project

May 9, 2024

```
[ ]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from matplotlib.colors import LogNorm
     from mpl_toolkits import mplot3d

     from sklearn.model_selection import StratifiedKFold
     from sklearn.mixture import GaussianMixture
     from sklearn.cluster import KMeans
     from sklearn.neighbors import KernelDensity
     from sklearn.metrics import accuracy_score, confusion_matrix
     import seaborn as sns

     import random
     ## Set random seed for reproducibility
     def set_all_seeds(RANDOM_SEED):
         random.seed(RANDOM_SEED)
         np.random.seed(RANDOM_SEED)

     set_all_seeds(42)
```

```
[ ]: def get_digit_data(data, digit):
         digit_index = np.where(data.labels == digit)[0]
         digit_data = data.dataset[digit_index[0]]
         for i in range(1, len(digit_index)):
                 digit_data = np.concatenate((digit_data, data.
      ↪dataset[digit_index[i]]))
         return digit_data
```

```
[ ]: class Arabic_Digit:
         def __init__(self, file, block_size, name):
             self.name = name
             with open(file, 'r') as f:
                 lines = f.readlines()
                 index = -1
                 block = 0
                 digit = 0
                 gender = True
```

```
            genders = [] ## M (True) / F (False)
            digits = [] ## Digits (0 ... 9)
            dataset = []
            MFCCs = [] ## 13 features
            data = []
            for line in lines:
                if line.strip() == "":
                    index += 1
                    if index != 0:
                        block += 1
                        dataset.append(np.array(MFCCs))
                        digits.append(digit)
                        genders.append(gender)
                        MFCCs = []
                        if block % block_size == 0:
                            digit += 1
                            block = 0
                            gender = not gender
                        elif block % (block_size // 2) == 0:
                            gender = not gender
                    continue

                line = line.split()
                line = [float(value) for value in line]
                MFCCs.append(line)
            dataset.append(np.array(MFCCs))
            digits.append(digit)
            genders.append(gender)
        self.dataset = dataset
        self.labels = np.array(digits)
        self.genders = np.array(genders)

    def plot_multi_MFCCs(self, samples, MFCCs, figsize):
        fig, axes = plt.subplots(len(samples), 1, figsize=figsize)
        for idx, sample in enumerate(samples):
            for MFCC in MFCCs:
                axes[idx].plot(self.dataset[sample].T[MFCC], label=f'MFCC {MFCC + 1}')
            axes[idx].set_xlabel('Frame Index')
            axes[idx].set_ylabel('MFCC Value')
            axes[idx].legend()
            axes[idx].set_title(f'MFCCs for Digit {self.labels[sample]}, Sample No. {sample}')
        plt.tight_layout()
        plt.show()

    def plot_MFCCs(self, sample, MFCCs, figsize):
```

```
        plt.figure(figsize=figsize)
        for MFCC in MFCCs:
            plt.plot(self.dataset[sample].T[MFCC], label=f'MFCC {MFCC + 1}')
        plt.xlabel('Frame Index')
        plt.ylabel('MFCC Value')
        plt.legend()
        plt.title(f'MFCCs for Digit {self.labels[sample]}, Sample No. {sample}')
        plt.show()

    def plot_pairwise_digit_contour(self, digit, MFCCs, xlim, ylim, clf,␣
↪test_data=None):
        assert len(MFCCs) == 2
        digit_data = get_digit_data(self, digit)[:, [MFCCs[0], MFCCs[1]]]
        clf.fit(digit_data)
        X, Y = np.meshgrid(np.linspace(xlim[0], xlim[1]), np.linspace(ylim[0],␣
↪ylim[1]))
        Z = (-clf.score_samples(np.array([X.ravel(), Y.ravel()]).T)).reshape(X.
↪shape)

        CS = plt.contour(
            X, Y, Z, norm=LogNorm(vmin=1.0, vmax=1000.0), levels=np.logspace(0,␣
↪3, 10)
        )

        CB = plt.colorbar(CS, shrink=0.8, extend="both")
        plt.scatter(digit_data[:, 0], digit_data[:, 1], s=1, label=self.name)

        if test_data != None:
            test_X = get_digit_data(test_data, digit)[:, [MFCCs[0], MFCCs[1]]]
            plt.scatter(test_X[:, 0], test_X[:, 1], s=1, c='orange', alpha=0.4,␣
↪label=test_data.name)

        plt.scatter(clf.means_[:, 0], clf.means_[:, 1], s=75, marker='x',␣
↪c="yellow")
        plt.title(f"Contour plot of GMM model for Digit: {digit}")
        plt.xlabel(f"MFCC {MFCCs[0] + 1}")
        plt.ylabel(f"MFCC {MFCCs[1] + 1}")
        plt.axis("tight")
        plt.legend()
        plt.show()

    def plot_pairwise_sample_scatter(self, sample, MFCCs):
        assert len(MFCCs) == 2

        plt.scatter(self.dataset[sample][MFCCs[0]], self.
↪dataset[sample][MFCCs[1]])
```

```python
        plt.title(f"Pairwise MFCCs scatter plot for Digit {self.
↪labels[sample]}, Sample No. {sample}")
        plt.xlabel(f"MFCC {MFCCs[0] + 1}")
        plt.ylabel(f"MFCC {MFCCs[1] + 1}")
        plt.show()

    def plot_pairwise_digit_scatter(self, digit, MFCCs, xlim=None, ylim=None):
        assert len(MFCCs) == 2
        digit_data = get_digit_data(self, digit)

        plt.scatter(digit_data.T[MFCCs[0]], digit_data.T[MFCCs[1]], s=1)
        plt.title(f"Pairwise MFCCs scatter plot for all Digit {digit}")
        plt.xlabel(f"MFCC {MFCCs[0] + 1}")
        plt.ylabel(f"MFCC {MFCCs[1] + 1}")

        ### Optional configuration of the plot
        if xlim != None:
            plt.xlim(xlim)
        if ylim != None:
            plt.ylim(ylim)
        plt.show()

    def plot_digit_3D_scatter(self, digit, MFCCs, figsize=None, xlim=None,␣
↪ylim=None, zlim=None):
        assert len(MFCCs) == 3
        digit_data = get_digit_data(self, digit)

        fig=plt.figure(figsize=figsize)
        ax = plt.axes(projection='3d')
        ax.scatter3D(digit_data.T[MFCCs[0]], digit_data.T[MFCCs[1]], digit_data.
↪T[MFCCs[2]], s=1)
        ax.set_xlabel(f"MFCC {MFCCs[0] + 1}")
        ax.set_ylabel(f"MFCC {MFCCs[1] + 1}")
        ax.set_zlabel(f"MFCC {MFCCs[2] + 1}")
        plt.title(f"3D MFCCs scatter plot for all Digit: {digit}")
        if xlim != None:
            ax.set_xlim(xlim)
        if ylim != None:
            ax.set_ylim(ylim)
        if zlim != None:
            ax.set_zlim(ylim)

        ax.set_box_aspect(None, zoom=0.8)

        plt.show()
```

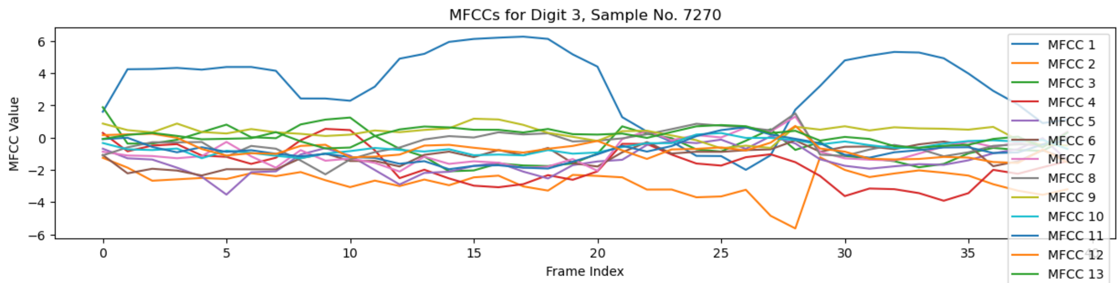
[ ]:
```python
class DigitModel():
    def __init__(self, n_components, covariance_type, max_iter):
        self.joint_models = [GaussianMixture(n_components=component,
                             covariance_type=cov, max_iter=iter)
                             for component, cov, iter in
                             zip(n_components, covariance_type, max_iter)]

    def fit_single(self, X, y, digit):
        assert np.all(y=digit)
        self.joint_models[digit].fit(X)

    def fit(self, X, y):
        for digit in range(10):
            digit_index = np.where(y == digit)[0]
            digit_data = X[digit_index[0]]
            for i in range(1, len(digit_index)):
                digit_data = np.concatenate((digit_data, X[digit_index[i]]))

            self.joint_models[digit].fit(digit_data)

    def predict(self, X):
        result = []
        for x in X:
            result.append(self.maximum_likelihood_classification(x))
        return result

    def maximum_likelihood_classification(self, sample):
        log_likelihood = []
        for digit in range(10):
            log_likelihood.append(self.likelihood_digit(sample, digit))
        return np.argmax(log_likelihood)

    def likelihood_digit(self, sample, digit):
        return np.sum(self.joint_models[digit].score_samples(sample)) ## For now
```

[ ]:
```python
train_file = "spoken+arabic+digit/Train_Arabic_Digit.txt"
test_file =  "spoken+arabic+digit/Test_Arabic_Digit.txt"
train_data = Arabic_Digit(train_file, 660, "Train Data")
test_data = Arabic_Digit(test_file, 220, "Test Data")

total_data = Arabic_Digit(train_file, 660, "Total Data")
total_data.dataset += test_data.dataset
total_data.labels = np.concatenate((total_data.labels, test_data.labels))
total_data.genders = np.concatenate((total_data.genders, test_data.genders))

X, y= train_data.dataset, train_data.labels
test_X, test_y = test_data.dataset, np.array(test_data.labels)
```

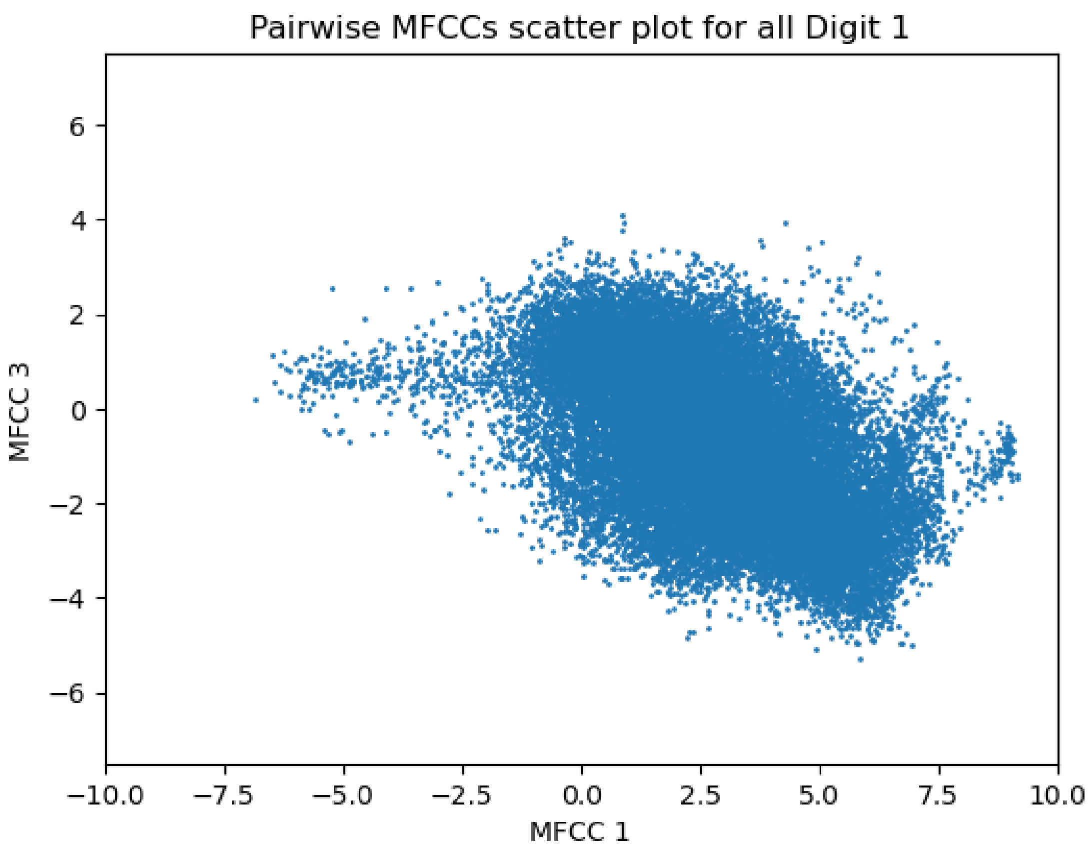
```
[ ]:
```

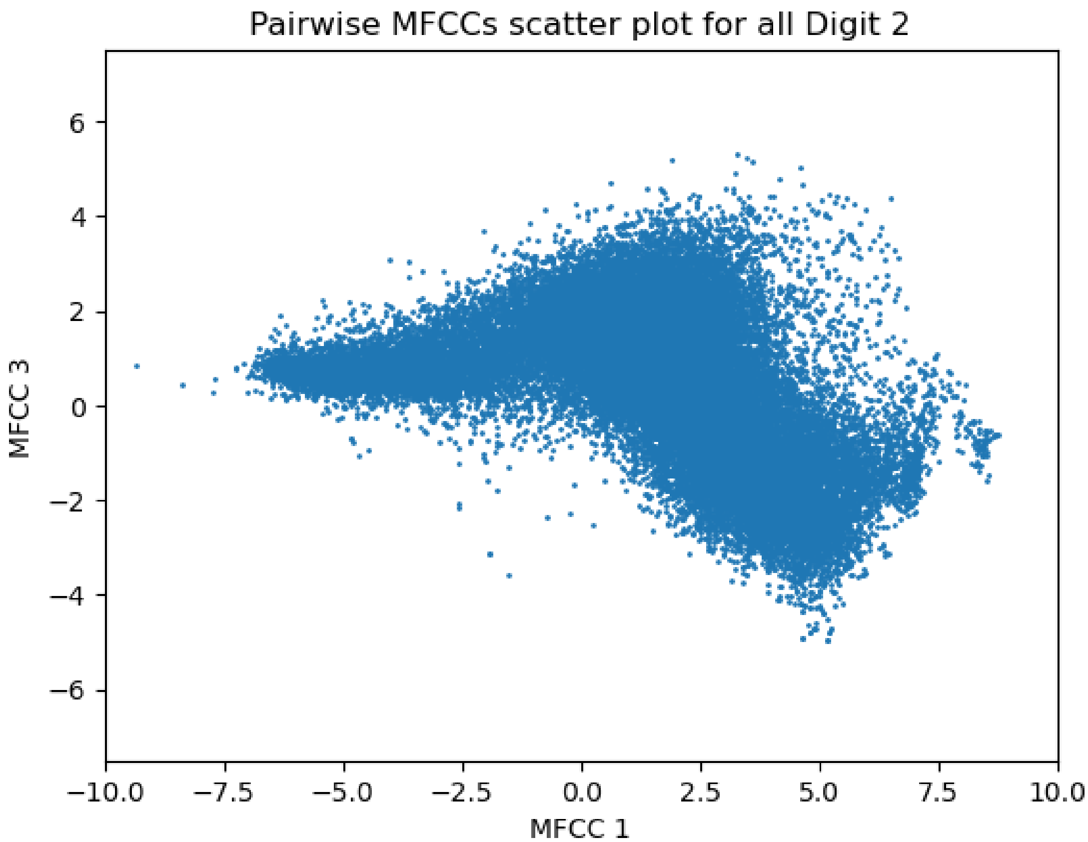
0.1 Data Exploration

```
[ ]: total_data.plot_MFCCs(np.random.randint(0,8800),
     [0,1,2,3,4,5,6,7,8,9,10,11,12], figsize=(15,3))
```

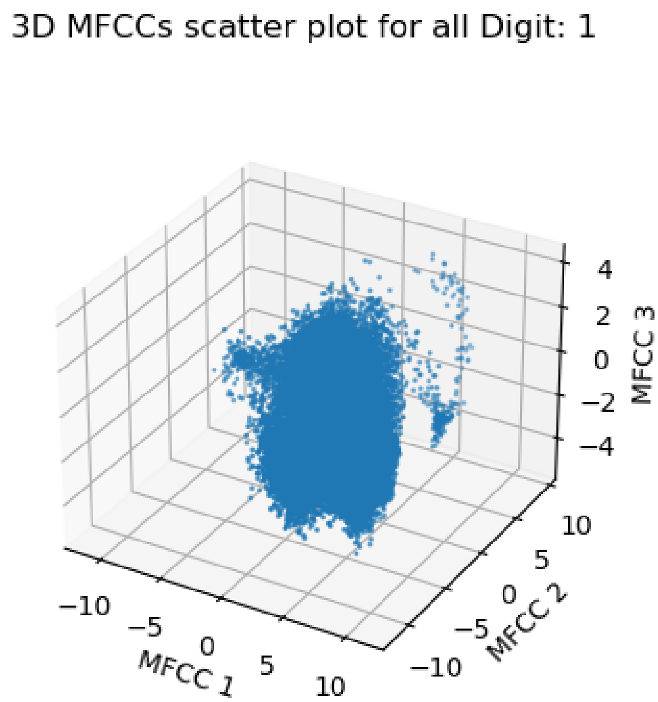


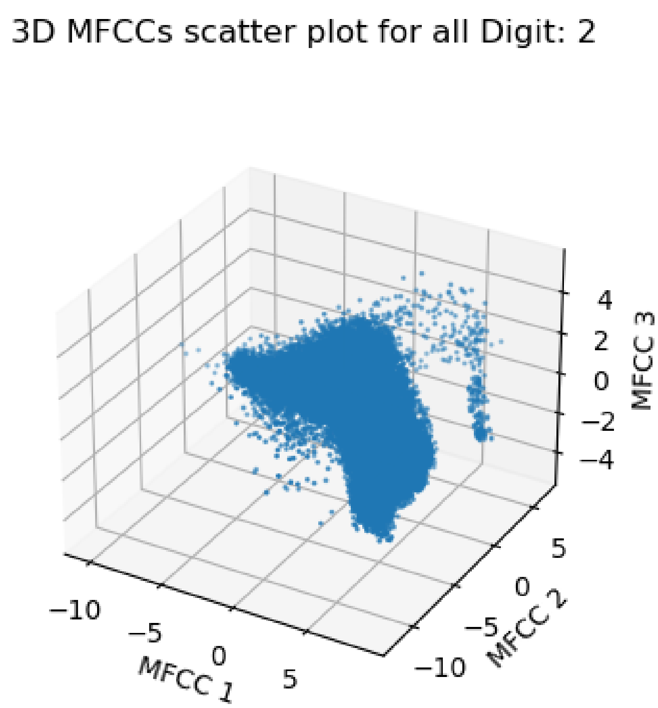

```
[ ]: total_data.plot_pairwise_digit_scatter(1, [0,2], xlim=(-10,10), ylim=(-7.5,7.5))
     total_data.plot_pairwise_digit_scatter(2, [0,2], xlim=(-10,10), ylim=(-7.5,7.5))
```



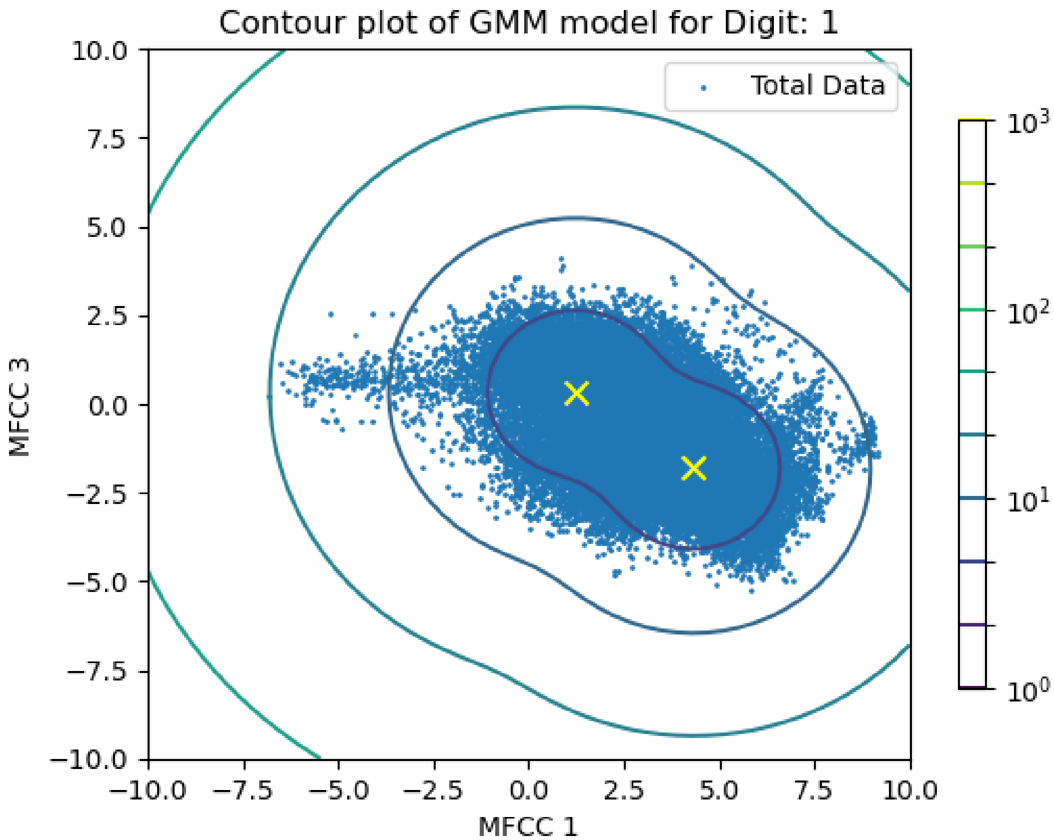


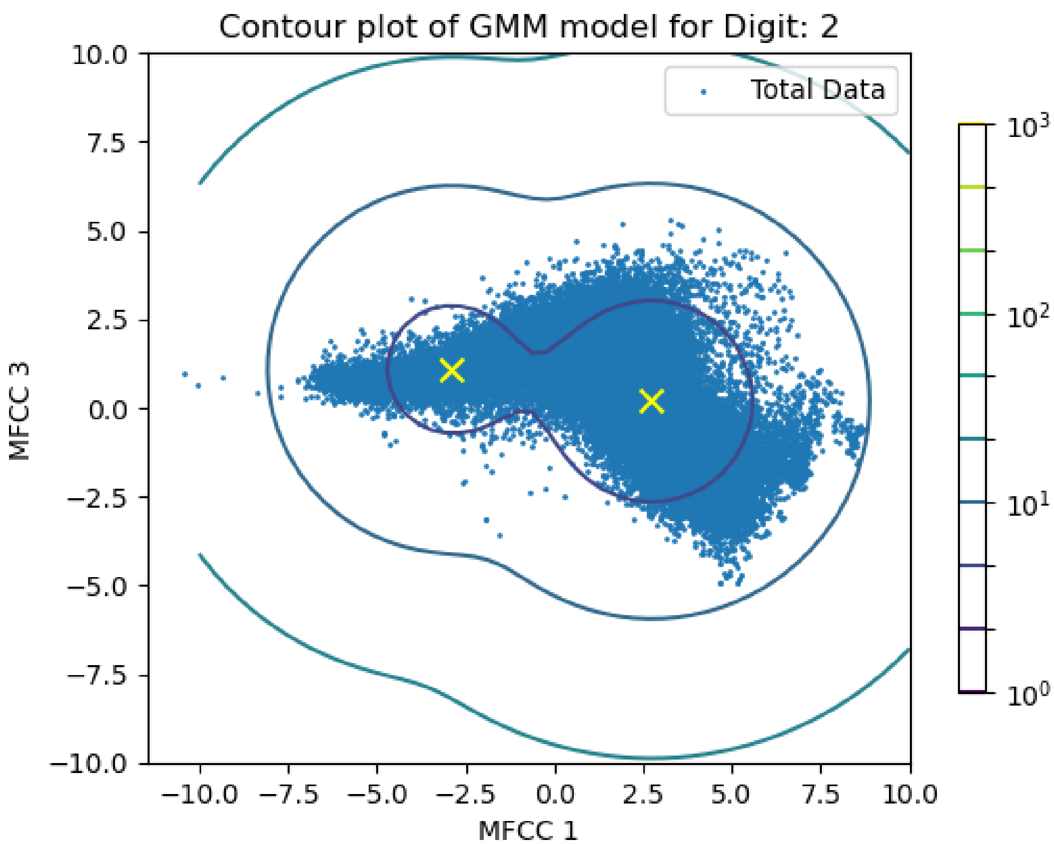
[ ]:
```
total_data.plot_digit_3D_scatter(1, [0,1,2], xlim=(-12.5,12.5))
total_data.plot_digit_3D_scatter(2, [0,1,2])
```


3D MFCCs scatter plot for all Digit: 1
MFCC 1
MFCC 2
MFCC 3
−10
−5
0
5
10

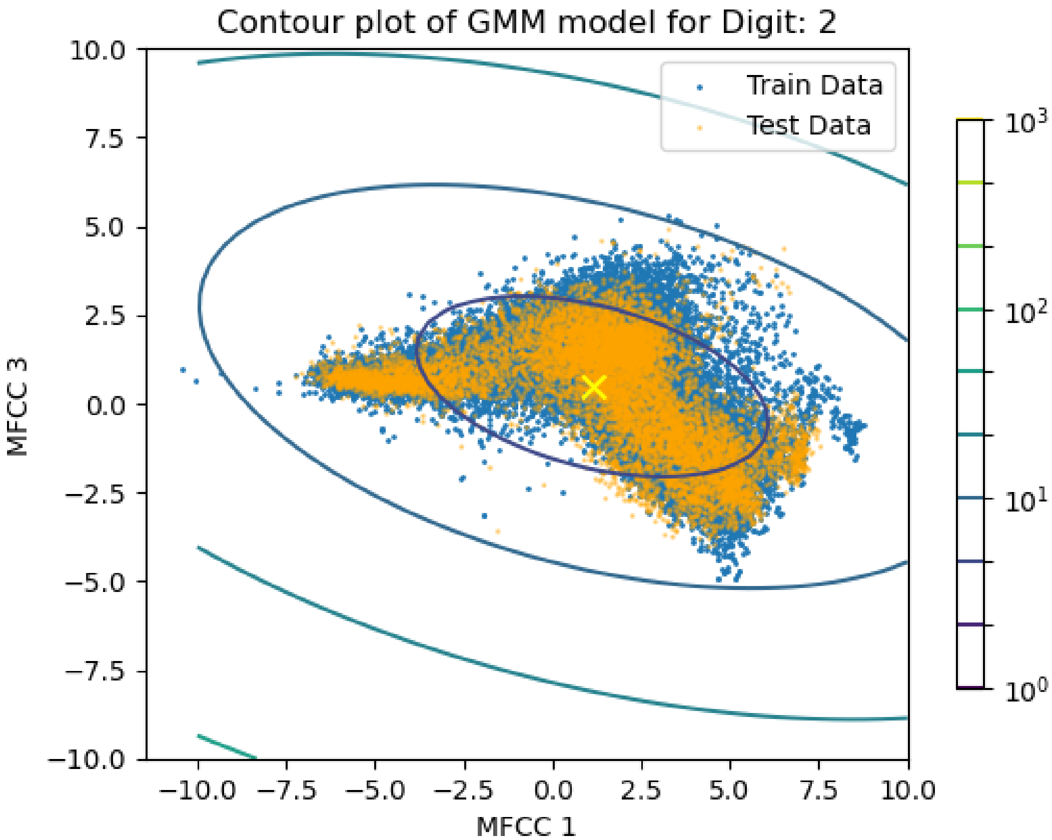

3D MFCCs scatter plot for all Digit: 2
MFCC 1
MFCC 2
MFCC 3
−10
−5
0
5

[ ]:
```
clf = GaussianMixture(n_components=2, max_iter=0, covariance_type="spherical")
total_data.plot_pairwise_digit_contour(1, [0,2], (-10,10), (-10,10), clf)
total_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf)
```

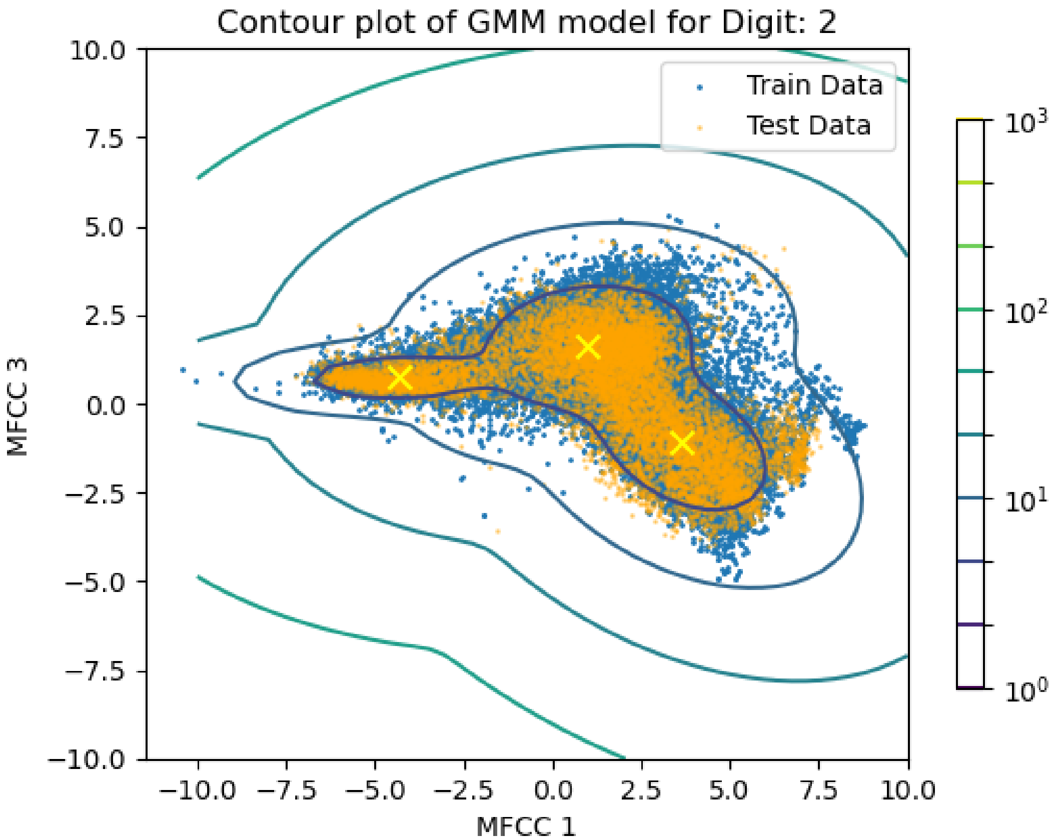




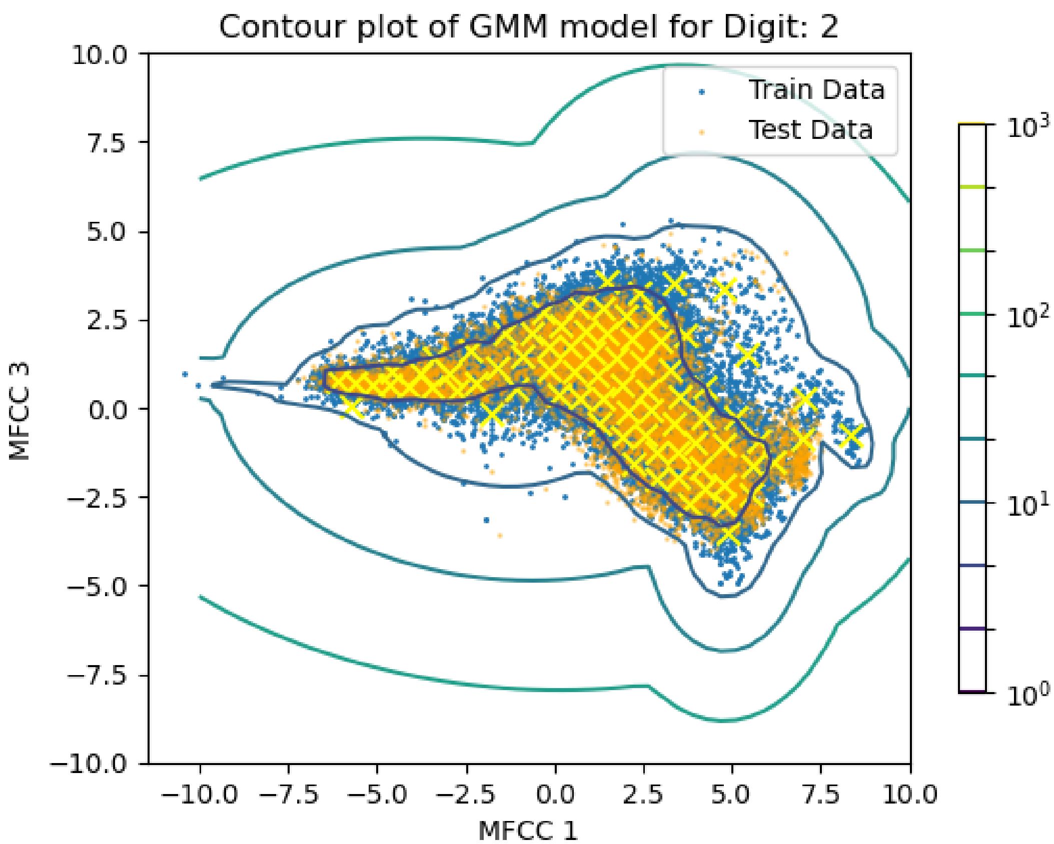
[ ]:
```
clf = GaussianMixture(n_components=1)
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
 ↪test_data)
clf = GaussianMixture(n_components=3)
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
 ↪test_data)
clf = GaussianMixture(n_components=80)
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
 ↪test_data)
```



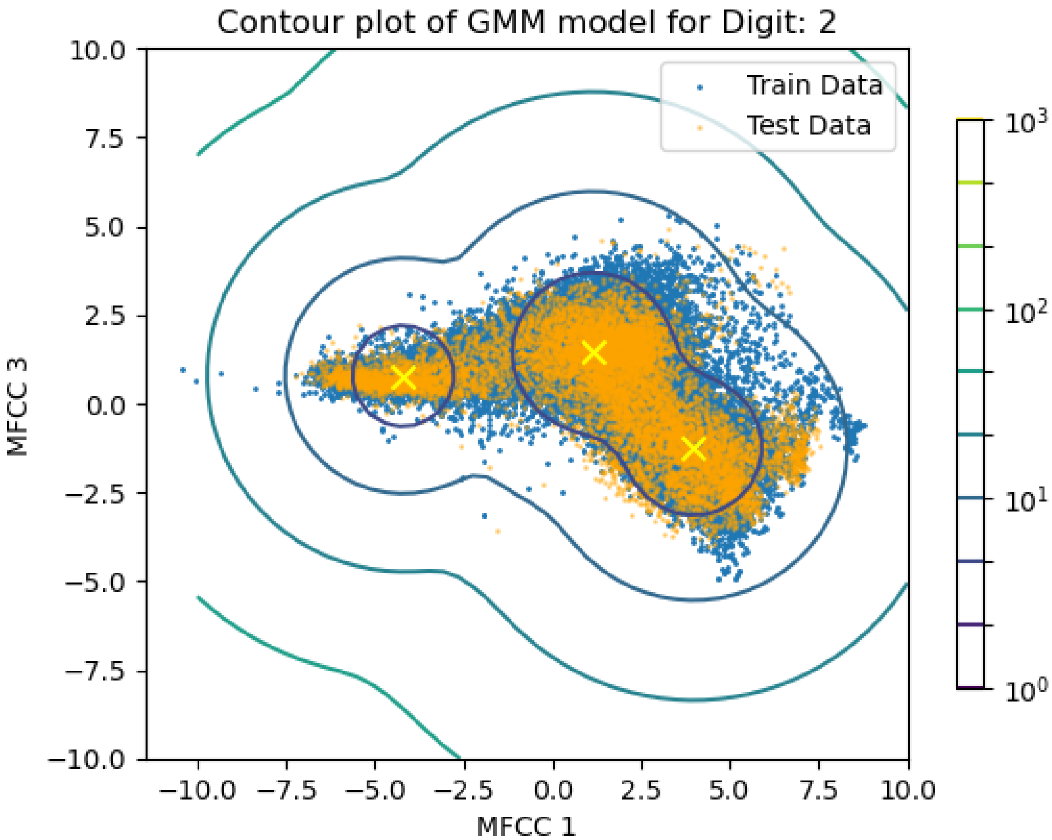
Contour plot of GMM model for Digit: 2
Train Data
Test Data
MFCC 1
MFCC 3

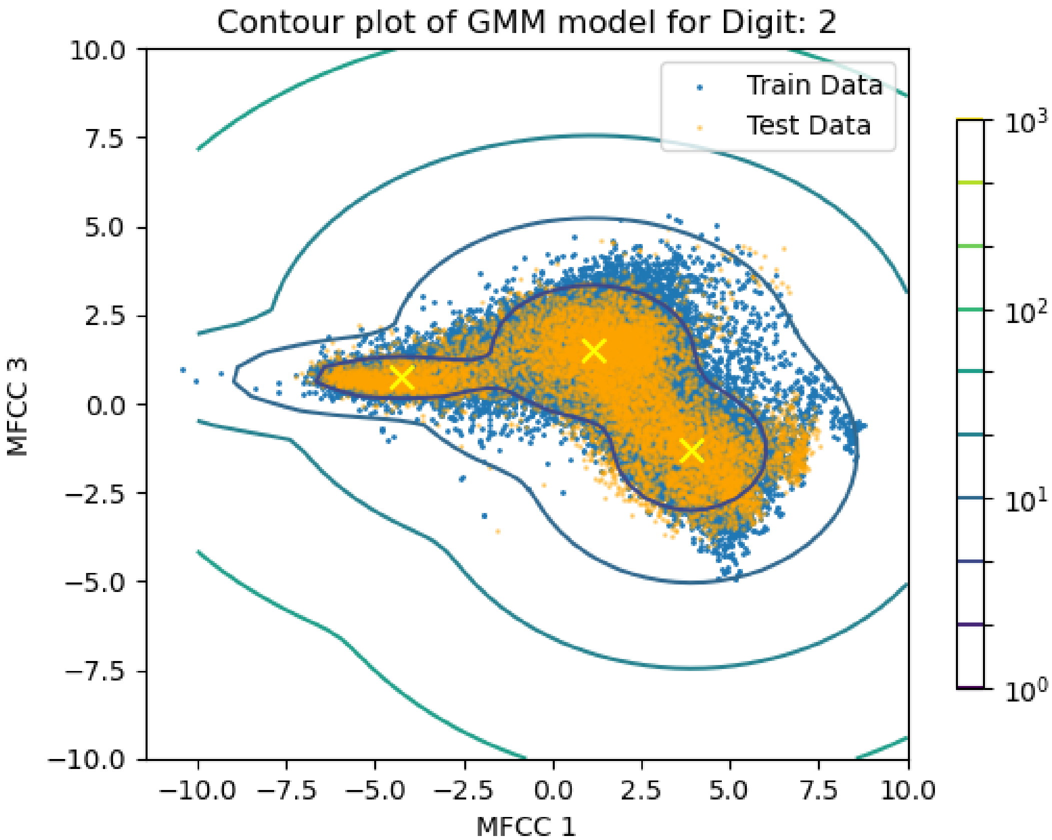


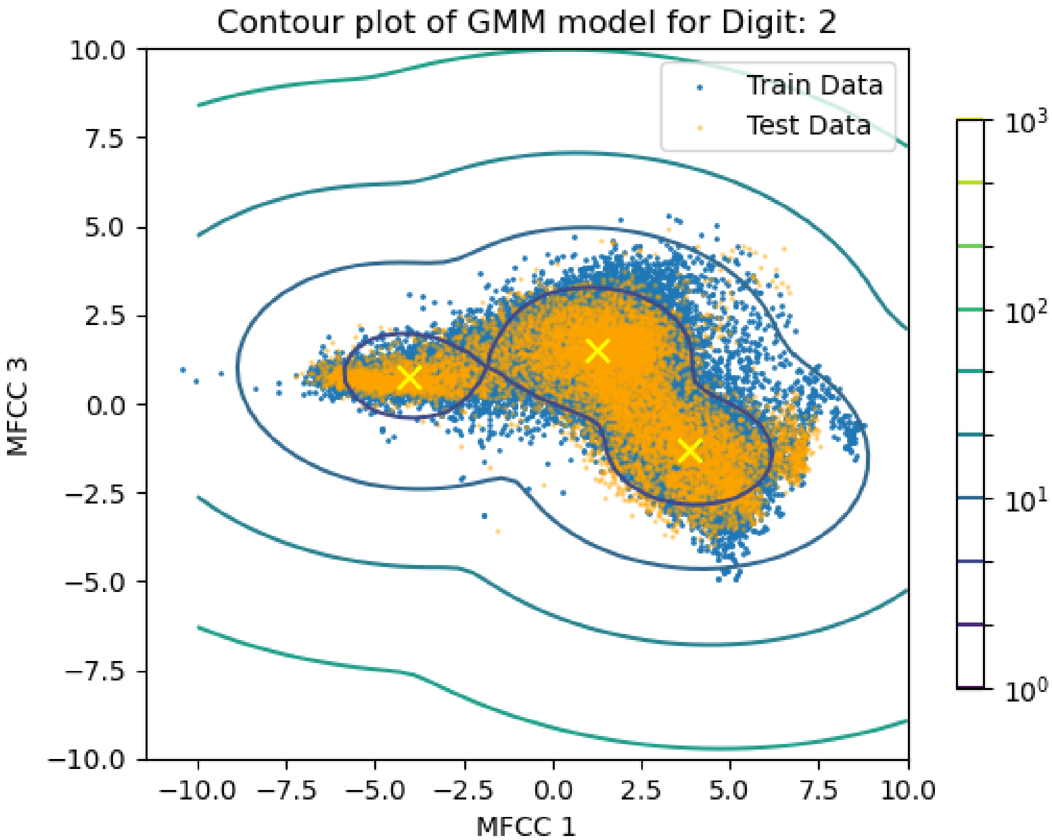
Contour plot of GMM model for Digit: 2
Train Data
Test Data
MFCC 3
MFCC 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10^3
10^2
10^1
10^0

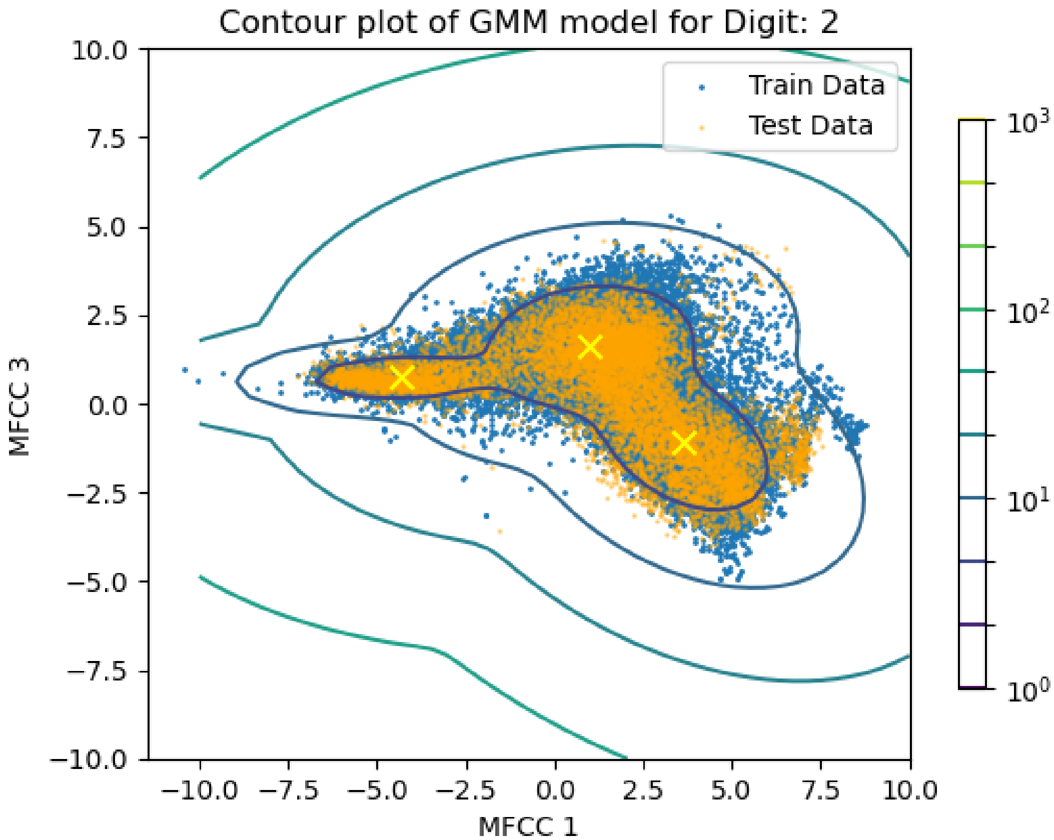


[ ]:
```
clf = GaussianMixture(n_components=3, covariance_type="spherical")
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
  ↪test_data)
clf = GaussianMixture(n_components=3, covariance_type="diag")
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
  ↪test_data)
clf = GaussianMixture(n_components=3, covariance_type="tied")
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
  ↪test_data)
clf = GaussianMixture(n_components=3, covariance_type="full")
train_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf,␣
  ↪test_data)
```

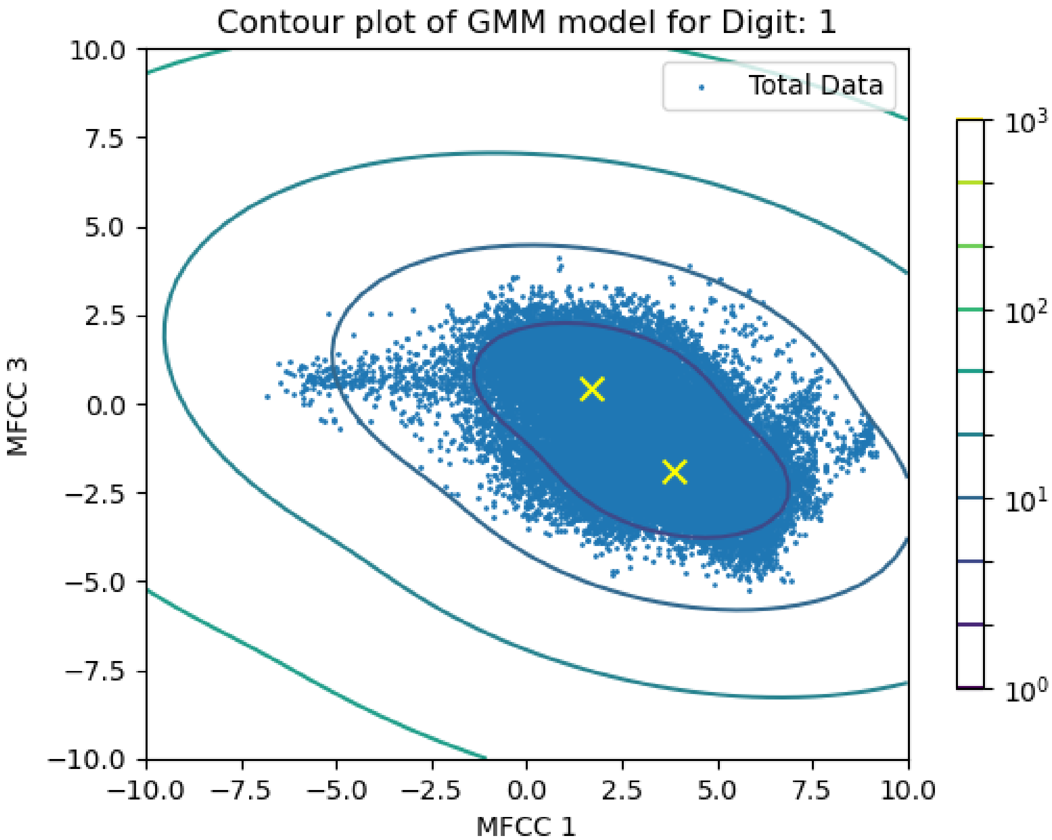

Contour plot of GMM model for Digit: 2
Train Data
Test Data
MFCC 1
MFCC 3
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10^3
10^2
10^1
10^0

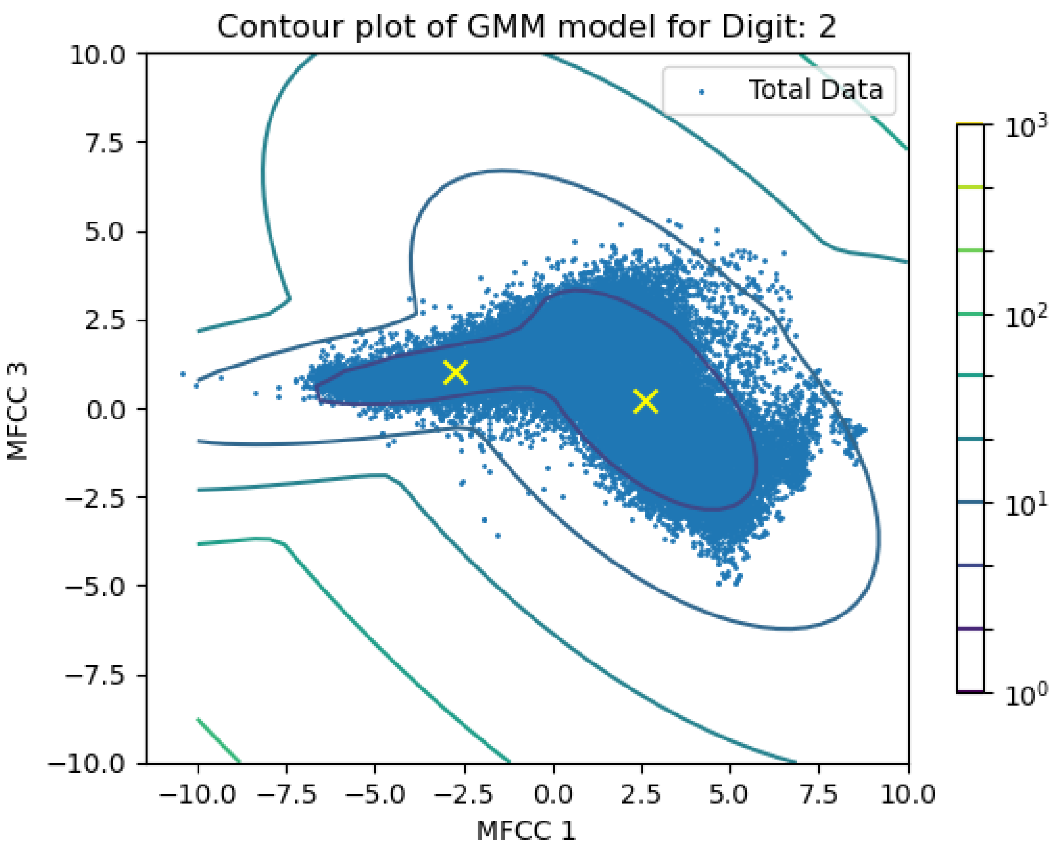

Contour plot of GMM model for Digit: 2
Train Data
Test Data
MFCC 1
MFCC 3
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10^3
10^2
10^1
10^0


Contour plot of GMM model for Digit: 2
Train Data
Test Data
MFCC 3
MFCC 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10^3
10^2
10^1
10^0



[ ]:
```
clf = GaussianMixture(n_components=2)
total_data.plot_pairwise_digit_contour(1, [0,2], (-10,10), (-10,10), clf)
total_data.plot_pairwise_digit_contour(2, [0,2], (-10,10), (-10,10), clf)
```


Contour plot of GMM model for Digit: 1
Total Data
MFCC 3
MFCC 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10^3
10^2
10^1
10^0

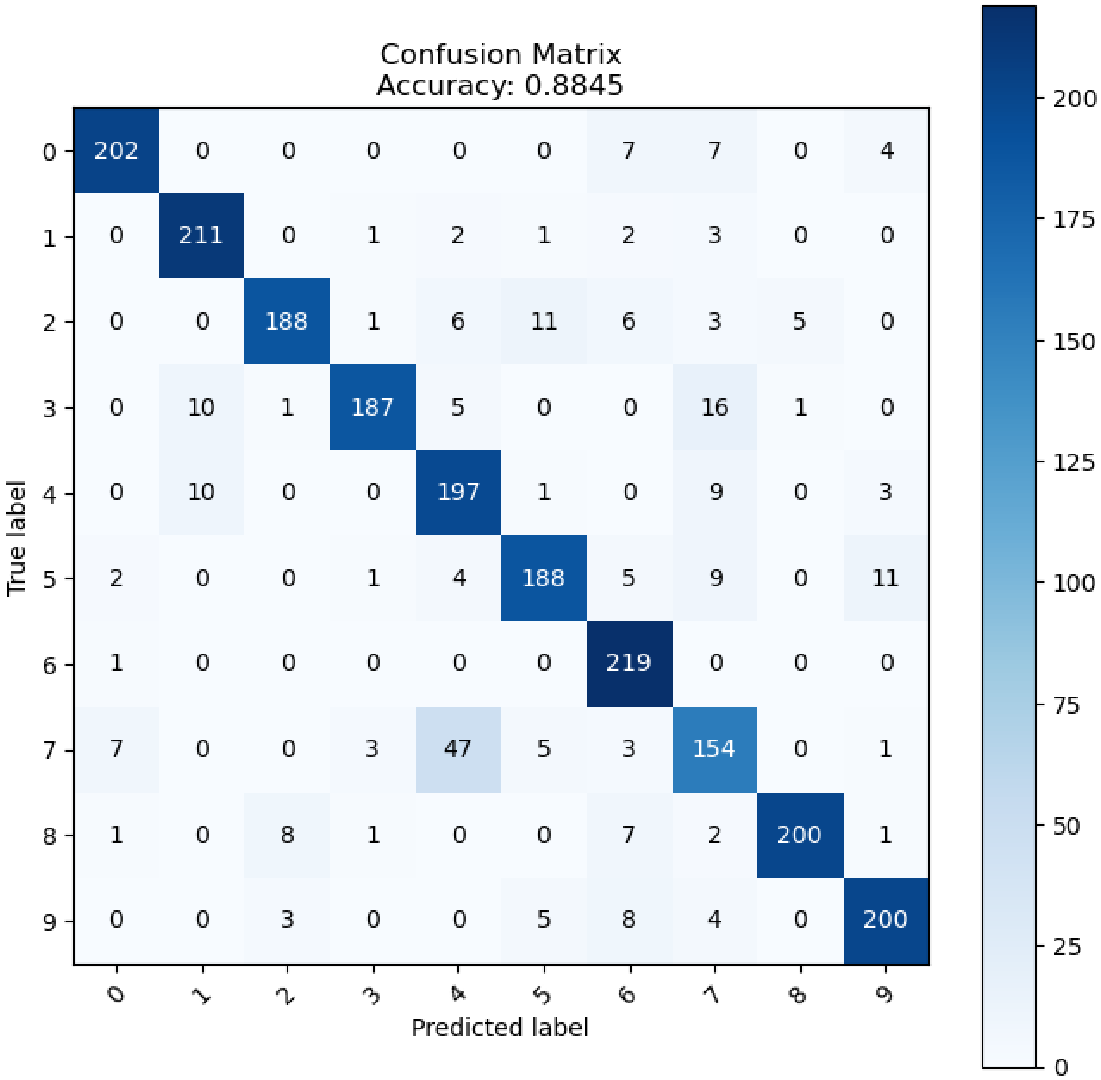


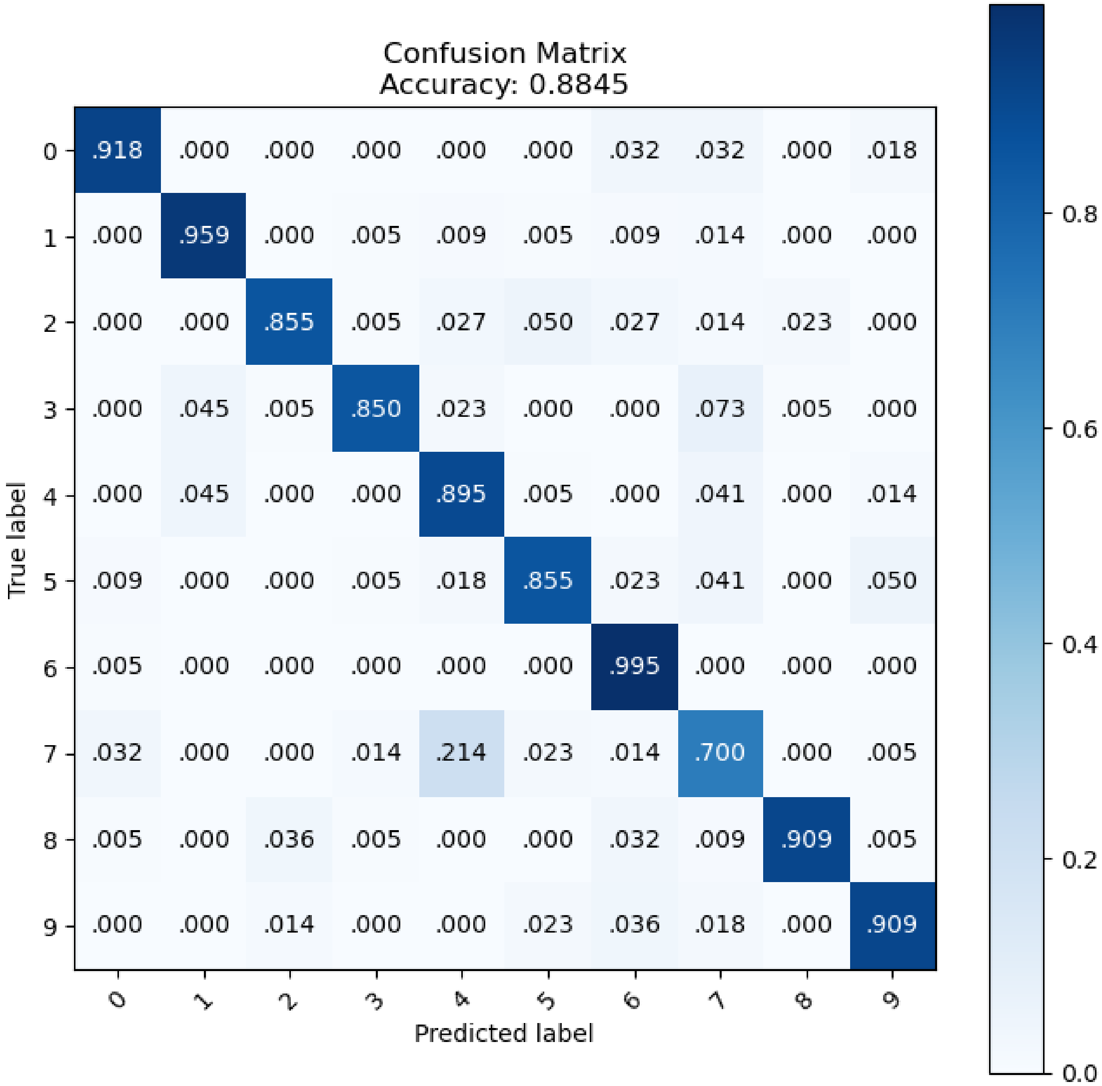

```
[ ]: def cross_val_eval(clf, X, y, cv=5, shuffle=True):
         skf = StratifiedKFold(n_splits=cv, shuffle=shuffle)
         scores = []
         predictions = []
         for i, (train_index, test_index) in enumerate(skf.split(X, y)):
             X_train = [X[i] for i in train_index]
             y_train = y[train_index]
             X_test = [X[i] for i in test_index]
             y_test = y[test_index]

             clf.fit(X_train, y_train)
             pred_y = clf.predict(X_test)

             predictions.append(pred_y)
             scores.append(accuracy_score(y_test, pred_y))

         return scores

     def print_conf(true, pred, normalize=None):
         conf_matrix = confusion_matrix(true, pred, normalize=normalize)
```

```
    accuracy = accuracy_score(true, pred)

    # Plot the confusion matrix
    plt.figure(figsize=(8,8))
    plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix\nAccuracy: {accuracy:.4f}')

    plt.colorbar()

    # Setting the ticks
    classes = [i for i in range(10)]
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    # Display the values inside the plot
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            value = conf_matrix[i, j]
            if normalize is not None:
                value = ("%.3f" % value).lstrip('0')
            plt.text(j, i, value,
                    horizontalalignment="center",
                    verticalalignment="center",
                    color="white" if conf_matrix[i, j] > conf_matrix.max() / 2.
↪0 else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

0.1.1 Baseline Model

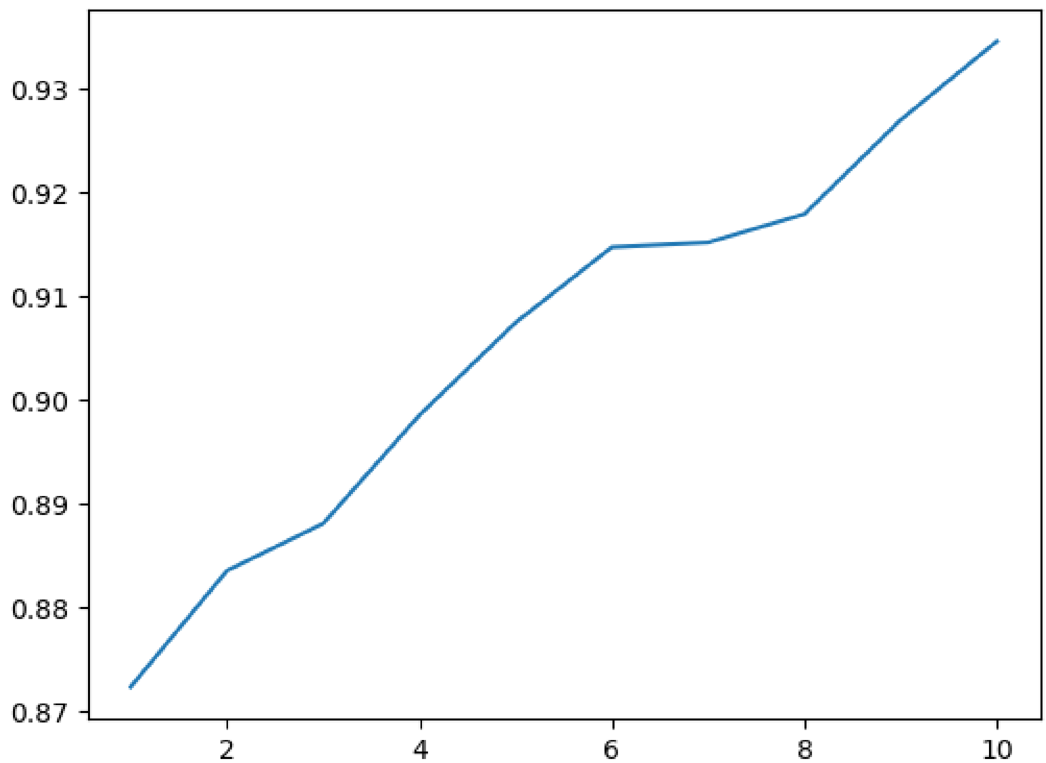
[ ]:
```
n_components = [3 for _ in range(10)]
covariance_type = ["full" for _ in range(10)]
max_iter = [0 for _ in range(10)]

model = DigitModel(n_components, covariance_type, max_iter)
model.fit(X,y)
pred_Y = model.predict(test_X)
print_conf(test_y, pred_Y, None)
print_conf(test_y, pred_Y, "true")
```


Confusion Matrix
Accuracy: 0.8845
True label
Predicted label



[ ]:
```
scores = []
for i in range(1,11):
    n_components = [i for _ in range(10)]
    covariance_type = ["full" for _ in range(10)]
    max_iter = [0 for _ in range(10)]

    model = DigitModel(n_components, covariance_type, max_iter)

    scores.append(np.average(cross_val_eval(model, X, y, 5, True)))
plt.plot(range(1,11), scores)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x17918b640>]
```

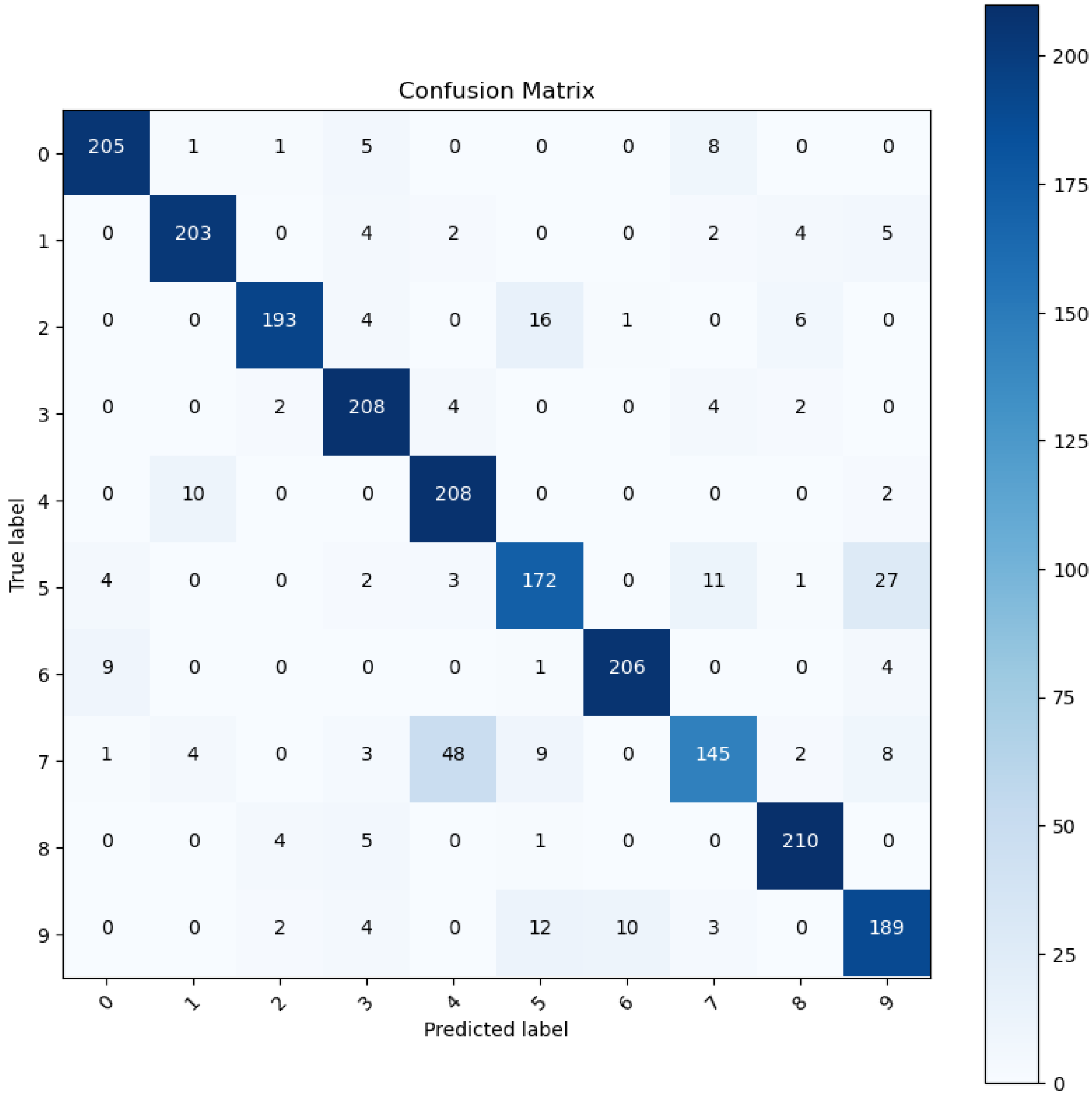


[ ]:
```
model.fit(X,y)
```

[ ]:

[ ]:

[ ]:



[ ]:
```
accuracy_score(test_y, pred_Y)
```

[ ]: 0.8813636363636363

[ ]:
```

```