Ideas for KNN from Scratch

Neba Nfonsang
University of Denver

In [1]:

```
import pandas as pd
import numpy as np

from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor

from sklearn import metrics
from sklearn.model_selection import cross_val_predict, cross_val_score, Gr


from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

Read the Data

In [2]:

```
delivery_train = pd.read_csv("delivery_time_train_data.csv")
delivery_test = pd.read_csv("delivery_time_test_data.csv")
student_train = pd.read_csv("student_train_data.csv")
student_test = pd.read_csv("student_test_data.csv")
```

In [3]:

```
delivery_train.head() # view the data
```

Out[3]:

	Miles	Deliveries	Time
0	100	4	9.3
1	50	3	4.8
2	100	4	8.9
3	100	2	6.5
4	50	2	4.2

In [4]:

```
student_train.head() # view the data
```

Out[4]:

	GPA	Age	Dropped
0	3.78	21	0
1	2.38	27	0
2	3.05	21	1
3	2.19	28	1
4	3.22	23	0

Split the Data

In [5]:

```
X_delivery_train = delivery_train[["Miles", "Deliveries"]]
y_delivery_train = delivery_train[["Time"]]

X_delivery_test = delivery_test[["Miles", "Deliveries"]]
y_delivery_test = delivery_test[["Time"]]

X_student_train = student_train[["GPA", "Age"]]
y_student_train = student_train[["Dropped"]]

X_student_test = student_test[["GPA", "Age"]]
y_student_test = student_test[["Dropped"]]
```

Find the Predicted y value for the Test Instance

- We want to demonstrate how to find the predicted value of the first test instance for the delivery data.
- Later on, you will then transfer this knowlege to write a loop that grabs each instance in the test data and find it's predicted output and append that to some list variable initialized outside the for loop.

In [6]:

```
# grab the first test instance, make sure it is returned as a 2D array
np.array([X_delivery_test.iloc[0]])
```

Out[6]:

```
array([[50,  3]], dtype=int64)
```

Note that the euclidean_distances() function inside the sklearn.metrics takes two dimensional or 2D array. So, this is why we want to include another set of square brackets when slicing out the test instace, so you should use np.array([X_delivery_test.iloc[0]]) which returns a two dimensional array as in the above code instead of X_delivery_test.iloc[0] which produces a one dimensional array.

In [7]:

```
# extract test instance, for example: extract the first test instance
test_instance = np.array([X_delivery_test.iloc[0]])

# compute distances between the test instance and all training instances
d = metrics.euclidean_distances(X_delivery_train, test_instance)
d[0:5] # show just the first five distances
```

Out[7]:

```
array([[50.009999],
       [ 0.      ],
       [50.009999],
       [50.009999],
       [ 1.      ]])
```

- Note that the distance array is two dimensional, you need to flatten the distances from a two dimensional array to a one dimensinoal array so you can be able to stack the distances with the y_train.
- When you stack the distances with the y_train, you will then sort the stacked matrix by the distance column.
- Finally, you will select k number of y_train values of the k-nearest neigbors and use these y_train values to make a prediction for the test instance.
  - For a regression problem you average the y_train values of the k-nearest neighbors to obtain the predicted value of the test instance
  - For a classification problem, using the y_train values of the k-nearest neighbors, pick the class with majority votes and use that class as the predicted value. If there is a tie, randomly pick any class as the predicted value.

In [8]:

```
# stack y_train and distances, then sort the stacked array by the distance
stacked = np.stack((d.flatten(), y_delivery_train.Time.values), axis=1)
stacked[np.argsort(stacked[:,-1])] [0:10] # view the first 10
```

Out[8]:

```
array([[10.04987562,  2.8       ],
       [ 5.38516481,  3.1       ],
       [10.19803903,  3.2       ],
       [ 5.38516481,  3.5       ],
       [ 2.        ,  3.5       ],
       [10.19803903,  3.5       ],
       [ 5.09901951,  3.6       ],
       [10.19803903,  3.6       ],
       [ 1.        ,  3.6       ],
       [ 5.38516481,  3.8       ]])
```

In [9]:

```
# find the y_train values for the nearest k-neighbors
k = 5 # let's assume k=5
y_train_nearest_k = stacked[np.argsort(stacked[:,-1])][0:k,0]
y_train_nearest_k
```

Out[9]:

```
array([10.04987562,  5.38516481, 10.19803903,  5.38516481,  2.
])
```

In [10]:

```
# find the predicted value of the test instance
predicted_y = np.mean(y_train_nearest_k)
predicted_y
```

Out[10]:

```
6.603648852515093
```

In [ ]:

```
```

Sorting a Numpy Array by Column

- We want to take a look again at how sorting an array by column works!

In [11]:

```
a = np.array([[3,4],[2,10]])
a
```

Out[11]:

```
array([[ 3,  4],
       [ 2, 10]])
```

Let's sort this array by the first column

In [12]:

```
# select the column you want to sort by
col = a[:, 0] # first column

# apply np.argsort() to the column to get indices
ind = np.argsort(col)

# use the index to sort the array
a[ind]
```

Out[12]:

```
array([[ 2, 10],
       [ 3,  4]])
```

In [13]:

```
def knn_predict(X_train, y_train, X_test, k=5):
    y_pred = []
    for i in range(0, len(X_test)):
        # grab a test instance from the X_test data
        test_instance = np.array([X_test.iloc[i]])
        # find distances between the test instance and all the training in
        # stack the distances with the y_train to get a matrix
        # sort the matrix by the distance column
        # pick k number of y_train values
            ## where k is much less than the length of the training set
        # make a prediction for the test instance
        # append the predicted value to the y_pred variable
        # pick the next test instace, repeat process until the loop termin
        # return the list of y_pred corresponding to the test instances in
```

Using Pipeline

- Pipeline takes a list of (name, object) tuples

```
pipe = Pipeline([("name1", object1()), ("name2", object2())])
```

- Make sure the objects are imported from sklearn, such objects could be a StandardScalar, KnearestNeighborClassifier, etc. Use intuitive names for the object at your discretion.
- There could be as many (name, object) tuples as needed

In [ ]:

```
```