ENCM 509 Final Project

# Speech Recognition using Gaussian Mixture Models

April 12th, 2023

Jessica Ritchie 30071655
Jade Fjestad 30064641

## Table of Contents

## Introduction and Objectives

Speaker recognition is the process of matching an individual to a voice sample [1]. In contrast, speech recognition is the challenge of interpreting what a person is saying in an audio clip [1]. Most home smart systems are concerned with speech recognition that reacts to user input. For example, Amazon's Alexa can respond to requests to play a certain song, recite your calendar details, or report the weather. In contrast, speaker recognition is mainly used for user authentication and verification. This can be used to replace the need for passwords on a phone or computer. Another application of speaker recognition is to incorporate it with speech recognition to improve the performance of these home smart systems. For example, speaker recognition would allow a member of a family to make a request, and the system would automatically recognize who was speaking and respond with personalized playlists, calendars, or emails.

Speaker recognition is also emerging in forensic science [1]. This technology can be used to identify people from their voices in recorded telephone calls for use in evidence [1]. Another application of speaker recognition is to improve live closed captioning. For example, in online conference calls, speaker recognition can identify how many different speakers are present and differentiate them to accurately label subtitles [2].

Speaker recognition is typically performed with a Gaussian Mixture Model (GMM) [3]. A GMM is an unsupervised clustering method that determines the probability that a sample belongs to each class [4]. GMM's are suitable for speaker recognition because they can handle vast amounts of numeric data with many features [4]. In addition, speech features are assumed to be normally distributed which permits simpler algorithms such as GMMs [5]. More recently, speaker recognition is moving towards neural networks as research in deep learning models has advanced. However, GMMs were long considered state of the art, and are sufficient for this project.

The objective of this report is to perform speaker recognition verification (am I whom I claim to be?) using a GMM. The GMM should accurately identify if a test sample is someone known to the database or not. The model will be trained on one individual to resemble a verification system for phone access. The GMM will be used to calculate test scores that will be compared to a threshold. Samples that score above the threshold would be accepted as the real user, and samples that score below the threshold would be considered imposters and denied access.

## Methods

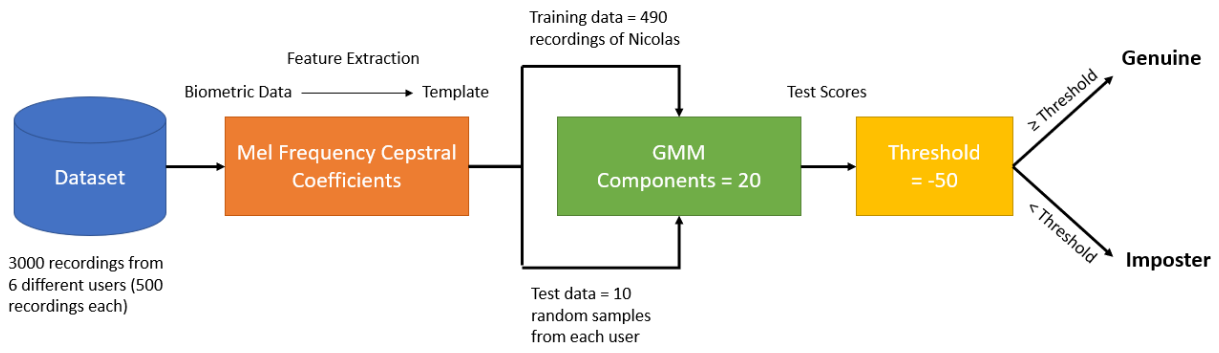

Dataset
3000 recordings from 6 different users (500 recordings each)
Feature Extraction
Biometric Data
Template
Mel Frequency Cepstral Coefficients
Training data = 490 recordings of Nicolas
Test data = 10 random samples from each user
GMM Components = 20
Test Scores
Threshold = -50
≥ Threshold
< Threshold
Genuine
Imposter

Figure 1. Project Flow Block Diagram

### Dataset

To train and test our system, we decided to use an alternative dataset to the one provided on D2L. The dataset we chose was from the Free Spoken Digit Dataset [6], which is a collection of recordings of spoken digits in wav files at 8kHz. This allowed us to use much more data than was available to us on D2L. In total, there are 3000 samples containing 500 recordings from 6 different speakers. The samples are of the different speakers saying the numbers 0 to 9. This provides us with a very clean dataset giving us very accurate results. However, this degree of clean data is unlikely to be a realistic real-world scenario, but works well for a simple project like this.

## Feature Extraction

One of the main challenges of speaker recognition is to extract features that capture the unique characteristics of each speaker's voice. In this project, we used MFCC (Mel Frequency Cepstral Coefficients) as the feature extraction method. MFCCs are based on the mel scale, which approximates the human perception of sound frequency (see Figure 2). To compute MFCCs, we first applied a Fourier transform to the audio signal, then mapped the frequency spectrum to the mel scale using triangular filters. Next, we divided the signal into overlapping frames and computed the logarithm of the filter bank energies. Finally, we applied a discrete cosine transform to reduce the dimensionality and decorrelate the features. We used the python_speech_features python library [7] to implement MFCC extraction. MFCCs are widely used in speaker recognition because they are robust to noise and capture the spectral envelope of speech.

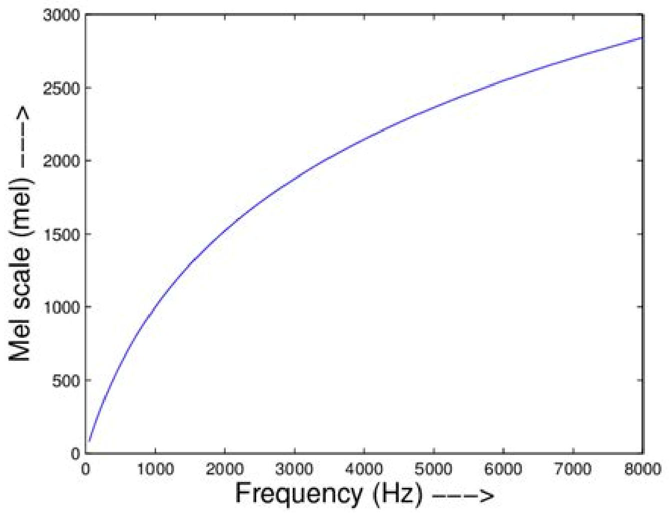


Figure 2. Mel Scale used to translate frequency to human perception

## Splitting Data

The GMM was trained on one speaker's data. The speaker chosen for our project was Nicolas. Before separating his data into training and test sets, an array containing the recording file names was randomly shuffled. This ensured the test and training sets contained a combination of spoken numbers. Next, the data was split. From 500 of Nicolas's recordings, the first 490 were used for training and the last 10 were used for testing. This test set represented the genuine user. For the imposter set, we selected 10 random recordings from each of the 5 remaining speakers for a total of 50 samples.

### Training Model

The GMM was trained on Nicolas's data using 20 Gaussian components. We built several models with varying numbers of Guassian components before settling on 20. Plotting each of these models output, we found 20 components provided good results while being quick to compute. The resulting model was used to calculate scores for testing data. Two arrays were used to store scores for both the genuine and imposter test sets.

## Results

The genuine and imposter distributions were estimated using the mean and standard deviation of their scores. These distributions and their raw scores are plotted in Figure 3.

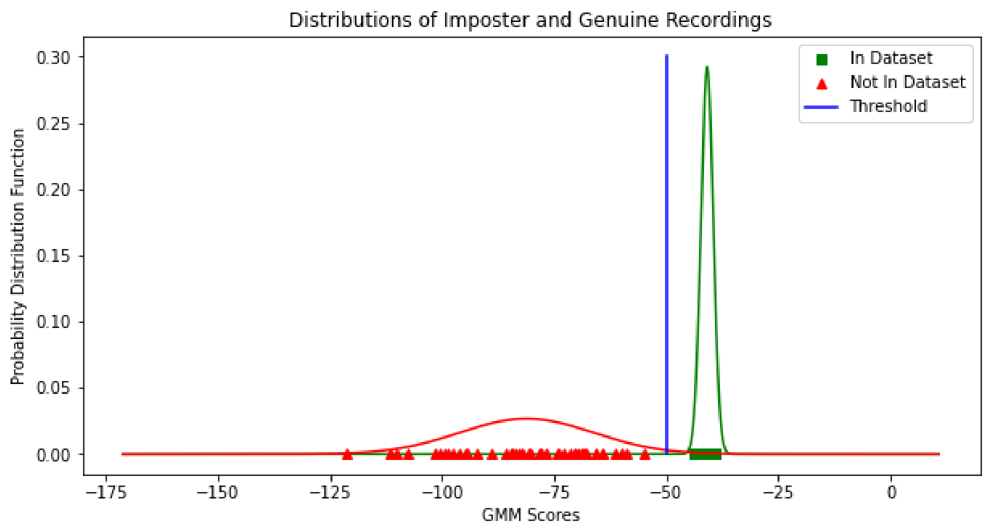


Figure 3. Scores From Genuine and Imposter Test Datasets

The genuine and imposter distributions are mutually exclusive as shown in the above image. Therefore, a threshold of -50 was chosen to categorize samples. The threshold is illustrated by the blue line. Scores above the threshold were labeled "In Dataset" (AKA: genuine) and scores below the threshold were labeled "Not in Dataset" (AKA: imposter). These labels were compared with the true identities to produce the confusion matrix and the performance metrics in Figure 4 and Table 1 respectively.

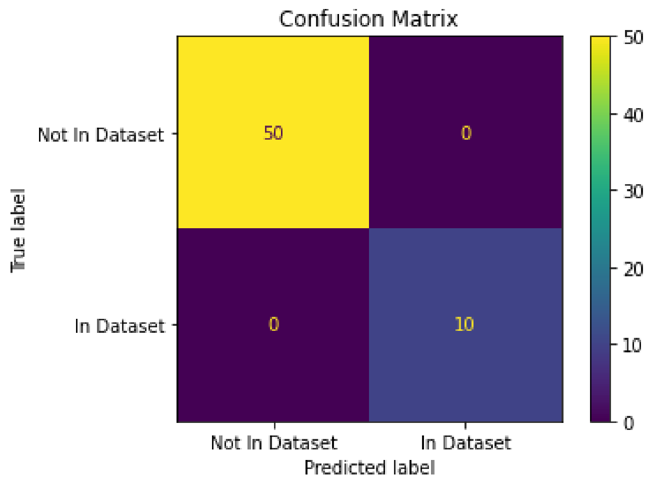


Figure 4. Results of Categorization

| Performance Metric | Value % |
|---|---|
| TRR | 100.0 |
| FAR | 0.0 |
| FRR | 0.0 |
| TAR | 100.0 |

Table 1. Performance Metrics

According to the above visuals, the GMM model had 100% accuracy. This is not surprising since the genuine and imposter distributions did not overlap. Therefore a threshold could be chosen that perfectly categorizes each sample.

## Future Work

This project is just a demonstration of the simple capabilities of speaker recognition. The next step would be to decide upon a use case for speaker recognition such as phone authentication. We could further develop this project as a speaker identification to assist in labeling who the speaker is within the audio. Likely the most helpful case is to be able to submit some samples of the speaker and then be able to further classify audio that contains multiple speakers, some of which the model is trained to detect.

Further investigation into algorithms and optimizations would have to be made if we were to pursue this project further.

## Conclusion

In conclusion, this project was an effective introduction to the capabilities of speaker recognition and the implementation of a Gaussian mixture model (GMM). The results were remarkable, achieving a 100% accuracy rate in speaker verification. However, the success of the project was largely attributed to the use of pristine data and a vast training set. The utilization of clean data provided a more precise representation of the speakers' unique characteristics, enabling the GMM to learn the features more effectively. Additionally, the large amount of training data provided the GMM with a diverse database of information, allowing for a more accurate identification of the test speakers.

Despite the success of the project, there are still limitations that need to be considered. One of the significant issues is the real-world implementation of speaker recognition. In actual scenarios, users would find it inconvenient to repeat the same phrase several times to develop a training set. It would be more practical to have a system that can quickly and accurately identify the user's voice with just a few samples of their speech. This challenge requires the development of a more robust and efficient model that can recognize the speaker's voice with minimal training data.

Another limitation is the binary classification nature of the project, which only compares a single person with everyone else. While this approach allowed us to set a single threshold that worked very well, in real-world applications, there would be multiple speakers, making it more complex to identify a particular person's voice. A more challenging task would be to identify an unknown speaker from a large pool of speakers, which would require more advanced classification techniques.

Despite these limitations, developing this project was a fascinating experience. The results achieved in this study show the potential of speaker recognition technology and its wide range of applications in various fields such as security, law enforcement, and communication systems. Future research can focus on addressing the limitations mentioned above and developing more advanced models that can accurately recognize speakers with limited training data. Overall, the project’s success opens up opportunities for further exploration in the field of speaker recognition.

## References

[1] S. S. Tirumala, S. R. Shahamiri, A. S. Garhwal, and R. Wang, “Speaker identification features extraction methods: A systematic review,” *Expert Syst. Appl.*, vol. 90, pp. 250–271, Dec. 2017, doi: 10.1016/j.eswa.2017.08.015.

[2] “Detect different speakers in an audio recording | Cloud Speech-to-Text Documentation,” *Google Cloud*. https://cloud.google.com/speech-to-text/docs/multiple-voices (accessed Apr. 05, 2023).

[3] Z. Bai and X.-L. Zhang, “Speaker Recognition Based on Deep Learning: An Overview.” arXiv, Apr. 03, 2021. Accessed: Apr. 05, 2023. [Online]. Available: http://arxiv.org/abs/2012.00931

[4] C. Ellis, “When to use gaussian mixture models,” *Crunching the Data*, Jun. 07, 2022. https://crunchingthedata.com/when-to-use-gaussian-mixture-models/ (accessed Apr. 05, 2023).

[5] A. Aroudi, H. Veisi, H. Sameti, and Z. Mafakheri, “Speech signal modeling using multivariate distributions,” *EURASIP J. Audio Speech Music Process.*, vol. 2015, no. 1, p. 35, Dec. 2015, doi: 10.1186/s13636-015-0078-1.

[6] “Audio MNIST,” Kaggle, Sep. 18, 2020. https://www.kaggle.com/datasets/alanchn31/free-spoken-digits

[7] “Welcome to python_speech_features’s documentation! — python_speech_features 0.1.0 documentation.” https://python-speech-features.readthedocs.io/en/latest/

[8] H. Sahai, “MFCC (Mel Frequency Cepstral Coefficients) for Audio format,” OpenGenus IQ: Computing Expertise & Legacy, [Online]. Available: https://iq.opengenus.org/mfcc-audio/

# gmm-kaggle-data

April 12, 2023

## 0.1 Project Tasks

[1]:
```
#pip install python_speech_features==0.4
```

[2]:
```
# data from
# https://www.kaggle.com/datasets/alanchn31/free-spoken-digits
```

[3]:
```
# Packages
import os
import re
import random
import numpy as np
from python_speech_features import mfcc
from sklearn.mixture import GaussianMixture
from scipy.io import wavfile
import matplotlib.pyplot as plt
from scipy.stats import norm
```

[4]:
```
# train the GMM on the concatenated MFCC features
random.seed(0)

person_num = 3  # Index number of person to train on
test_samples = 10  # Number of samples to test

paths = os.listdir('../data/recordings/')  # Filename of every recording
random.shuffle(paths) # Shuffle data so test and train contain multiple␣
 ↪different words

# extracts all the people names in the dataset
people = list(set([" ".join(re.findall("[a-zA-Z]+", "".join(x.split('.')[0:
 ↪-1]))) for x in paths]))
people.sort()

# add relative path
paths = ['../data/recordings/'+ x for x in paths]

print(f'Training on {people[person_num]}')
print(people)
```

```
Training on nicolas
['george', 'jackson', 'lucas', 'nicolas', 'theo', 'yweweler']
```

[5]:
```
# Testing with alternative larger and more interesting dataset

# directory containing the training data files
one_person_paths = [x for x in paths if people[person_num] in x]
train_paths = one_person_paths[0:-test_samples]  # Keep the first 490 samples␣
 ↪for training
test_paths = one_person_paths[-test_samples:]  # Keep the last 10 samples for␣
 ↪testing

print(len(train_paths))
print(len(test_paths))
#print(test_paths)
```

```
490
10
```

[6]:
```
# create a GMM with n components
n = 20
gmm = GaussianMixture(n_components=n)

# loop over each audio file in the data directory to extract features
mfcc_features = None  # Create empty dataframe
for file_path in train_paths:
    sample_rate, audio_data = wavfile.read(file_path)  # extract data and␣
 ↪sample rate for each file

    # compute MFCC features for the audio data
    mfcc_data = mfcc(audio_data, sample_rate)

    # concatenate the MFCC features into a single numpy array
    if mfcc_features is None:
        mfcc_features = mfcc_data
    else:
        mfcc_features = np.concatenate((mfcc_features, mfcc_data), axis=0)
```

[7]:
```
print(mfcc_features[0])
```

```
[ 18.28724711   0.10534504 -20.11034637 -46.68443102  -5.85794963
   3.57784646  -6.55325388 -15.80200711 -14.58062069  -3.6266417
  -8.03280999 -12.3525111    4.68319853]
```

[8]:
```
# train the GMM on the concatenated MFCC features
gmm.fit(mfcc_features)
```

[8]: 
```
GaussianMixture(n_components=20)
```

[9]: 
```
# Calculate scores for Imposter recordings

imposter_scores = []

for person in people:
    if person != people[person_num]:  # If person was not trained on get a␣
 ↪random sample of 10 recordings for testing
        person_paths = random.sample([x for x in paths if person in x],␣
 ↪test_samples)
    else:
        continue

    individual_scores = []
    for path in person_paths:
        sample_rate, test_data = wavfile.read(path)  # Extract data
        mfcc_data = mfcc(test_data, sample_rate, numcep=13)  # Feature␣
 ↪extraction
        score = gmm.score(mfcc_data) # Calulate score
        individual_scores.append(score)   # Get classification score for each␣
 ↪recording

    imposter_scores.extend(individual_scores)

print(len(imposter_scores))
```

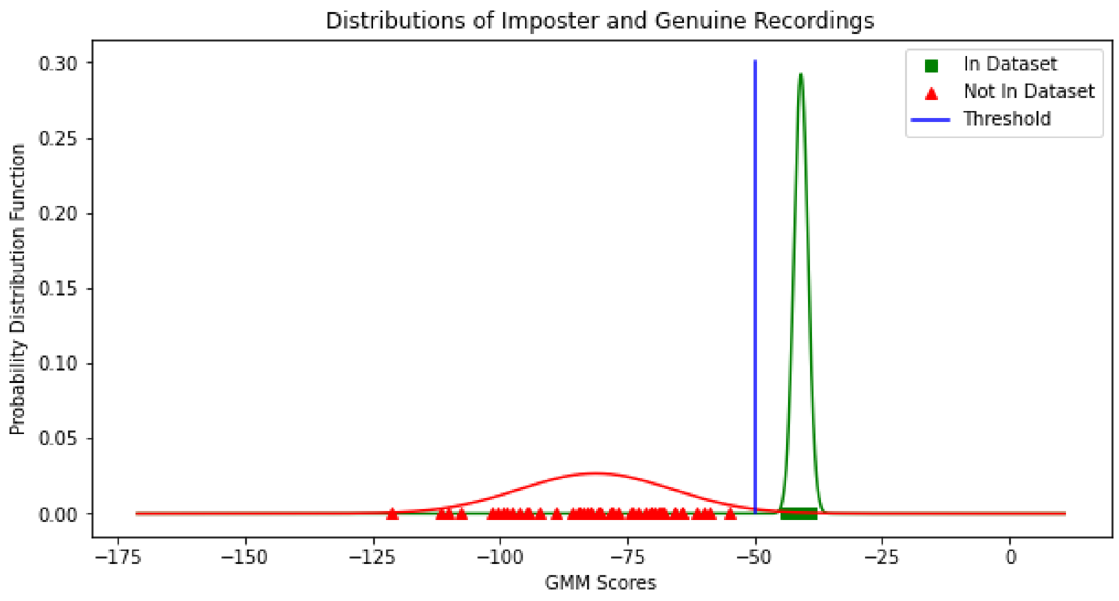
```
50
```

[10]: 
```
# Calculate scores for genuine recordings

genuine_scores = []

for path in test_paths:
    sample_rate, test_data = wavfile.read(path)  # Extract data
    mfcc_data = mfcc(test_data, sample_rate, numcep=13)  # Feature extraction
    score = gmm.score(mfcc_data) # Calulate score
    genuine_scores.append(score)   # Get classification score for each recording

print(len(genuine_scores))
```

```
10
```

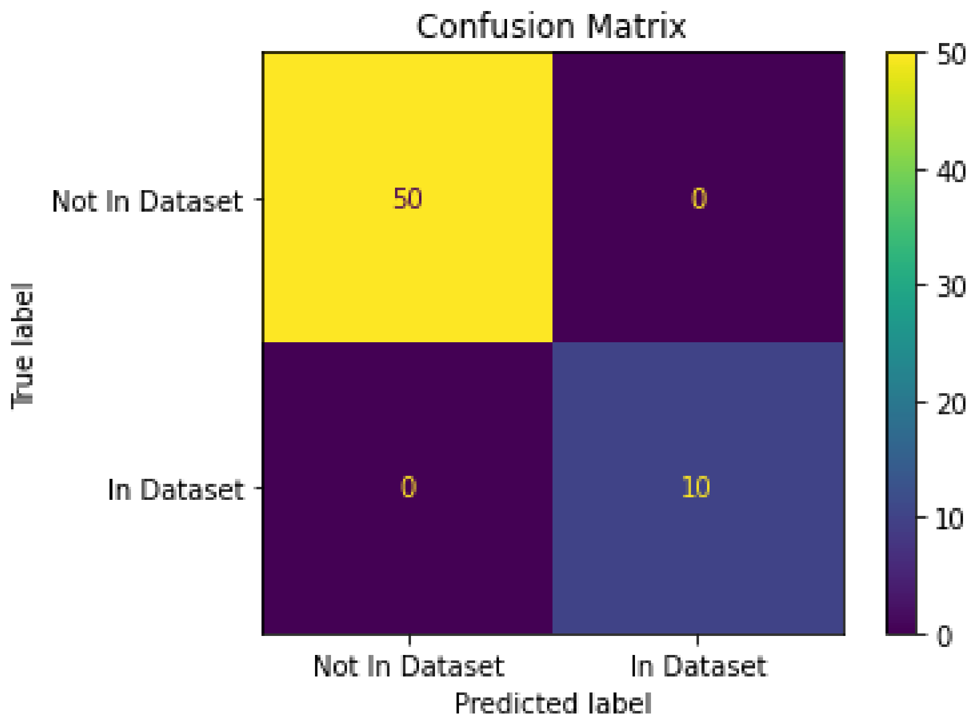
[11]: 
```
# Plot genuine and imposter distributions to determine a threshold

person_scores = list(genuine_scores)
person_scores.extend(imposter_scores)
```

```
lowest_value = min(person_scores)
highest_value = max(person_scores)

x = np.arange(lowest_value-50, highest_value+50, 0.01)

genuine = norm.pdf(x, loc=np.mean(genuine_scores), scale=np.std(genuine_scores))
imposter = norm.pdf(x, loc=np.mean(imposter_scores), scale=np.
 ↪std(imposter_scores))

threshold = -50

plt.figure(figsize=(10,5))
plt.plot(x, genuine, 'g')
plt.plot(genuine_scores, np.zeros(len(genuine_scores)), 'gs', lw=2, label='In␣
 ↪Dataset')
plt.plot(x, imposter, 'r')
plt.plot(imposter_scores, np.zeros(len(imposter_scores)), 'r^', lw=2,␣
 ↪label='Not In Dataset')
plt.plot([threshold]*20, np.linspace(0, 0.3, 20), 'b', label='Threshold')
plt.xlabel("GMM Scores")
plt.ylabel("Probability Distribution Function")
plt.legend();
plt.title('Distributions of Imposter and Genuine Recordings');

plt.show()
```



[12]:
```python
# Choose threshold and make confusion matrix

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

TA = 0
FA = 0
TR = 0
FR = 0

for score in genuine_scores:
    if score < threshold:
        FR = FR + 1
    else:
        TA = TA + 1

for score in imposter_scores:
    if score < threshold:
        TR = TR + 1
    else:
        FA = FA + 1

Total = len(genuine_scores) + len(imposter_scores)

print("True Rejection Rate :" + str((TR/(TR+FA))*100))
print("False Accecptance Rate :" + str((FA/(FA+TR))*100))
print("False Rejection Rate :" + str((FR/(FR+TA))*100))
print("True Accecptance Rate :" + str((TA/(TA+FR))*100))

cm = np.array([[TR, FA], [FR, TA]])
labels = np.array(["Not In Dataset", "In Dataset"])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot()
plt.title('Confusion Matrix');
plt.show()
```

```
True Rejection Rate :100.0
False Accecptance Rate :0.0
False Rejection Rate :0.0
True Accecptance Rate :100.0
```


Confusion Matrix
Not In Dataset
In Dataset
True label
50
0
0
10
Not In Dataset
In Dataset
Predicted label
50
40
30
20
10
0