1 Part 1 Accuracies (1A, 1B)

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

2 Part 1 Code Snippets

<table>
<tr><th>Attribute</th><th>Code Snippets</th></tr>
<tr><td>Calculation of Distribution Parameter</td><td>

```
        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]
```

</td></tr>
<tr><td>Calculation of Naïve Bayes Prediction</td><td>

```
    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)
```

</td></tr>
<tr><td>Train-Test Split Code</td><td>

```
    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
```

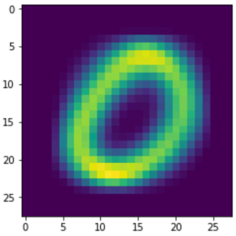
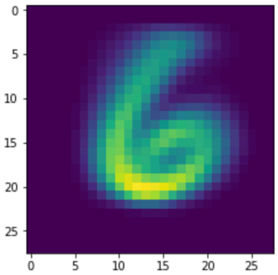
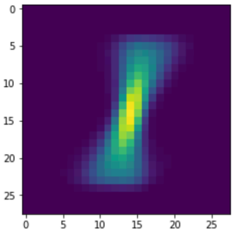
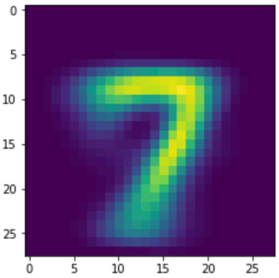
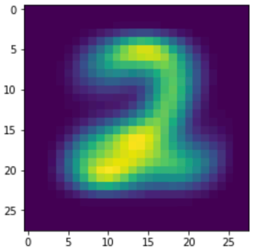
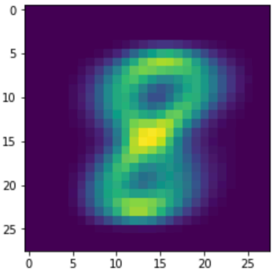
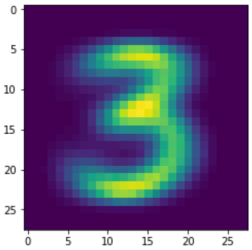
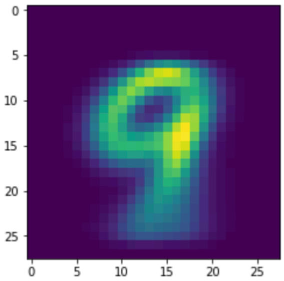
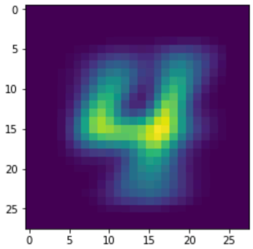
</td></tr>
</table>

Please refer to the Link for the full details of the code

3 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

4 Part 2A Digit Images

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

5 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_t`
	`est).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 for the full details of the code

6 Entire Code

6.1 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
```

```python
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(in
put_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_split
loc[f],impute_ind=False)
    dict_train_mean_stdev_ignore_missing=train_class_mean_std(df_p
ima,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_sp
litloc[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_igno
re_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_missi
ng
    print ("folder: {} Gaussian NB Accuracy: {}  Ignore Missing Ac
curacy:{}".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_mis
sing/fold))
```

6.2 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').cou
nt()[0]
    return priors

def mnist_transformed(df_input):
    df_mnist_train = (df_input.iloc[train_splitloc[0]].drop('targ
et',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 varianc
e to 0 (divisible by zero)
    for c in distinct_class:
        mean=input_df.iloc[input_train_splitloc][df_mnist.iloc[in
put_train_splitloc]['target'] == c].describe().loc['mean'][:-1]
        stdev=input_df.iloc[input_train_splitloc][df_mnist.iloc[i
nput_train_splitloc]['target'] == c].describe().loc['std'][:-1]+e
ps
        dict_train_mean_stdev[c] = mean,stdev
    if (print_ind):
        print ("Len Train:{}. Number of 0:{} 1:{}".format(len(inp
ut_train_splitloc),df_pima.iloc[input_train_splitloc][df_pima.ilo
c[input_train_splitloc]['Class'] == 0].shape,df_pima.iloc[input_t
rain_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 = cv
2.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_trai
n_mean_stdev,input_distinct_class):
    fold_predict_class = np.zeros((len(input_test_splitloc),len(i
nput_distinct_class)))
    for c in input_distinct_class:
        exp_nr = -((input_df.iloc[input_test_splitloc].drop('targ
et',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_m
nist_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_i
nput_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]))*10
0



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_spl
itloc[f])
    pred_train_val=naive_bayes_pred(df_mnist,train_splitloc[f],di
ct_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[tr
ain_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,t
rain_splitloc[f])
    pred_train_val=naive_bayes_pred(df_mnist_cropped,train_splitl
oc[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.i
loc[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_mnis
t['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_tra
in_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_mnis
t_cropped['target'])
print ("NB Bernouilli - Stretched Train Accuracy:{}".format(naive
_bayes_bernoulli(df_mnist_cropped,train_splitloc,df_mnist_train,d
f_mnist_train_summary)))
print ("NB Bernouilli - Stretched Test Accuracy:{}".format(naive_
bayes_bernoulli(df_mnist_cropped,test_splitloc,df_mnist_test,df_m
nist_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_split
loc[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(in
t)/df_mnist.iloc[test_splitloc[0]].shape[0])*100
        print ("Random Forrest - Untouched Tree:{} Depth:{}  Trai
n Accuracy:{} Test Accuracy:{}".format(t,d,rand_forrest_train_per
cent,rand_forrest_test_percent))

        clf_cropped=RandomForestClassifier(n_estimators=t,max_dep
th=d)
        clf_cropped.fit(X=df_mnist_cropped.drop('target',axis=1).
iloc[train_splitloc[0]],y=df_mnist_cropped['target'].iloc[train_s
plitloc[0]])
        cropped_rand_forrest_train_pred=clf_cropped.predict(df_mn
ist_cropped.drop('target',axis=1).iloc[train_splitloc[0]])
        cropped_rand_forrest_test_pred=clf_cropped.predict(df_mni
st_cropped.drop('target',axis=1).iloc[test_splitloc[0]])
        cropped_rand_forrest_train_percent=(np.sum(np.array(df_mn
ist_cropped.iloc[train_splitloc[0]]['target']) == cropped_rand_fo
rrest_train_pred).astype(int)/df_mnist.iloc[train_splitloc[0]].sh
ape[0])*100
        cropped_rand_forrest_test_percent=(np.sum(np.array(df_mni
st_cropped.iloc[test_splitloc[0]]['target']) == cropped_rand_forr
est_test_pred).astype(int)/df_mnist.iloc[test_splitloc[0]].shape[
0])*100
        print ("Random Forrest - Stretched Tree:{} Depth:{}  Trai
n Accuracy:{} Test Accuracy:{}".format(t,d,cropped_rand_forrest_t
rain_percent,cropped_rand_forrest_test_percent))
```