Part 1 Accuracies

Setup	Cross-validation Accuracy
Unprocessed data	0.7656
0-value elements ignored	0.75

Part 1 Code Snippets

1. Calculation of distribution parameters

```
def calculate_mean(data, ignore_missing_value):
    if ignore_missing_value:
        data[data == 0] = np.nan
        mean = np.nanmean(data)
        return mean
    return np.mean(data)

# for each class, each feature, calculate mean and variance
def get_class_feature_summary(train_set, ignore_missing_value):
    summary_df = pd.DataFrame(columns=['Class', 'Feature', 'Mean', 'Var'])
    i = 0
    p_classes, classes = get_class_probobilities_and_names(train_set)
    for label in classes:
        each_class_df = train_set[train_set['Class']==label]
        each_class_df = each_class_df.drop(labels='Class', axis=1)
        for column in each_class_df:
            feature_data = each_class_df[column]
            each_feature_mean = calculate_mean(feature_data, ((column in ['BloodPressure', 'SkinThi
ckness', 'BMI', 'Age']) and ignore_missing_value))
            each_feature_var = np.var(feature_data)
            summary_df.loc[i] = [label, column, each_feature_mean, each_feature_var]
            i = i + 1
    return summary_df
```

2. Calculation of naive Bayes predictions

```
def predict(class_feature_summary, feature_vec):
    p_classes, classes = get_class_probobilities_and_names(train_set)
    probabilities = {}
    for klass in classes:
        log_sum = 0
        for i, feature in enumerate(feature_vec, start=0):
            mean, var = get_mean_var(class_feature_summary, klass, features[i])
            log_sum = log_sum + np.log(norm.pdf(feature, mean , np.sqrt(var)))[0]
        log_sum = log_sum + np.log(p_classes[klass])
        probabilities[klass] = log_sum
    if (probabilities[0] > probabilities[1]):
        return 0
    return 1
```

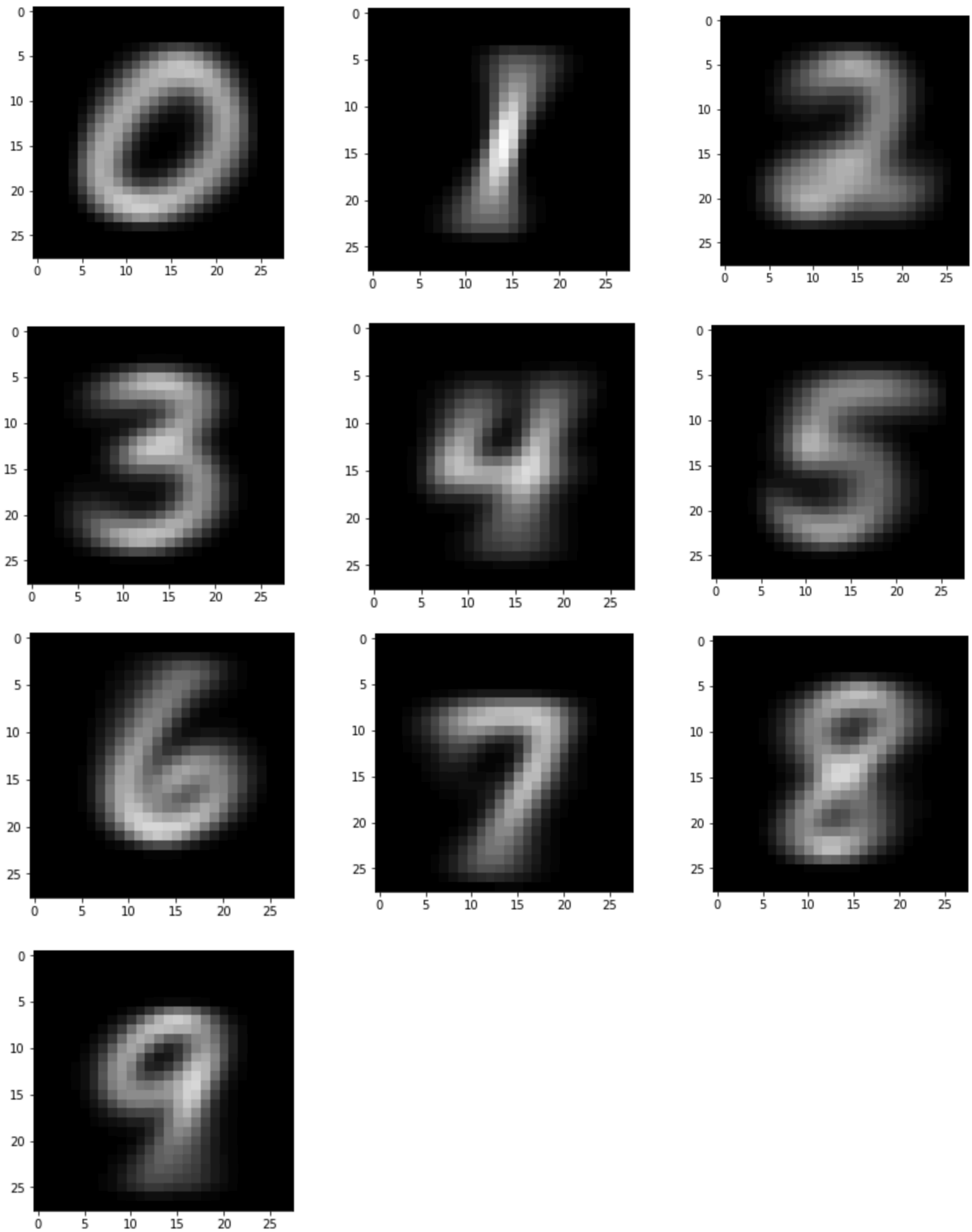
3. Test-train split code

```
def splitData(df, train_test_ratio):
    train_set = df.sample(frac=train_test_ratio)
    test_set = df.sample(frac=(1 - train_test_ratio))
    return train_set, test_set
```

Part 2 MNIST Accuracies

Method	Training Set Accuracy	Test Set Accuracy
Gaussian + untouched	0.7766	0.7824
Gaussian + stretched	0.8268	0.837
Bernoulli + untouched	0.8385	0.8434
Bernoulli + stretched	0.8222	0.8337
10 trees + 4 depth + untouched	0.7047	0.7165
10 trees + 4 depth + stretched	0.7032	0.7164
10 trees + 16 depth + untouched	0.9898	0.9387
10 trees + 16 depth + stretched	0.9947	0.952
30 trees + 4 depth + untouched	0.7298	0.7386
30 trees + 4 depth + stretched	0.7343	0.7466
30 trees + 16 depth + untouched	0.9948	0.9543
30 trees + 16 depth + stretched	0.9971	0.9612

Part 2A Digit Images



Part 2 Code

1. Calculation of the Normal distribution parameters

```
if(distribution_type == 'gaussian'):
    return images_df.apply(lambda x: np.asarray(norm.fit(x)), axis=0)
```

2. Calculation of the Bernoulli distribution parameters

```
if(distribution_type == 'bernoulli'):
    p_list = []
    for c in images_df.columns:
        value_count = images_df[c].value_counts(normalize=True)
        value = value_count.loc[1] if (1 in value_count.index) else 0
        p_list.append(value)
    return pd.Series(p_list)
```

3. Calculation of the Naive Bayes predictions

```
def calculate_likelihood_for_each_label(p_label, feature_vec, params, distribution_type):
    if(distribution_type == 'gaussian'):
        means = params.loc[0]
        stds = params.loc[1]
        likelihood = np.nansum(norm.logpdf(feature_vec, means, stds))
    elif(distribution_type == 'bernoulli'):
        p = params
        likelihood = np.nansum(bernoulli.logpmf(feature_vec, p))

    likelihood = likelihood + np.log(p_label['probability'])
    return np.array([p_label['label'], likelihood])


def get_predict(likelihoods):
    max_row = [float("-inf"), float("-inf")]
    for likelihood in likelihoods:
        if(likelihood[1] > max_row[1]):
            max_row = likelihood
    return max_row[0]
def predict(image, label_params, distribution_type):
    likelihoods = []
    for index, p_train_label in p_train_labels.iterrows():
        params = label_params.loc[p_train_label['label'], :]
        likelihoods.append(calculate_likelihood_for_each_label(p_train_label, image, params, distribution_type))

    return get_predict(np.array(likelihoods))
```

4. Training of a decision tree

```
train_set = train_images
test_set = test_images
if(stretched):
    train_set = train_strech
    test_set = test_strech
classifier.fit(train_set, train_labels)
```

5. Calculation of a decision tree predictions

```
classifier.score(train_set, train_labels), classifier.score(test_set, test_labels)
```

Problem 1: Diabetes Classification

In [19]:

```
import pandas as pd
import numpy as np
from scipy.stats import norm
```

Read data

In [8]:

```
column_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Diab
etesPedigreeFunction', 'Age', 'Class']
features = column_names[:8]
df = pd.read_csv('data/pima-indians-diabetes.csv', names=column_names)
```

In [9]:

```
df.head()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Util functions to split data

In [10]:

```
def splitData(df, train_test_ratio):
    train_set = df.sample(frac=train_test_ratio)
    test_set = df.sample(frac=(1 - train_test_ratio))
    return train_set, test_set
```

In [11]:

```
train_set, test_set = splitData(df, 0.8)
assert test_set.shape[0] + train_set.shape[0] == df.shape[0]
```

In [12]:

```
print(train_set.shape)
train_set.head()
```

```
(614, 9)
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
29	5	117	92	0	0	34.1	0.337	38	0
329	6	105	70	32	68	30.8	0.122	37	0
195	5	158	84	41	210	39.4	0.395	29	1
604	4	183	0	0	0	28.4	0.212	36	1
310	6	80	66	30	0	26.2	0.313	41	0

In [13]:

```
print(test_set.shape)
test_set.head()
```

```
(154, 9)
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
48	7	103	66	32	0	39.1	0.344	31	1
409	1	172	68	49	579	42.4	0.702	28	1
306	10	161	68	23	132	25.5	0.326	47	1
537	0	57	60	0	0	21.7	0.735	67	0
714	3	102	74	0	0	29.5	0.121	32	0

Get labels from data set

In [14]:

```
def get_class_probobilities_and_names(train_set):
    value_counts = train_set.Class.value_counts(normalize=True)
    p_labels = value_counts
    labels = value_counts.index
    return p_labels, labels
```

Calculate params

In [15]:

```
def calculate_mean(data, ignore_missing_value):
    if ignore_missing_value:
        data[data == 0] = np.nan
        mean = np.nanmean(data)
        return mean
    return np.mean(data)

# for each class, each feature, calculate mean and variance
def get_class_feature_summary(train_set, ignore_missing_value):
    summary_df = pd.DataFrame(columns=['Class', 'Feature', 'Mean', 'Var'])
    i = 0
    p_classes, classes = get_class_probobilities_and_names(train_set)
    for label in classes:
        each_class_df = train_set[train_set['Class']==label]
        each_class_df = each_class_df.drop(labels='Class', axis=1)
        for column in each_class_df:
            feature_data = each_class_df[column]
            each_feature_mean = calculate_mean(feature_data, ((column in ['BloodPressure', 'SkinThi
ckness', 'BMI', 'Age']) and ignore_missing_value))
            each_feature_var = np.var(feature_data)
            summary_df.loc[i] = [label, column, each_feature_mean, each_feature_var]
            i = i + 1
    return summary_df
```

Pridict

In [16]:

```
def get_mean_var(df, klass, feature):
    row = df[(df['Class']==klass) & (df['Feature']==feature)]
    return row['Mean'], row['Var']
```

```python
# for each class, get the log p(class|feature_vec) value and return the max
def predict(class_feature_summary, feature_vec):
    p_classes, classes = get_class_probobilities_and_names(train_set)
    probabilities = {}
    for klass in classes:
        log_sum = 0
        for i, feature in enumerate(feature_vec, start=0):
            mean, var = get_mean_var(class_feature_summary, klass, features[i])
            log_sum = log_sum + np.log(norm.pdf(feature, mean , np.sqrt(var)))[0]
        log_sum = log_sum + np.log(p_classes[klass])
        probabilities[klass] = log_sum
    if (probabilities[0] > probabilities[1]):
        return 0
    return 1
```

Evaluate

In [17]:

```python
def calculate_accuracy(actual, predicts):
    TP = 0
    num_total = len(actual)
    for i in range(num_total):
        if actual[i] == predicts[i]:
            TP = TP + 1
    return TP/num_total
```

In [18]:

```python
def get_accuracy_for_one_iteration(ignore_missing_value):
    test_set, test_set = splitData(df, 0.8)
    summary = get_class_feature_summary(train_set, ignore_missing_value)
    predicts = test_set.apply(lambda x: predict(summary, x[:8]), axis=1)
    accuracy = calculate_accuracy(test_set.Class.tolist(), predicts.tolist())
    return accuracy

def get_avg_accuracy(iteration, ignore_missing_value):
    avg_accuracy = 0
    for i in range(iteration):
        print(f'Itr {i + 1}')
        accuracy = get_accuracy_for_one_iteration(ignore_missing_value)
        print(f"accuracy: {accuracy}")
        avg_accuracy = (avg_accuracy * i + accuracy)/(i+1)
        print(f"avg_accuracy: {avg_accuracy}")
        print("\n")
    return avg_accuracy
```

Run 10 times and calculate average accuracy (with missing values)

In [21]:

```python
avg_accuracy1a = get_avg_accuracy(10, ignore_missing_value=False)
```

```
Itr 1
accuracy: 0.7922077922077922
avg_accuracy: 0.7922077922077922


Itr 2
accuracy: 0.7597402597402597
avg_accuracy: 0.775974025974026


Itr 3
accuracy: 0.7727272727272727
avg_accuracy: 0.774891774891775


Itr 4
accuracy: 0.7467532467532467
```

```
avg_accuracy: 0.7678571428571429


Itr 5
accuracy: 0.7857142857142857
avg_accuracy: 0.7714285714285715


Itr 6
accuracy: 0.7532467532467533
avg_accuracy: 0.7683982683982684


Itr 7
accuracy: 0.7857142857142857
avg_accuracy: 0.7708719851576994


Itr 8
accuracy: 0.7532467532467533
avg_accuracy: 0.7686688311688312


Itr 9
accuracy: 0.7402597402597403
avg_accuracy: 0.7655122655122656


Itr 10
accuracy: 0.7662337662337663
avg_accuracy: 0.7655844155844156
```

Run 10 times and calculate average accuracy (without missing values)

In [20]:

```python
avg_accuracy1b = get_avg_accuracy(10, ignore_missing_value=True)
```

```
Itr 1
```

```
/Users/qingemeng/Documents/dev/cs498aml/env/lib/python3.7/site-packages/ipykernel_launcher.py:3: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
accuracy: 0.7467532467532467
avg_accuracy: 0.7467532467532467


Itr 2
accuracy: 0.7337662337662337
avg_accuracy: 0.7402597402597402


Itr 3
accuracy: 0.7597402597402597
avg_accuracy: 0.7467532467532467


Itr 4
accuracy: 0.7467532467532467
avg_accuracy: 0.7467532467532467


Itr 5
accuracy: 0.7207792207792207
avg_accuracy: 0.7415584415584415
```

```
Itr 6
accuracy: 0.7857142857142857
avg_accuracy: 0.7489177489177489


Itr 7
accuracy: 0.7012987012987013
avg_accuracy: 0.7421150278293135


Itr 8
accuracy: 0.7792207792207793
avg_accuracy: 0.7467532467532467


Itr 9
accuracy: 0.7467532467532467
avg_accuracy: 0.7467532467532467


Itr 10
accuracy: 0.7792207792207793
avg_accuracy: 0.75
```

In [ ]:

Problem 2: MNIST Image Classification

In [3]:

```
from mnist import MNIST
from scipy.stats import norm, bernoulli
import numpy as np
import pandas as pd
import math
from PIL import Image
import matplotlib.pyplot as plt
from tqdm.autonotebook import tqdm
tqdm.pandas()
```

```
/usr/local/lib/python3.7/site-packages/tqdm/autonotebook/__init__.py:14: TqdmExperimentalWarning:
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.
g. in jupyter console)
  " (e.g. in jupyter console)", TqdmExperimentalWarning)
```

In [4]:

```
%matplotlib inline
```

Read train and test data

In [5]:

```
# http://yann.lecun.com/exdb/mnist/

mndata = MNIST('data/mnist_data_files')
mndata.gz=True
train_images, train_labels = mndata.load_training()
```

process data and threashold

In [6]:

```
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

In [7]:

```
train_images = (pd.DataFrame(train_images) > 127).astype(np.int)
train_labels = pd.DataFrame(train_labels,columns=["label"])
```

In [8]:

```
# 60000 rows 28*28 pixels
print(train_images.shape)
print(train_labels.shape)
```

```
(60000, 784)
(60000, 1)
```

In [9]:

```
value_counts = train_labels["label"].value_counts(normalize=True)
p_train_labels = pd.DataFrame()
p_train_labels['label'] = value_counts.index
p_train_labels['probability'] = value_counts.values
```

In [10]:

```
p_train_labels.head(10)
```

Out[10]:

	label	probability
0	1	0.112367
1	7	0.104417
2	3	0.102183
3	2	0.099300
4	9	0.099150
5	0	0.098717
6	6	0.098633
7	8	0.097517
8	4	0.097367
9	5	0.090350

In [11]:

```
test_images, test_labels = mndata.load_testing()
```

In [12]:

```
test_images = np.array(test_images)
test_labels = np.array(test_labels)
```

In [13]:

```
test_images = (pd.DataFrame(test_images) > 127).astype(np.int)
test_labels = pd.DataFrame(test_labels,columns=["label"])
```

Image processing

In [14]:

```
def stretch_image(ori_image):
    img = Image.fromarray(np.array(ori_image).reshape(28, 28).astype('uint8'))
    cropped = img.crop(img.getbbox())
    stretched = cropped.resize((28,28))
    return pd.Series(np.array(stretched).reshape(ori_image.shape))

def stretch_images(ori_images):
    print("Stretch images")
    return ori_images.progress_apply(stretch_image, axis=1)
```

In [28]:

```
def plot_mean_images(label_params):
    for index, p_train_label in p_train_labels.iterrows():
        params = label_params.loc[p_train_label['label'], :]
        means = params.loc[0]*255
        img = Image.fromarray(np.array(means).reshape(28, 28).astype('uint8'))
        plt.imshow(img)
        plt.show()
```

In [16]:

```
train_strech = stretch_images(train_images)
test_strech = stretch_images(test_images)
```

```
Stretch images


Stretch images
```

In [17]:

```
def get_params(label_group, distribution_type):
    images_df = label_group.drop(['label'], axis=1)
    if(distribution_type == 'gaussian'):
        return images_df.apply(lambda x: np.asarray(norm.fit(x)), axis=0)
    if(distribution_type == 'bernoulli'):
        p_list = []
        for c in images_df.columns:
            value_count = images_df[c].value_counts(normalize=True)
            value = value_count.loc[1] if (1 in value_count.index) else 0
            p_list.append(value)
        return pd.Series(p_list)
```

Naive Bayes - normal distribution - untouched

In [18]:

```
def calculate_likelihood_for_each_label(p_label, feature_vec, params, distribution_type):
    if(distribution_type == 'gaussian'):
        means = params.loc[0]
        stds = params.loc[1]
        likelihood = np.nansum(norm.logpdf(feature_vec, means, stds))
    elif(distribution_type == 'bernoulli'):
        p = params
        likelihood = np.nansum(bernoulli.logpmf(feature_vec, p))

    likelihood = likelihood + np.log(p_label['probability'])
    return np.array([p_label['label'], likelihood])


def get_predict(likelihoods):
    max_row = [float("-inf"), float("-inf")]
    for likelihood in likelihoods:
        if(likelihood[1] > max_row[1]):
            max_row = likelihood
    return max_row[0]
```

Evaluate

In [19]:

```
def predict(image, label_params, distribution_type):
    likelihoods = []
    for index, p_train_label in p_train_labels.iterrows():
        params = label_params.loc[p_train_label['label'], :]
        likelihoods.append(calculate_likelihood_for_each_label(p_train_label, image, params, distri
bution_type))

    return get_predict(np.array(likelihoods))

# predict(test_images.loc[1])
```

In [20]:

```
def calculate_accuracy(actual, predicts):
    TP = 0
    num_total = len(actual)
    for i in range(num_total):
        if actual[i] == predicts[i]:
            TP = TP + 1
    return TP/num_total
```

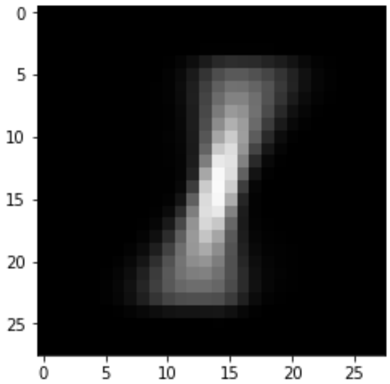
Entry point

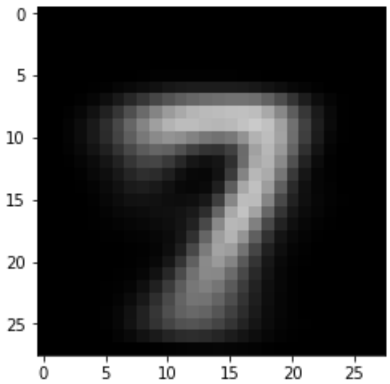
In [32]:

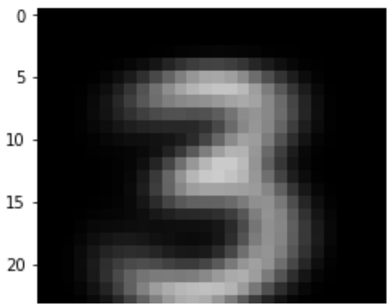
```
def accuracy(distribution, stretched, is_ploting_images=False):
    train_set = train_images
    test_set = test_images
    if(stretched):
        train_set = train_strech
        test_set = test_strech

    train_df = train_set.join(train_labels)
    label_params = train_df.groupby(['label']).apply(lambda x: get_params(x, distribution))
    if(distribution == 'gaussian'):
        assert label_params.shape == (20, 784)
        if(stretched == False and is_ploting_images):
            plot_mean_images(label_params)
            return

    if(distribution == 'bernoulli'):
        assert label_params.shape == (10, 784)

    print('Get predicts...')
    predicts_train = train_set.progress_apply(predict, args=(label_params, distribution, ), axis=1)
    predicts_test = test_set.progress_apply(predict, args=(label_params, distribution, ), axis=1)

    return (calculate_accuracy(np.array(train_labels), np.array(predicts_train)), calculate_accurac
y(np.array(test_labels), np.array(predicts_test)))
```

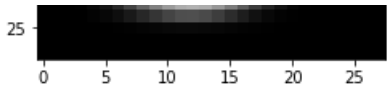
In [33]:

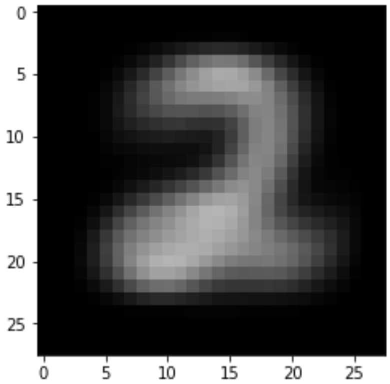
```
accuracy_norm_origin = accuracy('gaussian', False, is_ploting_images=True)
```

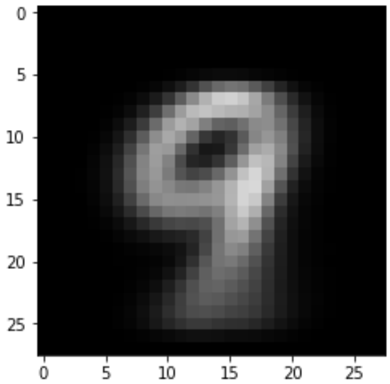


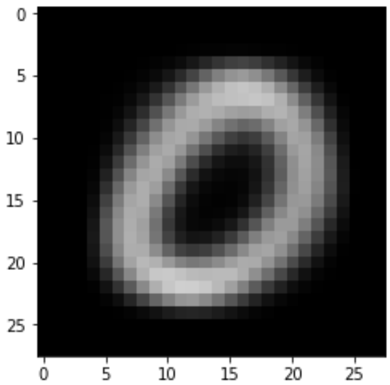


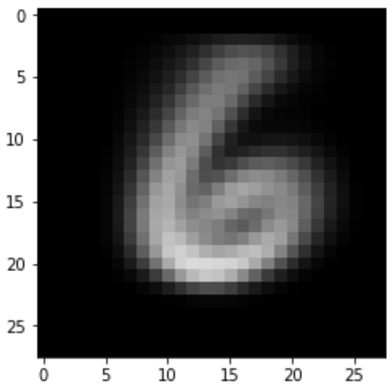


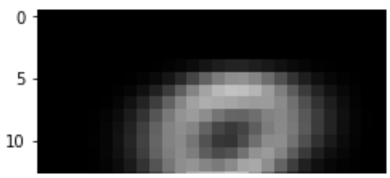

25
0 5 10 15 20 25

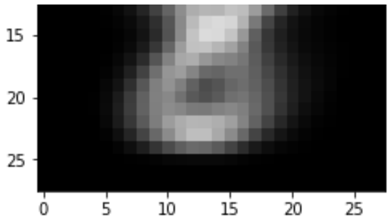

0 5 10 15 20 25
0 5 10 15 20 25

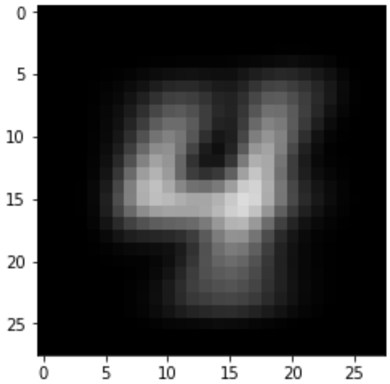

0 5 10 15 20 25
0 5 10 15 20 25

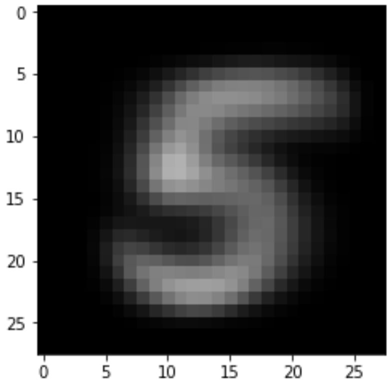

0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10







In [ ]:

```
accuracy_norm_origin = accuracy('gaussian', False)
accuracy_norm_stretched = accuracy('gaussian', True)
accuracy_bernoulli_origin = accuracy('bernoulli', False)
accuracy_bernoulli_streched = accuracy('bernoulli', True)
```

In [20]:

```
accuracy_norm_origin, accuracy_norm_stretched, accuracy_bernoulli_origin,
accuracy_bernoulli_streched
```

Out[20]:

```
((0.7765833333333333, 0.7824),
 (0.8267666666666666, 0.837),
 (0.8385333333333334, 0.8434),
 (0.8221666666666667, 0.8337))
```

RandomForestClassifier

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [22]:

```
def rfc_accuracy(classifier, stretched):
    train_set = train_images
```

```python
    test_set = test_images
    if(stretched):
        train_set = train_strech
        test_set = test_strech
    classifier.fit(train_set, train_labels)
    return classifier.score(train_set, train_labels), classifier.score(test_set, test_labels)
```

In [23]:

```python
classifier10_4 = RandomForestClassifier(n_estimators=10, max_depth=4, n_jobs=10)
classifier10_16 = RandomForestClassifier(n_estimators=10, max_depth=16, n_jobs=10)
classifier30_4 = RandomForestClassifier(n_estimators=30, max_depth=4, n_jobs=10)
classifier30_16 = RandomForestClassifier(n_estimators=30, max_depth=16, n_jobs=10)
```

In [24]:

```python
acc_rfc1 = rfc_accuracy(classifier10_4, stretched = True)
acc_rfc2 = rfc_accuracy(classifier10_16, stretched = True)
acc_rfc3 = rfc_accuracy(classifier30_4, stretched = True)
acc_rfc4 = rfc_accuracy(classifier30_16, stretched = True)
acc_rfc5 = rfc_accuracy(classifier10_4, stretched = False)
acc_rfc6 = rfc_accuracy(classifier10_16, stretched = False)
acc_rfc7 = rfc_accuracy(classifier30_4, stretched = False)
acc_rfc8 = rfc_accuracy(classifier30_16, stretched = False)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  import sys
```

In [25]:

```python
acc_rfc1, acc_rfc2, acc_rfc3, acc_rfc4, acc_rfc5, acc_rfc6, acc_rfc7, acc_rfc8
```

Out[25]:

```
((0.7032333333333334, 0.7164),
 (0.99465, 0.952),
 (0.7343333333333333, 0.7466),
 (0.9971333333333333, 0.9612),
 (0.7046833333333333, 0.7165),
 (0.9898166666666667, 0.9387),
 (0.7298166666666667, 0.7386),
 (0.9947666666666667, 0.9543))
```

In [ ]:

```

```