University of Tehran
College of Engineering
School of Electrical and Computer Engineering

Intelligent Systems

Dr.Hosseini

Homework 5

Soroush Mesforush Mashhad

SN:810198472

Dey 01

Contents

Abstract

In this project we shall begin with an analysis on the secretary problem and solving it via python.

In the next section we shall answer and analyze a question about **Naive Bayes**, then we shall find the mean and variance parameters using the **Maximum Likelihood Estimation** for i.i.d variables.

In the last section we shall use the **Naive Bayes** classifier to find the accuracy of recognizing numbers on the validation images.

1 Question 1 : Statistical Analysis (Implementation)

1.1 Part 1

1.1.1 Part A

In this section we shall study the **Secretary problem** and implement it for different values of k AKA rejected candidates, this has been done via the following functions.

```
def PreliminaryCandidates(n):
#Array of candidates
Cand = np.arange(0,n)
#Shuffling the candidates for interviews
random.shuffle(Cand)
return Cand

```

```
def SortandHireCandidates(Cand,k):
Others = Cand[k:]
BestNotHired = np.min(Cand[:k])
BetterCandidates = np.array([])
for i in range(0,len(Cand)-k):
if Others[i]<BestNotHired:
BetterCandidates=np.append(BetterCandidates,Others[i])
if len(BetterCandidates)!=0:
return BetterCandidates[0]
else:
return Cand[-1]

```

```
def Simulationandprob(n,numberoftests):
BestChoice = []
for k in range(5, n+1, 5):
Simulation=np.array([])
for i in range(numberoftests):
Simulation = np.append(Simulation,SortandHireCandidates(
 PreliminaryCandidates(n),k))
BestChoice.append(np.histogram(Simulation, bins=n)[0][0]/
 numberoftests)
x = BestChoice
return x

```

It is important to note that due to what we learned in the Engineering Probability and Statistics course, we can obtain the probability by dividing the **np.histogram** output by the number of trials. The rest of the functions are as follows.

```
def Plot(BestChoice,n,dimensions,xlabel,title):
plt.figure(figsize=(dimensions[0], dimensions[1]))
PlotN = range(5, n+1, 5)
plt.scatter(PlotN, BestChoice)
plt.xticks(np.arange(0, n+1, 5))
plt.xlabel(xlabel)
plt.title(title)
plt.ylabel('Probability to choose best applicant')
plt.grid(True)
plt.show()

```

The Code to test and the results are as follows.

```
n = 100
numberoftests = 10000
BestChoice = Simulationandprob(n,numberoftests)
print(BestChoice)
dimensions=np.array([15,10])
xlabel = 'Number of Rejected Candidates'
title = 'Best Hiring Probability Plot'
Plot(BestChoice,n,dimensions,xlabel,title)
#Now we shall depict the best K value.
BestK = (np.argmax(BestChoice)+1)*5
print(BestK)

```

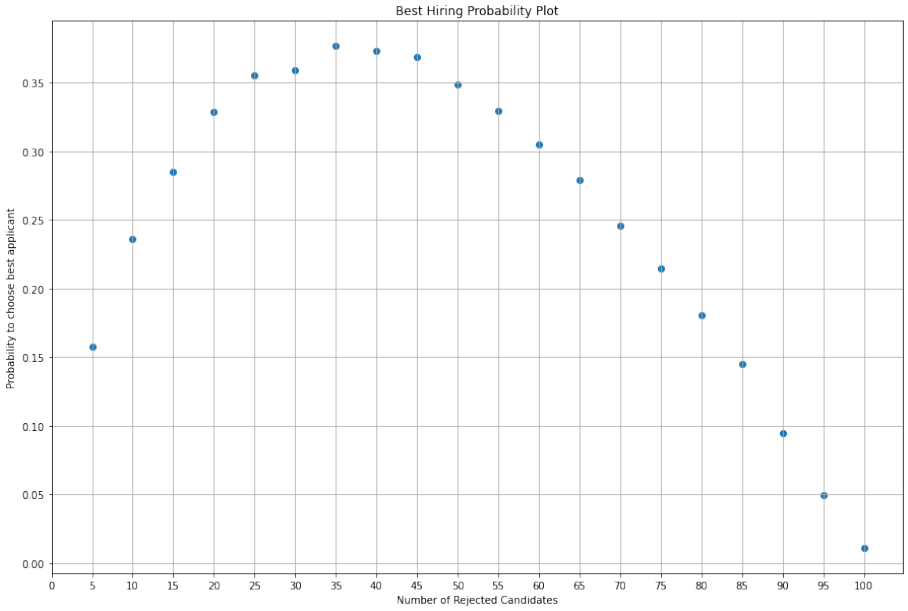

Figure 1: Best hiring probability plot

$$\text{Best K} = 35 \approx \frac{n}{e}$$

1.1.2 Part B

Here we shall perform our analysis for $k = \frac{n}{e}$ and multiple number of initial candidates accordingly, the codes are as follows.

```
def SimulationandprobPartB(n,numberoftests):
BestChoiceB = []
for n in range(3,n+1):
SimulationB=np.array([])
for i in range(numberoftests):
SimulationB = np.append(SimulationB,SortandHireCandidates(
  PreliminaryCandidates(n),int(n/np.exp(1))))
BestChoiceB.append(np.histogram(SimulationB, bins=n-2)
  [0][0]/2*numberoftests)
y=BestChoiceB
return y

```

```
def PlotB(BestChoice,n,dimensions,xlabel,title):
plt.figure(figsize=(dimensions[0], dimensions[1]))
PlotN2 = range(3,n+1)
BestChoice = np.array(BestChoice)
BestChoice = BestChoice/10**8
list(BestChoice)
plt.scatter(PlotN2, BestChoice)
plt.xticks(np.arange(3, n+1, 1))
plt.xlabel(xlabel)
plt.title(title)
plt.ylabel('Probability to choose best applicant')
plt.grid(True)
plt.show()

```

The code to test and results are as below.

```
n = 100
numberoftests = 10000
BestChoiceB = SimulationandprobPartB(n, numberoftests)
dimensionsB=np.array([30,15])
xlabelB = 'Number of Candidates'
titleB = 'Best Hiring Probability Plot'
PlotB(BestChoiceB,n,dimensionsB,xlabelB,titleB)

```

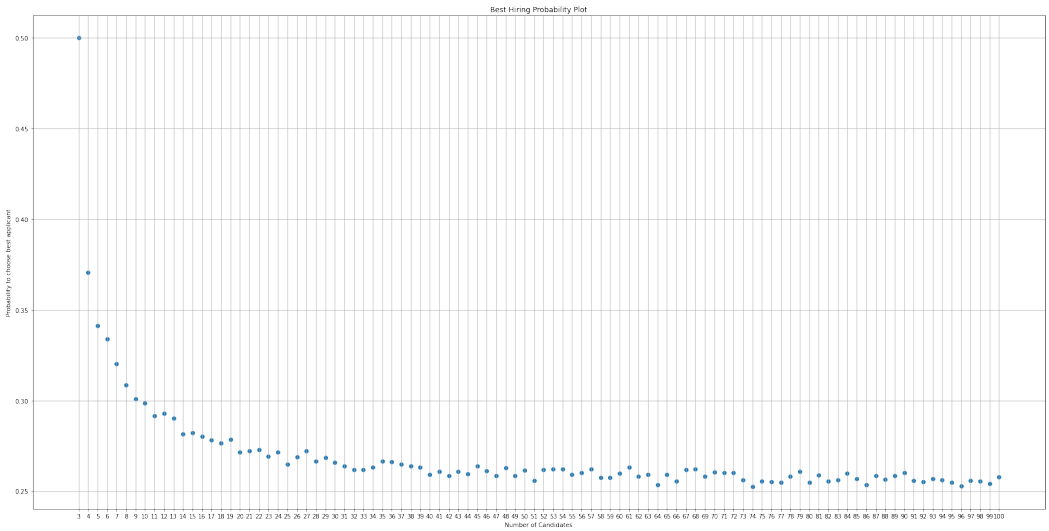

Figure 2: Best hiring probability plot

1.1.3 Part C (Bonus)

Here we shall attempt to solve the secretary problem analytically, it is important to note that we are solving the simple and standard version of this problem.

Some would go for a random approach to pick the secretary which is

foolish, due to the fact that the success probability for N candidates shall be $\frac{1}{N}$ which is very undesirable.

Here, we propose the strategy to scan through and reject the first l candidates and choose the first best one after these rejected candidates.

For this method to work, it is important that the best candidate is not located in the first l rejected candidates. To make it more simple if we consider the intervals $[1, l]$ and $[1, n]$ the maximum in both must be the same, the probability for this to happen shall be $\frac{l}{n}\frac{1}{N}$.

$$P(S) = \frac{1}{N}\left(\frac{l}{l} + \frac{l}{l+1} + \cdots + \frac{l}{N-1}\right) \longrightarrow P(S) = \frac{l}{N}\sum_{n=l}^{N-1}\frac{1}{n}$$

One might ask themselves what to do after obtaining this summation, to fix this problem we need the to study the Riemann sum.

1.1.4 Definite intrgral as Riemann sum

We saw in the **Calculus I** course the the definite integral of a continuous function such as $f(x)$ on the interval $[a, b]$ is the limit of a Riemann sum as below.

$$\int_a^b f(x)dx = \lim_{n\to\infty}\sum_{i=1}^{n}\Delta x.f(x_i), \quad \Delta x = \frac{b-a}{n}, \quad x_i = a + \Delta x.i$$

So in the case of our problem we have the following deductions.

$$P(S) = \frac{l}{N}\sum_{n=l}^{N-1}\frac{1}{n} = \lim_{N\to\infty}\frac{l}{N}\sum_{n=l}^{N-1}\frac{1}{N}\frac{N}{n} = x\int_x^1\frac{1}{t}dt$$

$$P(S) = x\int_x^1\frac{1}{t}dt = -x\ln(x), \quad \text{Optimum}: \frac{\partial}{\partial x}P(S) = 0,$$

$$-1 - \ln(x) = 0 \longrightarrow x = e^{-1}$$

So all in all, the possibility to choose the best candidate shall be:

$$P(S) = P(e^{-1}) = 0.36787944117 \approx 36.79\%$$

1.2 Part 2

Here we begin with calculating the mean of a population with the size of 100000, which has been sampled for n samples. We do this s times and plot the results accordingly.

```
def SamplerAndAVG(Population,n):
Population=list(Population)
Sampled = random.sample(Population,n)
np.array(Sampled)
Average = np.mean(Sampled)
return Average

```

```
def SamplerAndSTD(Population,n):
Population=list(Population)
Sampled = random.sample(Population,n)
np.array(Sampled)
STD = np.std(Sampled)
return STD

```

```
n=10000
s=5000
Avg =[]
for i in range(s):
Avg.append(SamplerAndAVG(Population,n))

```

The code to plot and result are as follows.

```
plt.figure(figsize=(15,8))
ax = plt.subplot(111)
ax.set_ylabel('');
ax.set_xlabel('');
ax.hist(Avg, bins=100);
ax.grid()

```

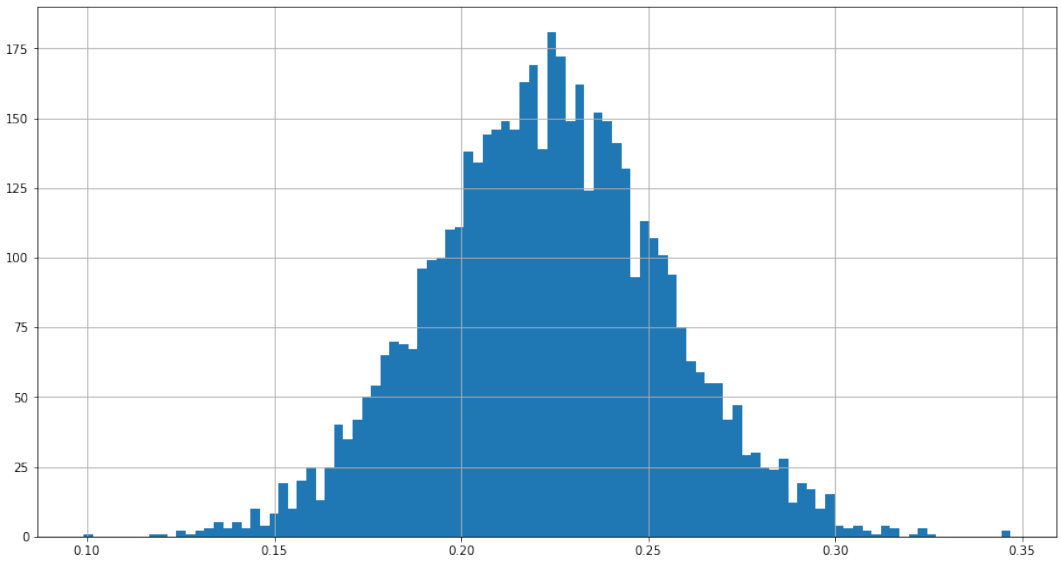

Figure 3: Plot of mean of sampled population

Next we shall go on to open the **wine.csv** dataset, we firstly plot the distribution function of the 12^{th} column of this dataset as follows.

```
data = pd.read_csv('wine.csv')
data

```

```
plt.figure(figsize=(15,8))
ax = plt.subplot(111)
ax.set_ylabel('')
ax.set_xlabel('')
ax.hist(Column12, bins=100)
ax.grid()

```

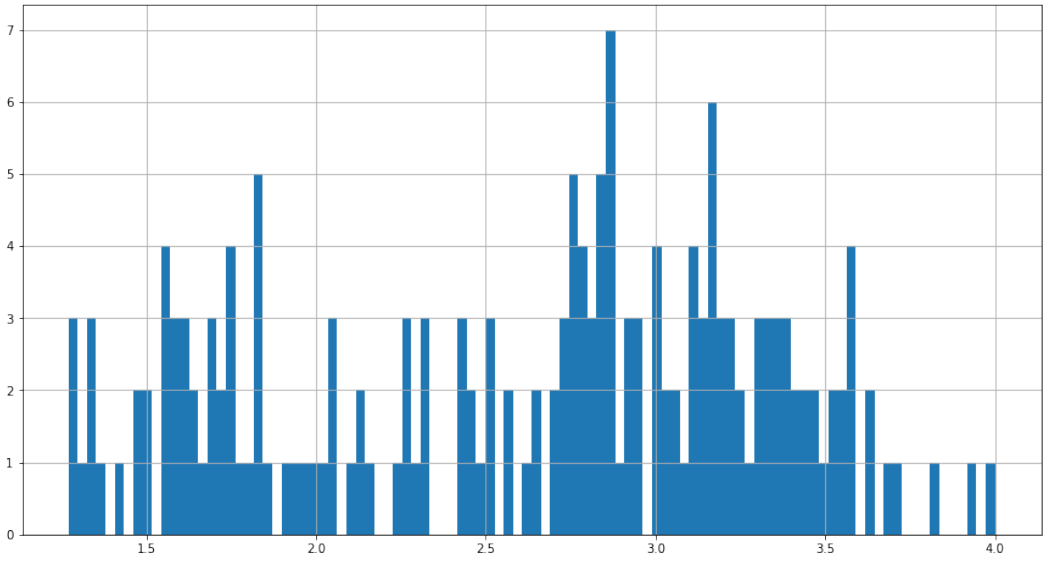

Figure 4: Column 12 distribution

Next we shall perform sampling on the different n's and compare the mean and standard deviation with the real ones.

The real mean and standard deviation is as follow.

$$\mu_{real} = 2.6116853932584267, \quad \sigma_{real} = 0.7079932646716005$$

The results for different n values are as follows.

N = 100

For this scenario we have the following.

```
n1=100
s=5000
AvgWine1 =[]
STDWine1=[]
for i in range(s):
AvgWine1.append(SamplerAndAVG(Column12,n1))
STDWine1.append(SamplerAndSTD(Column12,n1))

```

```
np.mean(AvgWine1)
2.61266356
np.mean(STDWine1)
0.7057141581407875

```

We have the following mean and standard deviation.

$$\mu_{n=100} = 2.61266356, \quad \sigma_{n=100} = 0.7057141581407875$$

The code to plot and result is as follows.

```
plt.figure(figsize=(15,8))
ax = plt.subplot(111)
ax.set_ylabel('');
ax.set_xlabel('');
ax.hist(AvgWine1,bins=100);
ax.grid()

```

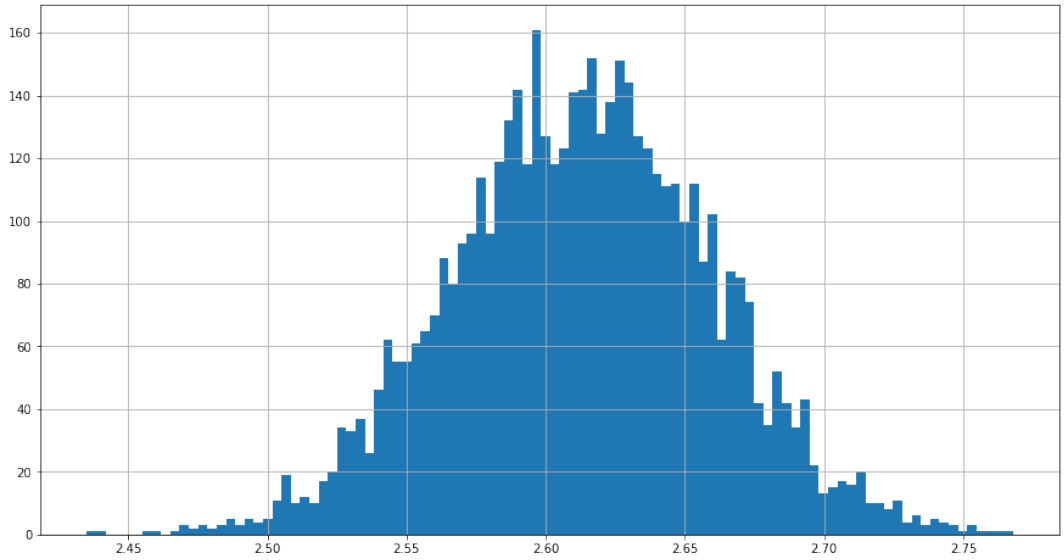

Figure 5: $n = 100$ wine plot

N = 120

For this scenario we have the following.

```
n2=120
s=5000
AvgWine2 =[]
STDWine2=[]
for i in range(s):
AvgWine2.append(SamplerAndAVG(Column12,n2))
STDWine2.append(SamplerAndSTD(Column12,n2))

```

```
np.mean(AvgWine2)
2.611849166666667
np.mean(STDWine2)
0.7070276177385078

```

We have the following mean and standard deviation.

$$\mu_{n=120} = 2.611849166666667, \quad \sigma_{n=120} = 0.7070276177385078$$

The code to plot and result is as follows.

```
plt.figure(figsize=(15,8))
ax = plt.subplot(111)
ax.set_ylabel('');
ax.set_xlabel('');
ax.hist(AvgWine2, bins=100);
ax.grid()

```

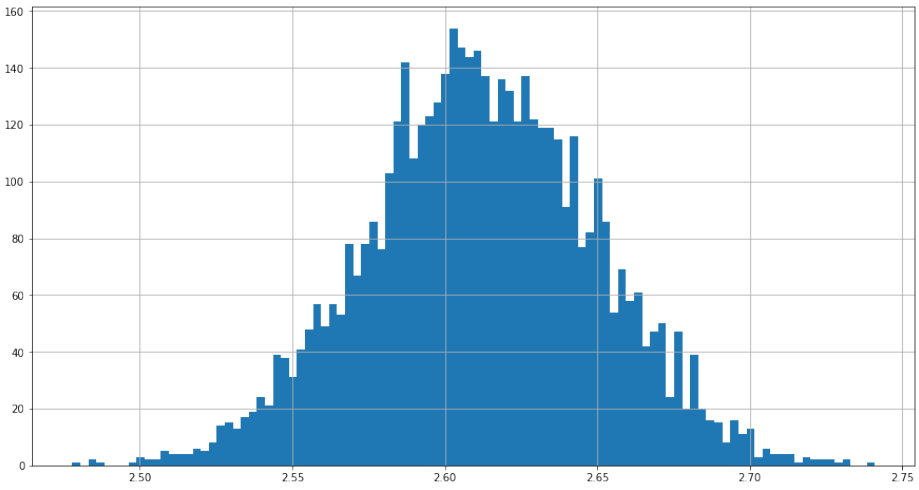

Figure 6: $n = 120$ wine plot

N = 150

For this scenario we have the following.

```
n3=150
s=5000
AvgWine3 =[]
STDWine3=[]
for i in range(s):
AvgWine3.append(SamplerAndAVG(Column12,n3))
STDWine3.append(SamplerAndSTD(Column12,n3))

```

```
np.mean(AvgWine3)
2.6122847866666667
np.mean(STDWine3)
0.7074754348148378

```

We have the following mean and standard deviation.

$$\mu_{n=150} = 2.6122847866666667, \quad \sigma_{n=150} = 0.7074754348148378$$

The code to plot and result is as follows.

```
plt.figure(figsize=(15,8))
ax = plt.subplot(111)
ax.set_ylabel('');
ax.set_xlabel('');
ax.hist(AvgWine3,bins=100);
ax.grid()

```

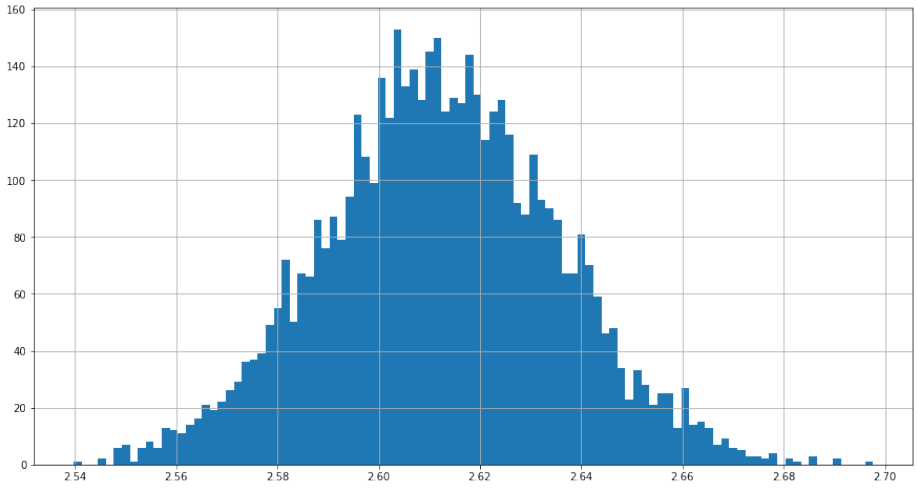

Figure 7: $n = 150$ wine plot

As we can see the central limit theorem holds and the mean and standard deviation is very close to the real values.

1.3 Central Limit Theorem

We know that $X_1, X_2, \ldots, X_n$ are random variables with $\mathbb{E}(X_i) = m_i$ and $Var(X_i) = \sigma_i^2 < \infty$, if we define $a = \lim_{n\to\infty} \frac{1}{\sqrt{n}} \sum_{i=1}^{n} m_i$ and $b = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \sigma_i^2$, it can be proven that $Z_n = \frac{\sum_{i=1}^{n} X_i}{\sqrt{n}}$ converges to Z in distribution where Z is defined as below.

$$Z_n \xrightarrow{d} Z, \quad Z \sim \mathcal{N}(a, b)$$

2 Question 2 : Statistical Analysis (Analytic)

2.1 Part 1

Here it is implied that Ahmad who is a machine learning student, proposes he has a solution for the problem that in real situations, features aren't independent, he claims that he has a statistical and probabilistic method to move data to another dimension in which the data is described by new features then he shall use Naive Bayes. He claims that due to the orthogonality and uncorrelatedness of the features in this new dimension, the Naive Bayes assumption holds, now we shall analyze this answer.

This hypothesis is visibly incorrect, the reason is simple, we know the following:

$$Orthogonality \rightarrow Uncorrelated$$
$$Uncorrelated \nrightarrow Indpendent$$

So, as we new we can't infer that features are independent, but if we have independent features it can be deduce that we have:

$$Indpendent \rightarrow Uncorrelated \rightarrow Orthogonality$$

So all in all, Ahmad's deduction and analysis is false.

2.2 Part 2

Here we shall perform the **Maximum Likelihood Estimation** to find the distribution parameters which in our case our the mean and variance.

$$\{x_1, x_2, ..., x_n\} \longrightarrow i.i.d \,\&\, Normal$$

We learned in the **Stochastic Processes** course that the probability density function of a vector of normal random variables is as follows.

$$f_x()$$

Now for a single gaussian random variable such as x_i we have.

$$f_x(x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_i-\mu)}{\sigma^2}}$$

As instructed in the course lectures we must form a likelihood function as follows.

$$\mathcal{P}(x_1, \ldots, x_n; \mu, \sigma^2) = \prod_{i=1}^{n} f_X(x_i; \mu, \sigma^2)$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_i-\mu)}{\sigma^2}} = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}$$

$$\mathcal{P}(x_1, \ldots, x_n; \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}$$

As we know, using this likelihood function is difficult, so we shall define the logarithmic loss using the natural logarithm as follows.

$$\mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2) = \ln\left(\mathcal{P}(x_1, \ldots, x_n; \mu, \sigma^2)\right)$$

$$= \ln\left(\frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}\right) = \ln\left((2\pi\sigma^2)^{-\frac{n}{2}}\right)$$

$$+ \ln\left(e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}\right) = -\frac{n}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

$$= -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

$$\longrightarrow \mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Now we shall go on to find the ML estimators.

2.2.1 Mean

$$\max_{\mu,\sigma^2} \mathcal{L}(\mu, \sigma^2; x_1, \ldots, x_n) \longrightarrow \frac{\partial}{\partial \mu}\mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2)$$
$$\longrightarrow \frac{\partial}{\partial \mu}\left(-\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2\right) = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)$$
$$\frac{1}{\sigma^2}\left(\sum_{i=1}^{n} x_i - n\mu\right), \quad \frac{\partial}{\partial \mu}\mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2) = 0 \longrightarrow \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

2.2.2 Variance

$$\max_{\mu,\sigma^2} \mathcal{L}(\mu, \sigma^2; x_1, \ldots, x_n) \longrightarrow \frac{\partial}{\partial \sigma^2}\mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2)$$
$$\longrightarrow \frac{\partial}{\partial \sigma^2}\left(-\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2\right)$$
$$= -\frac{n}{2\sigma^2} - \frac{\partial}{\partial \sigma^2}\left(\frac{1}{\sigma^2}\right)\left[\sum_{i=1}^{n}\frac{(x_i - \mu)^2}{2}\right] = -\frac{n}{2\sigma^2} + \left(\frac{1}{\sigma^4}\right)\left[\sum_{i=1}^{n}\frac{(x_i - \mu)^2}{2}\right]$$
$$= \frac{1}{2\sigma^2}\left[\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2 - n\right], \quad \frac{\partial}{\partial \sigma^2}\mathcal{L}(x_1, \ldots, x_n; \mu, \sigma^2) = 0$$
$$\longrightarrow \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

So the parameters are as below.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

3 Question 3 : Naive Bayes

In this section we shall implement a Naive Bayes design to calculate the accuracy of detection of the validation images.

3.1 Part 1

As instructed in the helper code, I have used the following attributes as features.

- Width
- Height
- Number of Hashtags

To calculate the prior probabilities we simply calculated the probability for each different class as follows on the train data.

$$P_{c_i} = \frac{N_{c_i}}{N_{tot}}$$

The used functions are as follows.

```
1 def TrainTestData(ALL_TRAINING_IMAGES,ALL_VALIDATION_IMAGES):
2 return np.array(ALL_TRAINING_IMAGES),np.array(
   ALL_VALIDATION_IMAGES)
3
4 def TrainTestLabels(ALL_TRAINING_LABELS,ALL_VALIDATION_LABELS)
   :
5 return np.array(ALL_TRAINING_LABELS),np.array(
   ALL_VALIDATION_LABELS)
6
```

An example digit is depicted as follows.

```
print("Printing digit example #" + str(250) + " with label: "
  \
+ str(ALL_TRAINING_LABELS[250]))
_print_digit_image(ALL_TRAINING_IMAGES[250])

```

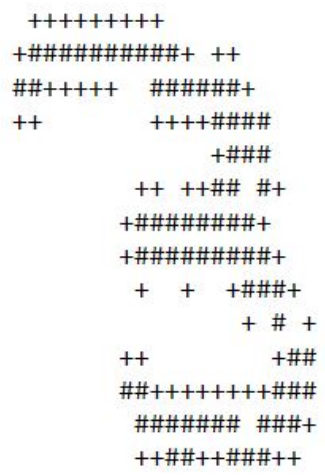

Figure 8: Sample Digit

```
xtrain,xval = TrainTestData(ALL_TRAINING_IMAGES,
  ALL_VALIDATION_IMAGES)
ytrain,yval = TrainTestLabels(ALL_TRAINING_LABELS,
  ALL_VALIDATION_LABELS)

```

```
def NumberShower(Image):
for i in range(DATA_HEIGHT):
print(Image[i])

```

The output of the following function for the same digit is as follows.

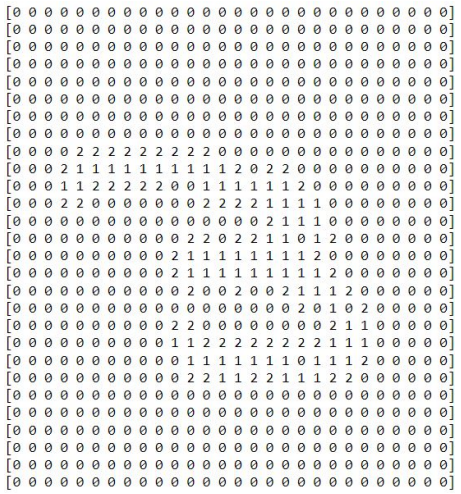

Figure 9: Sample Digit

As we can see the outline of 3 is vaguely distinguishable by the digits 0 , 1 and 2, we use this and the sums of the rows and columns to write functions that can calculate the height and width.

```
def HeightCalc(Image):
height = 0
for i in range(DATA_HEIGHT):
if(sum(Image[i])!=0):
height+=1
return height

```

```
def WidthCalc(Image):
width = 0
for i in range(DATA_WIDTH):
if(sum(Image[:,i])!=0):
width+=1
return width

```

Finally the last feature extractor function is as follows.

```
def HashtagCounter(Image):
HashtagNum = 0
for i in range(DATA_WIDTH):
HashtagNum+=np.count_nonzero((Image[:,i]==1))
return HashtagNum

```

After this we must categorize the data by the indices and then put them in a list with the corresponding labels.

```
def CategorizerByIndice(TrainLabels,ValLabels):
TrainCategory = []
ValCategory = []
for i in range(0,10):
valtemp=[]
for k in range(1000):
if ValLabels[k]==i:
valtemp.append(k)
ValCategory.append(valtemp)
for l in range(0,10):
traintemp=[]
for j in range(5000):
if TrainLabels[j]==l:
traintemp.append(j)
TrainCategory.append(traintemp)
return TrainCategory,ValCategory

```

```
def CategorizerByClass(Label,Category):
for m in range(10):
if Label in Category[m]:
return m
else:
continue

```

Next we go on to calculate the prior probabilities.

```
def PriorProbCalc(TrainCategory):
Prior = []
for i in range(0,10):
prob = len(TrainCategory[i])/NUMBER_OF_TRAINING_EXAMPLES
Prior.append(prob)
return Prior

```

```
PriorProbs = PriorProbCalc(TrainCategory)
PriorProbs
[0.0958, 0.1126, 0.0976, 0.0986, 0.107, 0.0868, 0.1002, 0.11, 0.0924, 0.099]

```

After this first of all I've written a function to save the features in them for future usage, then because we want to use the Gaussian Naive Bayes, we calculate the mean and variance.

```
def FeatureSaver(xtrain):
AllFeatures  = []
temp = []
for i in range(3):
AllFeatures.append(list(temp))
for j in range(10):
AllFeatures[0].append(list(temp))
AllFeatures[1].append(list(temp))
AllFeatures[2].append(list(temp))
for k in range(5000):
Category = CategorizerByClass(k,TrainCategory)
AllFeatures[0][Category].append(HeightCalc(xtrain[k]))
AllFeatures[1][Category].append(WidthCalc(xtrain[k]))
AllFeatures[2][Category].append(HashtagCounter(xtrain[k]))
return AllFeatures

```

Now we take a look at the Gaussian Naive Bayes formula.

$$P(X|Y=c) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
def Gaussian(xtrain, AllFeatures):
AllFeaturesGaussian  = []
temp = []
for i in range(3):
AllFeaturesGaussian.append(list(temp))
for j in range(10):
AllFeaturesGaussian[0].append(list(temp))
AllFeaturesGaussian[1].append(list(temp))
AllFeaturesGaussian[2].append(list(temp))
for k in range(10):
AllFeaturesGaussian[0][k].append(np.mean(AllFeatures[0][k]))
AllFeaturesGaussian[0][k].append(np.var(AllFeatures[0][k]))
AllFeaturesGaussian[1][k].append(np.mean(AllFeatures[1][k]))
AllFeaturesGaussian[1][k].append(np.var(AllFeatures[1][k]))
AllFeaturesGaussian[2][k].append(np.mean(AllFeatures[2][k]))
AllFeaturesGaussian[2][k].append(np.var(AllFeatures[2][k]))
return AllFeaturesGaussian

```

3.2 Part 2

In this part, we shall move in to finish, and report the accuracy of our design.

```
def NaiveBayesFinalAndAccuracy(AllFeatures,xval,yval,xtrain):
AllFeaturesGaussian=Gaussian(xtrain,AllFeatures)
Pred = []
true = 0
for k in range(1000):
ProbArr=[]
for i in range(0,10):
Prob1 = (1/(np.sqrt(AllFeaturesGaussian[0][i][1]*np.pi*2)))*np
  .exp((-.5*((HeightCalc(xval[k])-AllFeaturesGaussian[0][i][0])
  **2)/(AllFeaturesGaussian[0][i][1])))
Prob2 = (1/(np.sqrt(AllFeaturesGaussian[1][i][1]*np.pi*2)))*np
  .exp((-.5*((WidthCalc(xval[k])-AllFeaturesGaussian[1][i][0])
  **2)/(AllFeaturesGaussian[1][i][1])))
Prob3 = (1/(np.sqrt(AllFeaturesGaussian[2][i][1]*np.pi*2)))*np
  .exp((-.5*((HashtagCounter(xval[k])-AllFeaturesGaussian[2][i
  ][0])**2)/(AllFeaturesGaussian[2][i][1])))
ProbArr.append(Prob1*Prob2*Prob3*PriorProbs[i])
Pred.append(np.argmax(ProbArr))
for i in range(len(Pred)):
if yval[i]==Pred[i]:
true+=1
Acc= true/NUMBER_OF_VALIDATION_EXAMPLES * 100
print(f'Accuracy of Naive Bayes is {Acc}%')

```

```
NaiveBayesFinalAndAccuracy(AllFeatures,xval,yval,xtrain)
Accuracy of Naive Bayes is 22.7%

```

This concludes this assignment.

References

[1] Reshad Hosseini, *Intelligent Systems Lecture Notes, Fall 01*

[2] Ali Olfat, *Stochastic Processes Lecture Notes, Fall 01*

[3] AmirMasoud Rabiei,*Engineering Probability and Statistics Computer Assignments, Fall 99*

[4] Gaussian Naive Bayes

[5] Histogram in Numpy

[6] Naive Bayes Explained

[7] Check if element exists in list in Python

[8] Secretary Problem Explained: Dating Mathematically