naive-bayes

November 22, 2017

In [3]:
```
import numpy as np
from scipy.stats import iqr
from sklearn.datasets import load_digits
```

In [4]:
```
digits=load_digits()
print(digits.keys())
data=digits['data']
images=digits['images']
target=digits['target']
target_names=digits['target_names']
```

```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

In [5]:
```
def filter_numbers(number1,number2):
    """Returns subset of digits sample data containing only the numbers passed to the fu

       Takes number1 and number2 of the target_names as its arguments
       Returns two arrays: data, target
    """
    #if number1 and number2:
    if True:
        indices = np.where((digits['target'] == number1) | (digits['target'] == number2)
        #print(indices)
        data = (digits['data'])[indices]
        target = (digits['target'])[indices]
        #print(indices)
    return data, target

def reduced_dim(x):
    res=np.empty([np.shape(x)[0],2])
    for i in range(len(x)):
        # Define the features here:

        res[i][0]=x[i][60]# Feature 1
        res[i][1]=x[i][19]          # Feature 2

    return res
```

```
In [185]: def fit_naive_bayes(features,labels, bincount=0):
              """ calculates histograms for each of the D feature dimensions

                  If calles with bincount = 0, calculates number of bins by Freedman-Diaconis

                  returns
                  histograms - C x D x L array (C = #classes, D = #feature dimensions, L 0 #bins
                  bincount - C x D x 2 array (last dimensions: [lower bound of bin, bin width] )

              """
              class0, class1 = features[np.where(labels==0)],features[np.where(labels==1)]

              if (bincount == 0):
                  ## suggested bin width by Freedman Diaconis
                  IQR_0 = iqr(class0,axis=0)
                  IQR_1 = iqr(class1,axis=0)

                  binwidth_0 = 2 * IQR_0 / (len(class0))**(1/3)
                  binwidth_1 = 2 * IQR_1 / (len(class1))**(1/3)

                  #calculate number of bins; exclude feature dimensions where IQR is 0
                  bins_0 = np.ceil((np.max(class0[:,np.where(binwidth_0 != 0)],axis=0)-np.min(cl
                  bins_0_ = np.zeros((features.shape[1]))
                  bins_0_[np.where(binwidth_0 != 0)] = bins_0

                  bins_1 = np.ceil((np.max(class1[:,np.where(binwidth_1 != 0)],axis=0)-np.min(cl
                  bins_1_ = np.zeros((features.shape[1]))
                  bins_1_[np.where(binwidth_1 != 0)] = bins_1

                  bincount = np.round(np.mean([bins_0,bins_1])) # set bin counts as averaged num

              binwidth0 = np.ceil((np.max(class0,axis=0)-np.min(class0,axis=0))/bincount)
              binwidth1 = np.ceil((np.max(class1,axis=0)-np.min(class1,axis=0))/bincount)


              histograms = np.zeros((2,features.shape[1],int(bincount)))
              binning = np.ones((2,features.shape[1],2))

              binning[0,:,0] = np.min(class0,axis=0) # lower bound of first bin
              binning[0,np.where(bins_0_ >= bincount),1] = binwidth0[np.where(bins_0_ >= bincoun


              binning[1,:,0] = np.min(class1,axis=0)
              binning[1,np.where(bins_1_ >= bincount),1] = binwidth1[np.where(bins_1_ >= bincoun

              ### fill bins
              def frequency(k,i):
                  """
```

```
                """Bins feature dimension i"""
                class_ = class0 if k == 0 else class1

                unique, counts = np.unique(np.floor((class_[:,i] - binning[k,i,0])/binning[k,i
                return unique, counts
            for k in [0,1]:
                for i in range(features.shape[1]):
                    bin_, counts = frequency(k,i)
                    for m in range(len(bin_)):
                        histograms[k,i,int(m)] = counts[m]
            return histograms, binning
```

```
In [186]: feature39, label39 = filter_numbers(3,9)
          label39[label39 == 3] = 0
          label39[label39 == 9] = 1
          histograms, binning = fit_naive_bayes(feature39, label39)
          print(histograms.shape)
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-186-19cde3feaff0> in <module>()
  2 label39[label39 == 3] = 0
  3 label39[label39 == 9] = 1
----> 4 histograms, binning = fit_naive_bayes(feature39, label39)
  5 print(histograms.shape)


<ipython-input-185-030afa5bef9b> in fit_naive_bayes(features, labels, bincount)
 56             bin_, counts = frequency(k,i)
 57             for m in range(len(bin_)):
---> 58                 histograms[k,i,int(m)] = counts[m]
 59     return histograms, binning


IndexError: index 11 is out of bounds for axis 2 with size 11
```

The program is failing when it comes to determining the counts of each individual bin. The problem is that the minimum bin width which is set to 1 for feature dimensions, where the suggested number of bins is less than the average, seems to be too low so that outliers count into bins of the full spectrum of the greyscale (up to 17 bins). As it is quite likely that the IQR and thus the recommended bin width after Freedman-Diaconis' rule is 0, we need to investigate a better way of treating outliers.