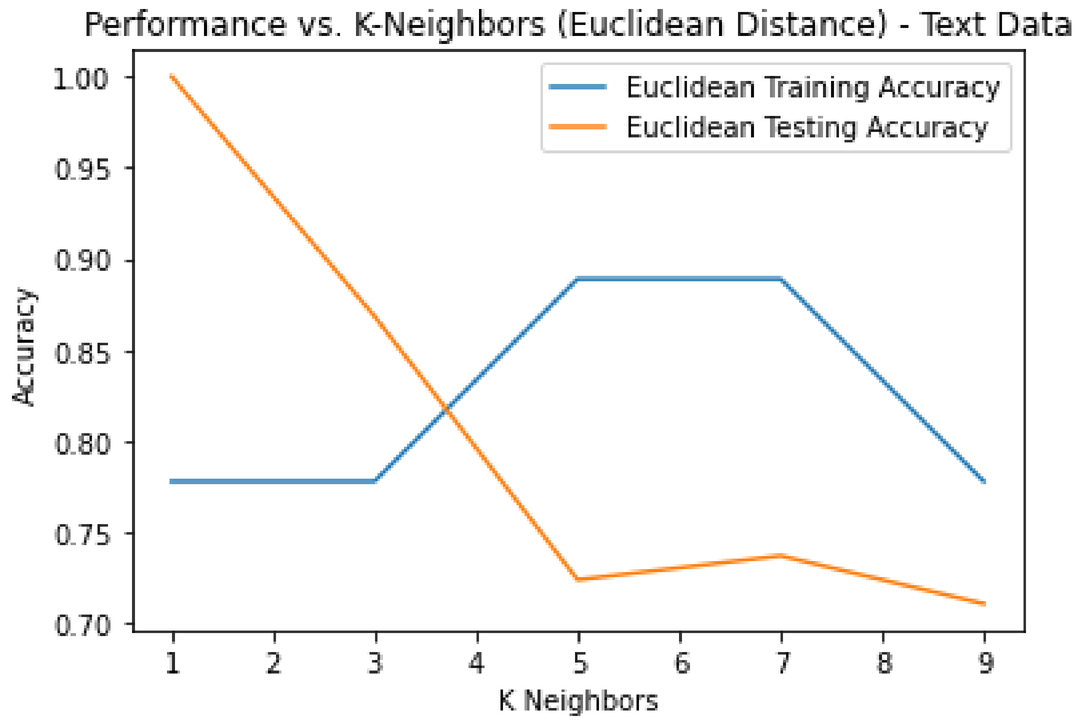
```python
#transform the output label
X = text_data.drop('meta_author',axis=1)
le = preprocessing.LabelEncoder()
labels = text_data['meta_author']
le.fit(labels)
y=le.transform(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
↪random_state=42)
```

[116]:

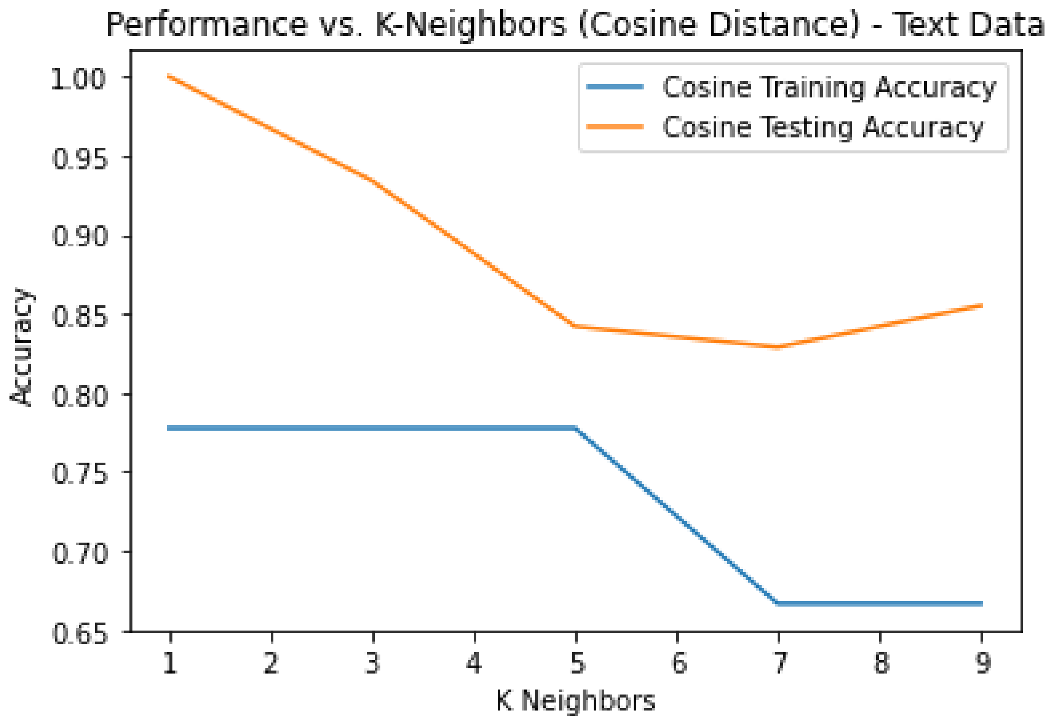
```python
#from k = 1 to 9 (odd) train a euclidean and cosine distance for performance.
euclidean_train = []
euclidean_test  = []
cosine_train    = []
cosine_test     = []
k_val = []

for k in range(1,10):
    if k%2==0:
        continue
    k_val.append(k)
    #train using cosine distance
    this_model = KNeighborsClassifier(n_neighbors=k,metric="cosine")
```

```
    this_model = this_model.fit(X_train,y_train)
    #get training accuracy
    this_train = this_model.score(X_train,y_train)
    #get testing accuracy
    this_pred  = this_model.predict(X_test)
    test_pred  = accuracy_score(this_pred,y_test)
    #append
    cosine_train.append(test_pred)
    cosine_test.append(this_train)

    #train using euclidean distance
    this_model = KNeighborsClassifier(n_neighbors=k,metric="euclidean")
    this_model = this_model.fit(X_train,y_train)
    #get training accuracy
    this_train = this_model.score(X_train,y_train)
    #get testing accuracy
    this_pred  = this_model.predict(X_test)
    test_pred  = accuracy_score(this_pred,y_test)
    #append
    euclidean_train.append(test_pred)
    euclidean_test.append(this_train)
```

[117]:
```
#plot euclidean performance for text data
plt.plot(k_val, euclidean_train, label = "Euclidean Training Accuracy")
plt.plot(k_val, euclidean_test,  label = "Euclidean Testing Accuracy")
plt.legend()
plt.xlabel("K Neighbors")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Neighbors (Euclidean Distance) - Text Data")
plt.show()
```



[118]:
```
#plot cosine performance for text data
plt.plot(k_val, cosine_train, label = "Cosine Training Accuracy")
plt.plot(k_val, cosine_test,  label = "Cosine Testing Accuracy")
plt.legend()
plt.xlabel("K Neighbors")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Neighbors (Cosine Distance) - Text Data")
plt.show()
```



3 Perceptron Implementation

begin by reloading our audio data.

[174]:
```
audio_data = pd.read_csv('audio_data.csv',index_col=0).
 ↪drop(['label','filename'],axis=1)
X = audio_data.drop('label_text',axis=1)
le = preprocessing.LabelEncoder()
labels = audio_data['label_text']
le.fit(labels)

#for the perceptron, we transform our 0 and 1 classes into -1 and 1.
y=le.transform(labels)
y[y==0] = -1

u_labels = le.classes_
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
 ↪random_state=42)
```

[183]:
```
#create a sign function for the training and testing stages.
def sign(dp):
    if (dp!=0):
        return(dp/abs(dp))
    else:
```

```
        return(0)
```

[368]:
```
#take in a set of predictors and their weight vectors and return the sign of␣
↪their dot product.
#iterate over each object in the test set, multiply it by the supplied weight␣
↪vector, and append it to a prediction vector
#after operating oer it with the sign function.

def predict_perceptron(X,w):
    predictions=[]
    for i in range(0,X.shape[0]):
        prediction = sign(np.dot(X.iloc[i].ravel(),w))
        predictions.append(prediction)
    return(predictions)
```

[351]:
```
#assumption - we require the response variable to be stored in a separate␣
↪vector y
def train_perceptron(X,y):
    #initialize a weight vector of size D (number of dimensions in X)
    D = X.shape[1]
    R = X.shape[0]
    w = np.array(np.zeros(D))
    x = np.array(np.zeros(D))

    #arbitrary max iteration
    maxIt = 100

    #iterate over each training sample. if the dot product sign != output sign,␣
↪update our weight vector.
    for i in range(1,maxIt):
        for j in range(0,R):
            this_row = np.array(X.iloc[[j]])
            this_y   = y[j]
            dp = np.dot(this_row.ravel(),w)
            a = dp

            if(a*this_y<=0):
                w = w + (this_row.ravel() * this_y)

    return(w)
```

[372]:
```
#train our perceptron model and then predict for our test set.
perceptron = train_perceptron(X_train,y_train)
p_predict  = predict_perceptron(X_test,perceptron)
p_accuracy = accuracy_score(p_predict,y_test)
print("Perceptron Accuracy: %",p_accuracy)
```

```
Perceptron Accuracy: % 0.9545454545454546
```

[374]:

```
#verify accuracy by looking at the scikit perceptron
from sklearn.linear_model import Perceptron
sk_perceptron = Perceptron()
sk_perceptron.fit(X_train,y_train)
sk_predict = sk_perceptron.predict(X_test)
print("SK Perceptron Accuracy: %",accuracy_score(sk_predict,y_test))
```

```
SK Perceptron Accuracy: % 1.0
```

Our Accuracies are within 4.5% of each other and both considerably high, so we can claim some level of success in recreating the perceptron by hand.