# Part 1 Accuracies (1A, 1B)

|  |  |
| --- | --- |
| Setup | Cross-Validation Accuracy |
| Unprocessed data | 73.40 |
| 0 – Value elements ignored | 73.66 |

# Part 1 Code Snippets

|  |  |
| --- | --- |
| Attribute | Code Snippets |
| **Calculation of Distribution Parameter** | df\_pima\_train\_set = input\_df.iloc[input\_train\_splitloc][input\_df.iloc[input\_train\_splitloc]['Class'] == c]  **if** (impute\_ind):  df\_pima\_train\_set['BloodPressure']=df\_pima\_train\_set['BloodPressure'].replace(0,np.NAN) *#impute to NAN, so it won't used in mean/std*  df\_pima\_train\_set['SkinThickness']=df\_pima\_train\_set['SkinThickness'].replace(0,np.NAN)  df\_pima\_train\_set['BMI']=df\_pima\_train\_set['BMI'].replace(0,np.NAN)  df\_pima\_train\_set['Age']=df\_pima\_train\_set['Age'].replace(0,np.NAN)  mean=df\_pima\_train\_set.describe().loc['mean'][:-1]  stdev=df\_pima\_train\_set.describe().loc['std'][:-1] |
| **Calculation of Naïve Bayes Prediction** | **for** c **in** input\_distinct\_class:  exp\_nr = -((df\_pima.iloc[input\_test\_splitloc].drop('Class',axis=1)-np.array(input\_dict\_train\_mean\_stdev[c][0]))\*\*2)  exp\_dn = (2\*((dict\_train\_mean\_stdev[c][1]) \*\* 2 ))  exp = exp\_nr / exp\_dn  exp = np.exp(exp)  coef = (1/((np.sqrt(2\*np.pi))\*input\_dict\_train\_mean\_stdev[c][1]))  ndf = np.sum(np.log(coef \* exp),axis=1)  fold\_predict\_class[:,c] = ndf  pred\_test = pd.Series(pd.DataFrame(fold\_predict\_class).idxmax(axis=1).values,index=input\_test\_splitloc) |
| **Train-Test Split Code** | train\_splitloc = []  test\_splitloc = []  train\_end\_loc = np.round(input\_df.shape[0]\*(train\_split/100)).astype(int)  **for** f **in** range(fold):  loc\_arr = np.arange(input\_df.shape[0])  np.random.shuffle(loc\_arr)  train\_splitloc.append(loc\_arr[:train\_end\_loc])  test\_splitloc.append(loc\_arr[train\_end\_loc:])  **return** train\_splitloc,test\_splitloc |

Please refer to the [Link](#_PIMA_Diabetes_Classification) for the full details of the code

# Part 2 MNIST Accuracies (2A, 2B)

|  |  |  |  |
| --- | --- | --- | --- |
| X | Method | Training Set Accuracy | Test Set Accuracy |
| 1 | Gaussian + untouched | 53.10 | 51.89 |
| 2 | Gaussian + Stretched | 81.12 | 82.38 |
| 3 | Bernoulli + untouched | 83.71 | 84.38 |
| 4 | Bernoulli + Stretched | 81.81 | 83.15 |
| 5 | 10 trees + 4 depth + untouched | 74.98 | 75.61 |
| 6 | 10 trees + 4 depth + stretched | 75.58 | 76.72 |
| 7 | 10 trees + 16 depth + untouched | 99.53 | 94.03 |
| 8 | 10 trees + 16 depth + stretched | 99.77 | 95.21 |
| 9 | 30 trees + 4 depth + untouched | 80.80 | 81.37 |
| 10 | 30 trees + 4 depth + stretched | 78.91 | 79.83 |
| 11 | 30 trees + 16 depth + untouched | 99.76 | 96.22 |
| 12 | 30 trees + 16 depth + stretched | 99.87 | 96.71 |

# Part 2A Digit Images

|  |  |  |  |
| --- | --- | --- | --- |
| Digit | Mean Image | Digit | Mean Image |
| 0 |  | 6 |  |
| 1 |  | 7 |  |
| 2 |  | 8 |  |
| 3 |  | 9 |  |
| 4 |  |

# Part 2 Code

|  |  |
| --- | --- |
| Attribute | Code Snippets |
| **Calculation of Normal Distribution Parameters** | eps = 1e-4 *#Added a small value in order to avoid the variance to 0 (divisible by zero)*  **for** c **in** distinct\_class:  *#print ("Running for the Class: {}".format(c))*  mean=input\_df.iloc[input\_train\_splitloc][df\_mnist.iloc[input\_train\_splitloc]['target'] == c].describe().loc['mean'][:-1]  stdev=input\_df.iloc[input\_train\_splitloc][df\_mnist.iloc[input\_train\_splitloc]['target'] == c].describe().loc['std'][:-1]+eps  dict\_train\_mean\_stdev[c] = mean,stdev |
| **Calculation of Bernoulli Distribution Parameters** | priors=df\_input.iloc[train\_splitloc[0]].groupby('target').count()[0]  df\_input\_train['target'] = df\_train\_target  df\_train\_summary=df\_input\_train.groupby('target').sum()  **for** p **in** range(len(priors)):  df\_train\_summary.iloc[p] = (df\_train\_summary.iloc[p]+0.01)/(priors[p]+0.02)  df\_input\_train.drop('target',axis=1,inplace=**True**) |
| **Calculation of the Naïve Bayes Predictions** | naive\_bayes\_pred(input\_df,input\_test\_splitloc,input\_dict\_train\_mean\_stdev,input\_distinct\_class):  fold\_predict\_class = np.zeros((len(input\_test\_splitloc),len(input\_distinct\_class)))  **for** c **in** input\_distinct\_class:  exp\_nr = -((input\_df.iloc[input\_test\_splitloc].drop('target',axis=1)-np.array(input\_dict\_train\_mean\_stdev[c][0]))\*\*2)  exp\_dn = (2\*((dict\_train\_mean\_stdev[c][1]) \*\* 2 ))  exp = exp\_nr / exp\_dn  exp = np.exp(exp)  coef = (1/((np.sqrt(2\*np.pi))\*input\_dict\_train\_mean\_stdev[c][1]))  ndf = np.sum(np.log(coef \* exp),axis=1)  fold\_predict\_class[:,c] = ndf  naive\_bayes\_bernoulli(df\_input,input\_test\_splitloc,df\_input\_mnist\_test,df\_train\_summary):  pred\_test\_val=np.argmax(np.dot((np.log(1-df\_train\_summary)),(1-df\_input\_mnist\_test).T)+np.dot((np.log(df\_train\_summary)),(df\_input\_mnist\_test).T),axis=0) |
| **Training a decision Tress** | clf=RandomForestClassifier(n\_estimators=t,max\_depth=d)  clf.fit(X=df\_mnist.drop('target',axis=1).iloc[train\_splitloc[0]],y=df\_mnist['target'].iloc[train\_splitloc[0]]) |
| **Calculation of a decision tree predictions** | clf.predict(df\_mnist.drop('target',axis=1).iloc[train\_splitloc[0]]) |

Please refer to the [link](#_MNIST_Image_Classification) for the full details of the code

# Entire Code

## PIMA Diabetes Classification

|  |
| --- |
| **import** **pandas** **as** **pd**  **import** **numpy** **as** **np**  **import** **warnings**  warnings.filterwarnings('ignore')  **def** load\_dataset(filepath="data/pima-indians-diabetes.csv"):  df\_pima = pd.read\_csv(filepath)  df\_pima.columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Class']  distinct\_class=df\_pima['Class'].unique()  **return** df\_pima,distinct\_class  **def** train\_test\_split(input\_df=**None**,fold=10,print\_ind=**False**,train\_split=80):  train\_splitloc = []  test\_splitloc = []  train\_end\_loc = np.round(input\_df.shape[0]\*(train\_split/100)).astype(int)  **for** f **in** range(fold):  loc\_arr = np.arange(input\_df.shape[0])  np.random.shuffle(loc\_arr)  train\_splitloc.append(loc\_arr[:train\_end\_loc])  test\_splitloc.append(loc\_arr[train\_end\_loc:])  **return** train\_splitloc,test\_splitloc  **def** train\_class\_mean\_std(input\_df,input\_train\_splitloc,impute\_ind=**False**):  dict\_train\_mean\_stdev\_calc = {}  dict\_train\_mean\_stdev\_impute\_calc = {}  **for** c **in** distinct\_class:  df\_pima\_train\_set = input\_df.iloc[input\_train\_splitloc][input\_df.iloc[input\_train\_splitloc]['Class'] == c]  **if** (impute\_ind):  df\_pima\_train\_set['BloodPressure']=df\_pima\_train\_set['BloodPressure'].replace(0,np.NAN) *#impute to NAN, so it won't used in mean/std*  df\_pima\_train\_set['SkinThickness']=df\_pima\_train\_set['SkinThickness'].replace(0,np.NAN) *#impute to NAN, so it won't used in mean/std*  df\_pima\_train\_set['BMI']=df\_pima\_train\_set['BMI'].replace(0,np.NAN) *#impute to NAN, so it won't used in mean/std*  df\_pima\_train\_set['Age']=df\_pima\_train\_set['Age'].replace(0,np.NAN) *#impute to NAN, so it won't used in mean/std*  mean=df\_pima\_train\_set.describe().loc['mean'][:-1]  stdev=df\_pima\_train\_set.describe().loc['std'][:-1]  dict\_train\_mean\_stdev\_calc[c] = mean,stdev  **return** dict\_train\_mean\_stdev\_calc  **def** gaussian\_naive\_bayes\_pred(input\_test\_splitloc,input\_dict\_train\_mean\_stdev,input\_distinct\_class):  fold\_predict\_class = np.zeros((len(input\_test\_splitloc),len(input\_distinct\_class)))  **for** c **in** input\_distinct\_class:  exp\_nr = -((df\_pima.iloc[input\_test\_splitloc].drop('Class',axis=1)-np.array(input\_dict\_train\_mean\_stdev[c][0]))\*\*2)  exp\_dn = (2\*((dict\_train\_mean\_stdev[c][1]) \*\* 2 ))  exp = exp\_nr / exp\_dn  exp = np.exp(exp)  coef = (1/((np.sqrt(2\*np.pi))\*input\_dict\_train\_mean\_stdev[c][1]))  ndf = np.sum(np.log(coef \* exp),axis=1)  fold\_predict\_class[:,c] = ndf  pred\_test = pd.Series(pd.DataFrame(fold\_predict\_class).idxmax(axis=1).values,index=input\_test\_splitloc)  **return** pred\_test  fold = 10  overall\_match\_class = 0  overall\_match\_class\_ignore\_missing=0  df\_pima,distinct\_class=load\_dataset() *#Load the Dataset*  train\_splitloc,test\_splitloc=train\_test\_split(df\_pima) *#Split the Dataset*  **for** f **in** range(fold): *#For each Fold*  match\_class = 0  dict\_train\_mean\_stdev=train\_class\_mean\_std(df\_pima,train\_splitloc[f],impute\_ind=**False**)  dict\_train\_mean\_stdev\_ignore\_missing=train\_class\_mean\_std(df\_pima,train\_splitloc[f],impute\_ind=**True**)  pred\_test\_val=gaussian\_naive\_bayes\_pred(test\_splitloc[f],dict\_train\_mean\_stdev,distinct\_class)  pred\_test\_val\_ignore\_missing=gaussian\_naive\_bayes\_pred(test\_splitloc[f],dict\_train\_mean\_stdev\_ignore\_missing,distinct\_class)  match\_class = (np.sum(np.array(pred\_test\_val) == df\_pima.iloc[test\_splitloc[f]]['Class'].values)/len(test\_splitloc[f]))\*100  match\_class\_ignore\_missing=(np.sum(np.array(pred\_test\_val\_ignore\_missing) == df\_pima.iloc[test\_splitloc[f]]['Class'].values)/len(test\_splitloc[f]))\*100  overall\_match\_class += match\_class  overall\_match\_class\_ignore\_missing += match\_class\_ignore\_missing  print ("folder: **{}** Gaussian NB Accuracy: **{}** Ignore Missing Accuracy:**{}**".format(f,match\_class,match\_class\_ignore\_missing))  print ("Gaussian NB Average Accuracy: **{}** Ignore Missing Accuracy:**{}**".format(overall\_match\_class/fold,overall\_match\_class\_ignore\_missing/fold)) |

## MNIST Image Classification

|  |
| --- |
| **import** **pandas** **as** **pd**  **import** **numpy** **as** **np**  **import** **cv2**  **from** **sklearn.ensemble** **import** RandomForestClassifier  **import** **matplotlib.pyplot** **as** **plt**  %**matplotlib** inline  **import** **warnings**  warnings.filterwarnings('ignore')  **def** load\_mnist\_data():  **from** **sklearn.datasets** **import** fetch\_mldata  mnist = fetch\_mldata('MNIST original')  X,Y = mnist["data"],mnist["target"].astype(int)  df\_mnist=pd.DataFrame(X)  df\_mnist['target'] = Y  distinct\_class=pd.Series(Y).unique().astype(int)  **return** df\_mnist,distinct\_class  **def** train\_test\_split(df\_input,train\_set=60000,print\_ind=**False**):  data\_loc=np.arange(df\_input.shape[0])  train\_splitloc=[data\_loc[:60000]]  test\_splitloc=[data\_loc[60000:]]  **return** train\_splitloc,test\_splitloc  **def** priors\_calc(df\_input):  priors=df\_input.iloc[train\_splitloc[0]].groupby('target').count()[0]  **return** priors  **def** mnist\_transformed(df\_input):  df\_mnist\_train = (df\_input.iloc[train\_splitloc[0]].drop('target',axis=1) >= 128).astype(int)  df\_mnist\_test = (df\_input.iloc[test\_splitloc[0]].drop('target',axis=1) >= 128).astype(int)  **return** df\_mnist\_train,df\_mnist\_test  **def** mnist\_train\_summary(df\_input\_train,df\_train\_target):  df\_input\_train['target'] = df\_train\_target  df\_train\_summary=df\_input\_train.groupby('target').sum()  **for** p **in** range(len(priors)):  df\_train\_summary.iloc[p] = (df\_train\_summary.iloc[p]+0.01)/(priors[p]+0.02)  df\_input\_train.drop('target',axis=1,inplace=**True**)  **return** df\_train\_summary  **def** train\_class\_mean\_std(input\_df,input\_train\_splitloc,print\_ind=**False**):  dict\_train\_mean\_stdev = {}  eps = 1e-4 *#Added a small value in order to avoid the variance to 0 (divisible by zero)*  **for** c **in** distinct\_class:  mean=input\_df.iloc[input\_train\_splitloc][df\_mnist.iloc[input\_train\_splitloc]['target'] == c].describe().loc['mean'][:-1]  stdev=input\_df.iloc[input\_train\_splitloc][df\_mnist.iloc[input\_train\_splitloc]['target'] == c].describe().loc['std'][:-1]+eps  dict\_train\_mean\_stdev[c] = mean,stdev  **if** (print\_ind):  print ("Len Train:**{}**. Number of 0:**{}** 1:**{}**".format(len(input\_train\_splitloc),df\_pima.iloc[input\_train\_splitloc][df\_pima.iloc[input\_train\_splitloc]['Class'] == 0].shape,df\_pima.iloc[input\_train\_splitloc][df\_pima.iloc[input\_train\_splitloc]['Class'] == 1].shape))  **return** dict\_train\_mean\_stdev  **def** mnist\_cropped\_func(input\_df,width=20,height=20):  i=0  sr\_mnist\_cropped = []  df\_mnist\_cropped = pd.DataFrame()  **for** k **in** np.array(input\_df.drop('target',axis=1)):  x=k.reshape(28,28)  coord=np.argwhere(x)  x0,y0=np.min(coord,axis=0)  x1,y1=np.max(coord,axis=0)  X\_cropped=x[x0:x1,y0:y1]    dim = (width, height)    X\_stretched=cv2.resize(X\_cropped, dim, interpolation = cv2.INTER\_NEAREST)  X\_stretched=X\_stretched.reshape(width\*height,)  sr\_mnist\_cropped.append(X\_stretched)  *#df\_sr\_mnist\_train\_cropped=sr\_mnist\_train\_cropped*  df\_output\_mnist\_cropped=pd.DataFrame(sr\_mnist\_cropped)  df\_output\_mnist\_cropped['target'] = df\_mnist['target']  **return** df\_output\_mnist\_cropped  **def** naive\_bayes\_pred(input\_df,input\_test\_splitloc,input\_dict\_train\_mean\_stdev,input\_distinct\_class):  fold\_predict\_class = np.zeros((len(input\_test\_splitloc),len(input\_distinct\_class)))  **for** c **in** input\_distinct\_class:  exp\_nr = -((input\_df.iloc[input\_test\_splitloc].drop('target',axis=1)-np.array(input\_dict\_train\_mean\_stdev[c][0]))\*\*2)  exp\_dn = (2\*((dict\_train\_mean\_stdev[c][1]) \*\* 2 ))  exp = exp\_nr / exp\_dn  exp = np.exp(exp)  coef = (1/((np.sqrt(2\*np.pi))\*input\_dict\_train\_mean\_stdev[c][1]))  ndf = np.sum(np.log(coef \* exp),axis=1)  fold\_predict\_class[:,c] = ndf  pred\_test = pd.Series(pd.DataFrame(fold\_predict\_class).idxmax(axis=1).values,index=input\_test\_splitloc)  **return** pred\_test  **def** naive\_bayes\_bernoulli(df\_input,input\_test\_splitloc,df\_input\_mnist\_test,df\_train\_summary):  pred\_test\_val=np.argmax(np.dot((np.log(1-df\_train\_summary)),(1-df\_input\_mnist\_test).T)+np.dot((np.log(df\_train\_summary)),(df\_input\_mnist\_test).T),axis=0)  *#np.array(df\_input.iloc[test\_splitloc[0]]['target'])*  **return** ((np.sum((np.array(df\_input.iloc[input\_test\_splitloc[0]]['target'])) == pred\_test\_val))/len(input\_test\_splitloc[0]))\*100  df\_mnist,distinct\_class=load\_mnist\_data() *#Load the Dataset*  train\_splitloc,test\_splitloc=train\_test\_split(df\_mnist)  df\_mnist\_cropped=mnist\_cropped\_func(df\_mnist,20,20)  fold=1  **for** f **in** range(fold): *#For each Fold*  match\_train\_class = 0  match\_test\_class = 0  dict\_train\_mean\_stdev=train\_class\_mean\_std(df\_mnist,train\_splitloc[f])  pred\_train\_val=naive\_bayes\_pred(df\_mnist,train\_splitloc[f],dict\_train\_mean\_stdev,distinct\_class) *# Train Accuracy*  pred\_test\_val=naive\_bayes\_pred(df\_mnist,test\_splitloc[f],dict\_train\_mean\_stdev,distinct\_class) *# Test Accuracy*    match\_class\_train = np.sum(pred\_train\_val == df\_mnist.iloc[train\_splitloc[f]]['target'].values) / len(train\_splitloc[0])  match\_class\_test = np.sum(pred\_test\_val == df\_mnist.iloc[test\_splitloc[f]]['target'].values) / len(test\_splitloc[0])  print ("Gaussian Untouched Train Accuracy: **{}** Test Accuracy:**{}**".format(match\_class\_train\*100,match\_class\_test\*100))  **for** f **in** range(fold): *#For each Fold*  match\_train\_class = 0  match\_test\_class = 0  dict\_train\_mean\_stdev=train\_class\_mean\_std(df\_mnist\_cropped,train\_splitloc[f])  pred\_train\_val=naive\_bayes\_pred(df\_mnist\_cropped,train\_splitloc[f],dict\_train\_mean\_stdev,distinct\_class) *# Train Accuracy*  pred\_test\_val=naive\_bayes\_pred(df\_mnist\_cropped,test\_splitloc[f],dict\_train\_mean\_stdev,distinct\_class) *# Test Accuracy*  match\_class\_train = np.sum(pred\_train\_val == df\_mnist\_cropped.iloc[train\_splitloc[f]]['target'].values) / len(train\_splitloc[0])  match\_class\_test = np.sum(pred\_test\_val == df\_mnist\_cropped.iloc[test\_splitloc[f]]['target'].values) / len(test\_splitloc[0])  print ("Gaussian Stretched Train Accuracy: **{}** Test Accuracy:**{}**".format(match\_class\_train\*100,match\_class\_test\*100))  priors = priors\_calc(df\_mnist)  df\_mnist\_train,df\_mnist\_test=mnist\_transformed(df\_mnist)  df\_mnist\_train\_summary=mnist\_train\_summary(df\_mnist\_train,df\_mnist['target'])  print ("NB Bernouilli - Untouched Train Accuracy:**{}**".format(naive\_bayes\_bernoulli(df\_mnist,train\_splitloc,df\_mnist\_train,df\_mnist\_train\_summary)))  print ("NB Bernouilli - Untouched Test Accuracy:**{}**".format(naive\_bayes\_bernoulli(df\_mnist,test\_splitloc,df\_mnist\_test,df\_mnist\_train\_summary)))  priors = priors\_calc(df\_mnist)  df\_mnist\_train,df\_mnist\_test=mnist\_transformed(df\_mnist\_cropped)  df\_mnist\_train\_summary=mnist\_train\_summary(df\_mnist\_train,df\_mnist\_cropped['target'])  print ("NB Bernouilli - Stretched Train Accuracy:**{}**".format(naive\_bayes\_bernoulli(df\_mnist\_cropped,train\_splitloc,df\_mnist\_train,df\_mnist\_train\_summary)))  print ("NB Bernouilli - Stretched Test Accuracy:**{}**".format(naive\_bayes\_bernoulli(df\_mnist\_cropped,test\_splitloc,df\_mnist\_test,df\_mnist\_train\_summary)))  *#Predict for Random Forrest*  **for** t **in** [10,30]:  **for** d **in** [4,16]:  *#print ("Processing for Tree:{} Depth:{}".format(t,d))*  clf=RandomForestClassifier(n\_estimators=t,max\_depth=d)  clf.fit(X=df\_mnist.drop('target',axis=1).iloc[train\_splitloc[0]],y=df\_mnist['target'].iloc[train\_splitloc[0]])  rand\_forrest\_train\_pred=clf.predict(df\_mnist.drop('target',axis=1).iloc[train\_splitloc[0]])  rand\_forrest\_test\_pred=clf.predict(df\_mnist.drop('target',axis=1).iloc[test\_splitloc[0]])  rand\_forrest\_train\_percent=(np.sum(np.array(df\_mnist.iloc[train\_splitloc[0]]['target']) == rand\_forrest\_train\_pred).astype(int)/df\_mnist.iloc[train\_splitloc[0]].shape[0])\*100  rand\_forrest\_test\_percent=(np.sum(np.array(df\_mnist.iloc[test\_splitloc[0]]['target']) == rand\_forrest\_test\_pred).astype(int)/df\_mnist.iloc[test\_splitloc[0]].shape[0])\*100  print ("Random Forrest - Untouched Tree:**{}** Depth:**{}** Train Accuracy:**{}** Test Accuracy:**{}**".format(t,d,rand\_forrest\_train\_percent,rand\_forrest\_test\_percent))    clf\_cropped=RandomForestClassifier(n\_estimators=t,max\_depth=d)  clf\_cropped.fit(X=df\_mnist\_cropped.drop('target',axis=1).iloc[train\_splitloc[0]],y=df\_mnist\_cropped['target'].iloc[train\_splitloc[0]])  cropped\_rand\_forrest\_train\_pred=clf\_cropped.predict(df\_mnist\_cropped.drop('target',axis=1).iloc[train\_splitloc[0]])  cropped\_rand\_forrest\_test\_pred=clf\_cropped.predict(df\_mnist\_cropped.drop('target',axis=1).iloc[test\_splitloc[0]])  cropped\_rand\_forrest\_train\_percent=(np.sum(np.array(df\_mnist\_cropped.iloc[train\_splitloc[0]]['target']) == cropped\_rand\_forrest\_train\_pred).astype(int)/df\_mnist.iloc[train\_splitloc[0]].shape[0])\*100  cropped\_rand\_forrest\_test\_percent=(np.sum(np.array(df\_mnist\_cropped.iloc[test\_splitloc[0]]['target']) == cropped\_rand\_forrest\_test\_pred).astype(int)/df\_mnist.iloc[test\_splitloc[0]].shape[0])\*100  print ("Random Forrest - Stretched Tree:**{}** Depth:**{}** Train Accuracy:**{}** Test Accuracy:**{}**".format(t,d,cropped\_rand\_forrest\_train\_percent,cropped\_rand\_forrest\_test\_percent)) |