ENGR421
Homework 3
Ata Sayın, 64437
23.11.2020

Naïve Bayes' Classifier

Introduction

In the homework, a Naïve Bayes' Classifier is implemented for discrimination of handwritten letters-A, B, C, D, E. The picture had 20 pixels x 16 pixels which makes 320 pixels for the input. The labels are imported as strings, "A" i.e.. Each letters has 39 samples whose 25 samples are the train samples and 14 are the test samples.

Importing Data(same as HW2)

"hw02_data_set_images.csv:" file has the inputs and "hw02_data_set_labels.csv:" has labels for the input. numpy genfromtxt function is used for importing and the y labels are imported with dtype string for correct importation.

```
data_set_x=np.genfromtxt("hw02_data_set_images.csv",delimiter=",")
data_set_y=np.genfromtxt("hw02_data_set_labels.csv",delimiter=",",dtype=np.str )
```

Since string are impractical for mathematical calculations, the label must be change to 1-2-3-4-5 respectively. Pandas' DataFrame and sklearns LabelEncoder allows the change.

```
df=pd.DataFrame(data_set_y)
labelencoder = LabelEncoder()
df = labelencoder.fit_transform(df)
data_set_y=np.array(df)+1
data_set_y
```

After importing the data, the data must be arranged for calculations. D is the amount of the inputs' dimensions which is the number of pixels in this case (320). K is equal to 5 for number of classes. A-B-C-D-E. xtrain has 25 samples for each class, xtest has 14 samples. The for loops allows the capture correct samples. The label are organized by one hot encoding for matrix multiplications.

```
ntrain=int(data_set_x.shape[0]*25/39)
ntest=int(data_set_x.shape[0]*14/39)
D=data_set_x.shape[1]
K=5
```

```
xtrain=np.vstack([data_set_x[39*i:25+39*i] for i in range(K)])
xtest=np.vstack([data_set_x[25+39*i:39+39*i] for i in range(K)])
xtrain.shape
```

```
(125, 320)
```

```
ytrain=np.concatenate([data_set_y[39*i:25+39*i] for i in range(K)])
ytest=np.concatenate([data_set_y[25+39*i:39+39*i] for i in range(K)])
ytrain.shape
```

```
(125,)
```

```
Y_train = np.zeros((ntrain, K)).astype(int)
Y_train[range(ntrain), ytrain - 1] = 1

Y_test = np.zeros((ntest, K)).astype(int)
Y_test[range(ntest), ytest - 1] = 1
Y_test.shape
```

```
(70, 5)
```

Implementation of NBC

The distribution of the pixel points is claimed as a Bernoulli Distribution therefore their p values were the mean of the pixels which has the same location in the image.

```
pdc=np.stack([np.mean(xtrain[i*25:25+25*i],axis=0) for i in range(K)])
print(pdc[0,])
print("-----")
print(pdc[1,])
print("-----")
print(pdc[2,])
print("-----")
print(pdc[3,])
print("-----")
print(pdc[4,])
print("-----")
```

The prior's values for each class is founded by the mean of the one-hot matrix labels.

```
]: class_priors_train=np.mean(Y_train,axis=0)
   class_priors_train

]: array([0.2, 0.2, 0.2, 0.2, 0.2])

]: class_priors_test=np.mean(Y_test,axis=0)
   class_priors_test

]: array([0.2, 0.2, 0.2, 0.2, 0.2])
```

The letters were imported as 320x1 arrays, for plotting purposes, 320x1 array is transformed to 16x20 array, and is taken transformed for a vertical image. Cmap binary has spectrums where 0's are white and 1' are black.

Ploting PCD's

```
]: fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5)
   ax1.imshow(pdc[0,].reshape(16,20).T,cmap='binary')
   ax1.axis('off')
   ax2.imshow(pdc[1,].reshape(16,20).T,cmap='binary')
   ax2.axis('off')
   ax3.imshow(pdc[2,].reshape(16,20).T,cmap='binary')
   ax3.axis('off')
   ax4.imshow(pdc[3,].reshape(16,20).T,cmap='binary')
   ax4.axis('off')
   ax5.imshow(pdc[4,].reshape(16,20).T,cmap='binary')
   ax5.axis('off')
   plt.show()
```

In this case, Bernoulli distribution is used therefore the score values of each class is implemented as following and then the maximum value is stated as the label of the class in the confusion matrices for train and test. Safelog is used for when log goes to infinity while taking zeros as input.

$$\begin{aligned} g_i(\boldsymbol{x}) &= \log p(\boldsymbol{x}|C_i) + \log P(C_i) \\ &= \sum_j \left[x_j \log p_{ij} + (1 - x_j) \log(1 - p_{ij}) \right] + \log P(C_i) \end{aligned}$$

Naive Bayes Classifier

```
p0=safelog(pdc[0,])
p1=safelog(pdc[1,])
p2=safelog(pdc[2,])
p3=safelog(pdc[3,])
p4=safelog(pdc[4,])

p=np.array([p0,p1,p2,p3,p4]).T
n_p=np.array([safelog(1-pdc[0,]),safelog(1-pdc[1,]),safelog(1-pdc[2,]),safelog(1-pdc[3,]),safelog(1-pdc
[4,])]).T
```

```
y_hat=np.matmul(xtrain,p)+np.matmul(1-xtrain,n_p)+np.array(np.repeat(class_priors_train[None,:],125,axi
s=0))
y_hat.shape
```

```
(125, 5)
```

The training data has a better accuracy against test data in the confusion matrices.

```
y_predicted = np.argmax(y_hat, axis = 1) + 1
confusion_matrix = pd.crosstab(y_predicted, ytrain, rownames = ['y_pred'], colnames = ['y_train'])
print(confusion_matrix)
```

```
y_train   1   2   3   4   5
y_pred
1        25   0   0   0   0
2         0  24   1   0   1
3         0   0  24   0   0
4         0   1   0  25   0
5         0   0   0   0  24
```

```
y_hat=np.matmul(xtest,p)+np.matmul(1-xtest,n_p)+np.array(np.repeat(class_priors_test[None,:],70,axis=0
))
y_hat.shape
```

```
(70, 5)
```

```
y_predicted = np.argmax(y_hat, axis = 1) + 1
confusion_matrix = pd.crosstab(y_predicted, ytest, rownames = ['y_pred'], colnames = ['y_train'])
print(confusion_matrix)
```

```
y_train  1   2  3   4   5
y_pred
1        7   0  0   0   0
2        0  11  3   2   4
3        0   0  7   0   0
4        7   3  3  12   0
5        0   0  1   0  10
```