# K - Nearest Neigbhor

**Note** : All analysis now onwards has been done on Google colab for fast processing with aid of Graphic Processing Unit(GPU).

In [0]:
```
from google.colab import drive
drive.mount('/content/drive') # Note all analysis now onwards has been done on Google colab for fast processing
                              # with aid of Graphic Processing Unit(GPU).
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
import numpy as np
import pandas as pd
import random
random.seed(3)
```

In [0]:
```
df3 = pd.read_csv("/content/drive/My Drive/image classification project/df3_final.csv",header=None)  #importing data from directory
df3.head(5)
```

Out[0]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | HAM_0000118 | ISIC_0027419 | bkl | histo | 80.0 | male | scalp | 187 | 148 | 190 | 191 | 153 | 194 |
| **1** | HAM_0000118 | ISIC_0025030 | bkl | histo | 80.0 | male | scalp | 25 | 14 | 23 | 66 | 40 | 56 |
| **2** | HAM_0002730 | ISIC_0026769 | bkl | histo | 80.0 | male | scalp | 146 | 133 | 186 | 157 | 145 | 198 |
| **3** | HAM_0002730 | ISIC_0025661 | bkl | histo | 80.0 | male | scalp | 27 | 16 | 31 | 70 | 55 | 86 |
| **4** | HAM_0001466 | ISIC_0031633 | bkl | histo | 75.0 | male | ear | 134 | 110 | 153 | 171 | 142 | 188 |

5 rows × 3080 columns

In [0]:
```
df3.tail(5)
```

Out[0]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10010** | HAM_0002867 | ISIC_0033084 | akiec | histo | 40.0 | male | abdomen | 196 | 177 | 192 | 195 |
| **10011** | HAM_0002867 | ISIC_0033550 | akiec | histo | 40.0 | male | abdomen | 3 | 7 | 5 | 14 |
| **10012** | HAM_0002867 | ISIC_0033536 | akiec | histo | 40.0 | male | abdomen | 130 | 125 | 132 | 137 |
| **10013** | HAM_0000239 | ISIC_0032854 | akiec | histo | 80.0 | male | face | 145 | 123 | 159 | 152 |
| **10014** | HAM_0003521 | ISIC_0032258 | mel | histo | 70.0 | female | back | 119 | 140 | 173 | 130 |

5 rows × 3080 columns

In [0]:
```
X = df3.iloc[:,7:3079].values
y = df3.iloc[:, 3079].values
```

In [0]:
```
from sklearn.model_selection import train_test_split
```

In [0]:
```
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.30)
```

In [0]:
```
Xtr
```

Out[0]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [161, 154, 189, ..., 149, 143, 175],
       [152, 147, 220, ..., 133, 139, 209],
       ...,
       [198, 175, 219, ..., 168, 158, 209],
       [223, 193, 234, ..., 194, 169, 207],
       [172, 162, 183, ..., 173, 164, 185]])
```

In [0]:
```
Xte
```

Out[0]:
```
array([[162, 162, 241, ..., 148, 153, 219],
       [157, 138, 229, ..., 136, 138, 212],
       [201, 181, 205, ..., 144, 144, 174],
       ...,
       [151, 137, 215, ..., 143, 127, 202],
       [169, 137, 224, ..., 163, 160, 224],
       [161, 171, 238, ..., 154, 168, 226]])
```

In [0]:
```
ytr
```

Out[0]:
```
array([5, 4, 5, ..., 2, 3, 5])
```

In [0]:
```
yte
```

Out[0]:
```
array([5, 5, 1, ..., 5, 5, 5])
```

In [0]:
```
Xtr_rows = Xtr.reshape(Xtr.shape[0], 32 * 32 * 3)
Xte_rows = Xte.reshape(Xte.shape[0], 32 * 32 * 3)
```

In [0]:
```
Xtr_rows
```

Out[0]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [161, 154, 189, ..., 149, 143, 175],
       [152, 147, 220, ..., 133, 139, 209],
       ...,
       [198, 175, 219, ..., 168, 158, 209],
       [223, 193, 234, ..., 194, 169, 207],
       [172, 162, 183, ..., 173, 164, 185]])
```

In [0]:
```
class NearestNeighbor(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

In [0]:
```
nn = NearestNeighbor() # create a Nearest Neighbor classifier class
nn.train(Xtr_rows, ytr) # train the classifier on the training images and labels
Yte_predict = nn.predict(Xte_rows) # predict labels on the test images
# and now print the classification accuracy, which is the average number
# of examples that are correctly predicted (i.e. label matches)
print('accuracy: %f' % ( np.mean(Yte_predict == yte) ))
```

```
accuracy: 0.673877
```

In [0]:
```
Accuracy=np.mean(Yte_predict == yte)
Accuracy
```

Out[0]:
```
0.6738768718801996
```

In [0]:
```
# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

results = confusion_matrix(yte, Yte_predict)

print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(yte, Yte_predict))
print ('Report : ',classification_report(yte, Yte_predict))
```

```
Confusion Matrix :
[[  22   11   18    4    2   45    3]
 [   5   46   21    6    6   55    7]
 [   6   22   98    8   23  160    6]
 [   0    7    7    4    0   16    1]
 [   7    7   51    3   61  190    4]
 [   9   27   85    6   84 1789   37]
 [   0    2    3    1    2   23    5]]
Accuracy Score : 0.6738768718801996
Report :               precision    recall  f1-score   support

           0       0.45      0.21      0.29       105
           1       0.38      0.32      0.34       146
           2       0.35      0.30      0.32       323
           3       0.12      0.11      0.12        35
           4       0.34      0.19      0.24       323
           5       0.79      0.88      0.83      2037
           6       0.08      0.14      0.10        36

    accuracy                           0.67      3005
   macro avg       0.36      0.31      0.32      3005
weighted avg       0.64      0.67      0.65      3005
```

In [0]:
```
class NearestNeighbor1(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of s
ize N """
    # the nearest neighbor classifier simply remembers all the trainin
g data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict lab
el for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L2 distance (sqaured sum of absolute value differenc
es)
      distances = np.sqrt(np.sum(np.square(self.Xtr - X[i,:]), axis =
1))
      min_index = np.argmin(distances) # get the index with smallest d
istance
      Ypred[i] = self.ytr[min_index] # predict the label of the neares
t example

    return Ypred
```

In [0]:
```
nn1 = NearestNeighbor1() # create a Nearest Neighbor classifier class
nn1.train(Xtr_rows, ytr) # train the classifier on the training images
and labels
Yte_predict1 = nn1.predict(Xte_rows) # predict labels on the test imag
es
# and now print the classification accuracy, which is the average numb
er
# of examples that are correctly predicted (i.e. label matches)
print('accuracy: %f' % ( np.mean(Yte_predict1 == yte) ))
```

```
accuracy: 0.684193
```

In [0]:
```
Accuracy1=np.mean(Yte_predict1 == yte)
Accuracy1
```

Out[0]:
```
0.6841930116472545
```

In [0]:
```
results1 = confusion_matrix(yte, Yte_predict1)

print ('Confusion Matrix :')
print(results1)

print ('Accuracy Score :',accuracy_score(yte, Yte_predict1))
print ('Report : ',classification_report(yte, Yte_predict1))
```

```
Confusion Matrix :
[[  21   13   19    5    4   40    3]
 [   7   36   28    6    2   59    8]
 [   5   20  118   11   22  144    3]
 [   2    3    5    6    0   19    0]
 [   8    7   46    2   69  189    2]
 [   5   18  102    6   77 1802   27]
 [   0    1    6    0    3   22    4]]
Accuracy Score : 0.6841930116472545
Report :               precision    recall  f1-score   support

           0       0.44      0.20      0.27       105
           1       0.37      0.25      0.30       146
           2       0.36      0.37      0.36       323
           3       0.17      0.17      0.17        35
           4       0.39      0.21      0.28       323
           5       0.79      0.88      0.84      2037
           6       0.09      0.11      0.10        36

    accuracy                           0.68      3005
   macro avg       0.37      0.31      0.33      3005
weighted avg       0.65      0.68      0.66      3005
```