Susan Radu

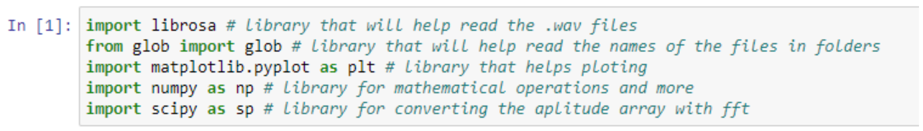
Machine Learning Project Documentation

Surgical mask detection

The program is given 8000 audio.wav train files of a recorded voice, each labeled 0 for no mask and 1 for mask, 1000 validation files, and 3000 test files.

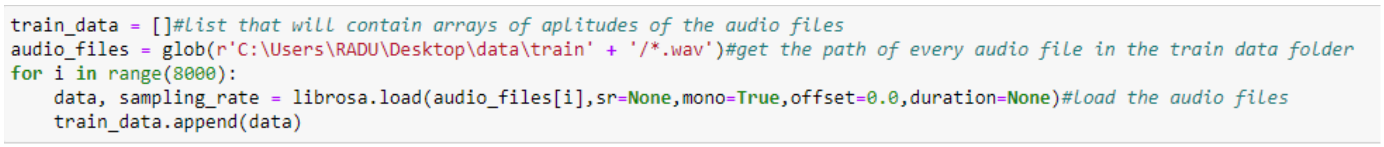
We must build a model that will learn to recognize which recorded voice has a mask and which does not, using the train and validation files.

First import the needed libraries:

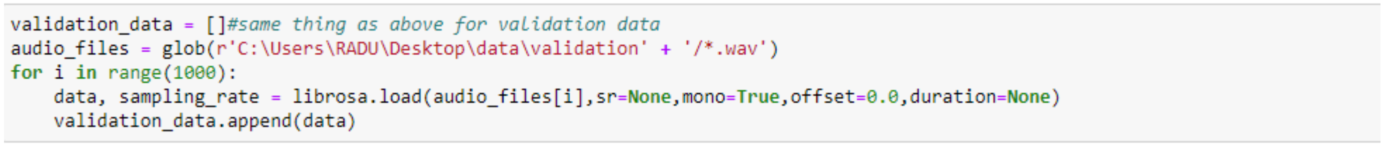


```
In [1]: import librosa # library that will help read the .wav files
        from glob import glob # library that will help read the names of the files in folders
        import matplotlib.pyplot as plt # library that helps ploting
        import numpy as np # library for mathematical operations and more
        import scipy as sp # library for converting the aplitude array with fft
```

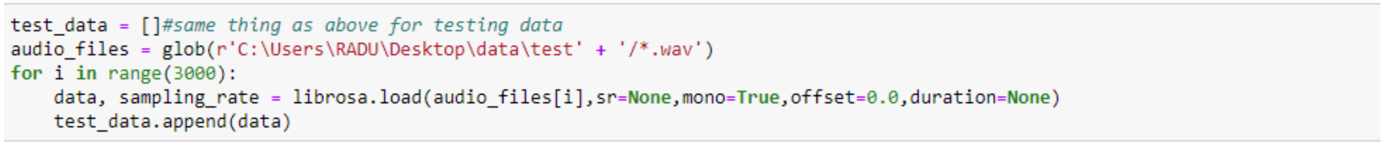
Then we load the audio files:



```
train_data = []#list that will contain arrays of aplitudes of the audio files
audio_files = glob(r'C:\Users\RADU\Desktop\data\train' + '/*.wav')#get the path of every audio file in the train data folder
for i in range(8000):
    data, sampling_rate = librosa.load(audio_files[i],sr=None,mono=True,offset=0.0,duration=None)#load the audio files
    train_data.append(data)
```

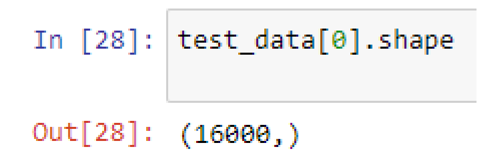


```
validation_data = []#same thing as above for validation data
audio_files = glob(r'C:\Users\RADU\Desktop\data\validation' + '/*.wav')
for i in range(1000):
    data, sampling_rate = librosa.load(audio_files[i],sr=None,mono=True,offset=0.0,duration=None)
    validation_data.append(data)
```

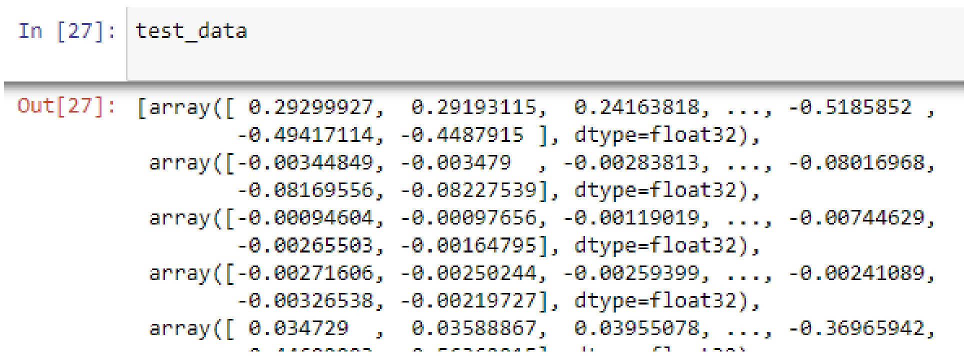


```
test_data = []#same thing as above for testing data
audio_files = glob(r'C:\Users\RADU\Desktop\data\test' + '/*.wav')
for i in range(3000):
    data, sampling_rate = librosa.load(audio_files[i],sr=None,mono=True,offset=0.0,duration=None)
    test_data.append(data)
```

Now we have 3 lists of arrays, each array consists of 16000 numbers that are the amplitude of each audio (16000 because the audio files all had a sampling rate of 16000 and one second duration):



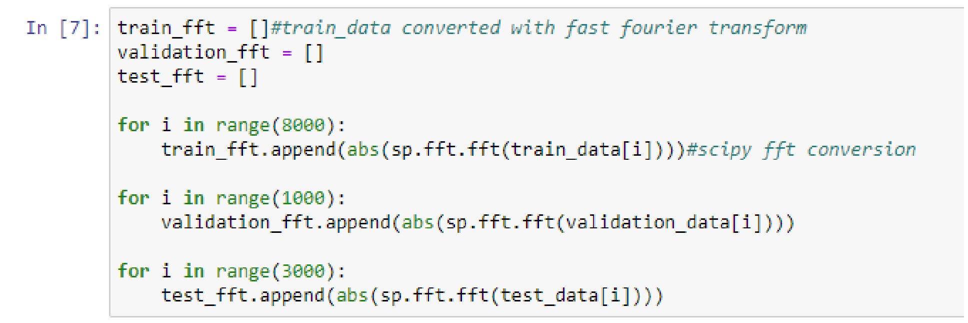
```
In [28]: test_data[0].shape

Out[28]: (16000,)
```



```
In [27]: test_data

Out[27]: [array([ 0.29299927,  0.29193115,  0.24163818, ..., -0.5185852 ,
                 -0.49417114, -0.4487915 ], dtype=float32),
          array([-0.00344849, -0.003479  , -0.00283813, ..., -0.08016968,
                 -0.08169556, -0.08227539], dtype=float32),
          array([-0.00094604, -0.00097656, -0.00119019, ..., -0.00744629,
                 -0.00265503, -0.00164795], dtype=float32),
          array([-0.00271606, -0.00250244, -0.00259399, ..., -0.00241089,
                 -0.00326538, -0.00219727], dtype=float32),
          array([ 0.034729  ,  0.03588867,  0.03955078, ..., -0.36965942,
```

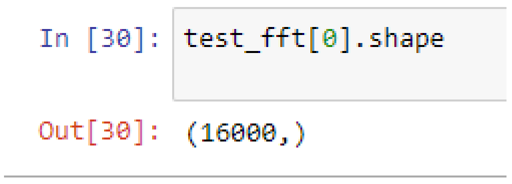
But if we train our model using the raw amplitude it won’t learn very effectively, so we need to convert those arrays to something more meaningful for the model.

We choose the fast fourier transform algorithm (FFT), which converts the audio data that we had from amplitudes to frequencies (from time-domain to frequency-domain):

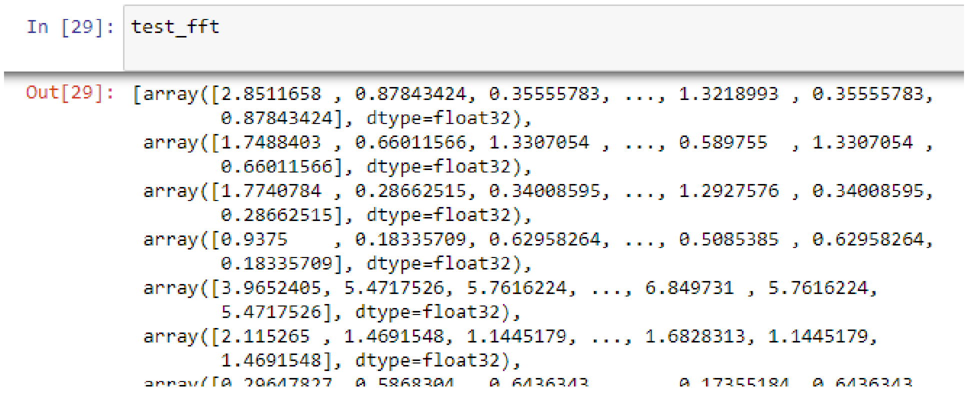


```
In [7]: train_fft = []#train_data converted with fast fourier transform
        validation_fft = []
        test_fft = []

        for i in range(8000):
            train_fft.append(abs(sp.fft.fft(train_data[i])))#scipy fft conversion

        for i in range(1000):
            validation_fft.append(abs(sp.fft.fft(validation_data[i])))

        for i in range(3000):
            test_fft.append(abs(sp.fft.fft(test_data[i])))
```

Our arrays should look like this now:



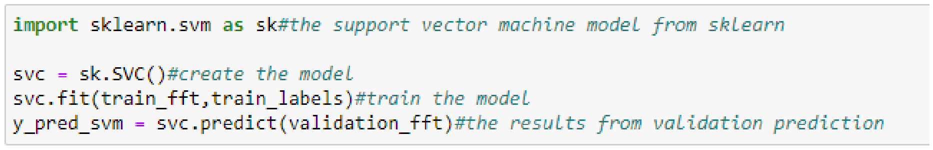
```
In [30]: test_fft[0].shape

Out[30]: (16000,)
```



```
In [29]: test_fft

Out[29]: [array([2.8511658 , 0.87843424, 0.35555783, ..., 1.3218993 , 0.35555783,
                0.87843424], dtype=float32),
          array([1.7488403 , 0.66011566, 1.3307054 , ..., 0.589755  , 1.3307054 ,
                0.66011566], dtype=float32),
          array([1.7740784 , 0.28662515, 0.34008595, ..., 1.2927576 , 0.34008595,
                0.28662515], dtype=float32),
          array([0.9375    , 0.18335709, 0.62958264, ..., 0.5085385 , 0.62958264,
                0.18335709], dtype=float32),
          array([3.9652405, 5.4717526, 5.7616224, ..., 6.849731 , 5.7616224,
                5.4717526], dtype=float32),
          array([2.115265 , 1.4691548, 1.1445179, ..., 1.6828313, 1.1445179,
                1.4691548], dtype=float32),
          array([0.29647827, 0.5868304 , 0.6436343 , ..., 0.17355184, 0.6436343 ,
```

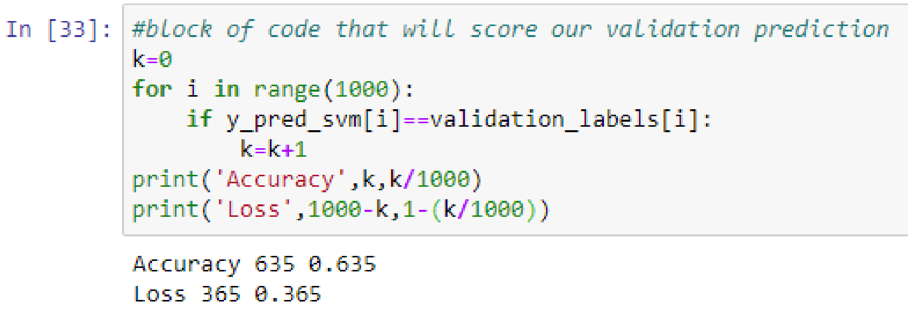
Now that we have the data ready, we can start making our model

The first model that we will use is Support vector machine (SVM):



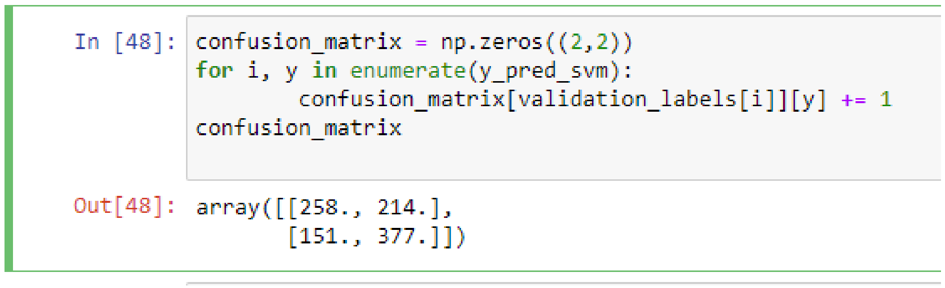
```
In [24]: import sklearn.svm as sk#the support vector machine model from sklearn

         svc = sk.SVC()#create the model
         svc.fit(train_fft,train_labels)#train the model
         y_pred_svm = svc.predict(validation_fft)#the results from validation prediction
```

And the accuracy and loss for the validation is:



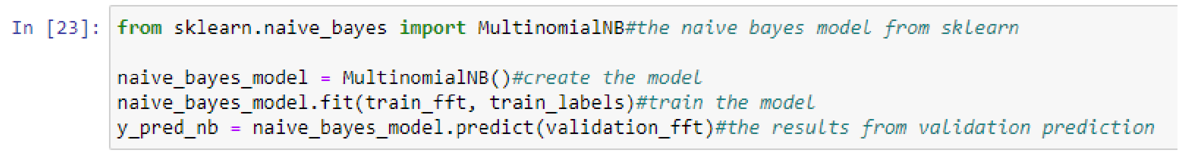
```
In [33]: #block of code that will score our validation prediction
         k=0
         for i in range(1000):
             if y_pred_svm[i]==validation_labels[i]:
                 k=k+1
         print('Accuracy',k,k/1000)
         print('Loss',1000-k,1-(k/1000))

         Accuracy 635 0.635
         Loss 365 0.365
```

And the confusion matrix:



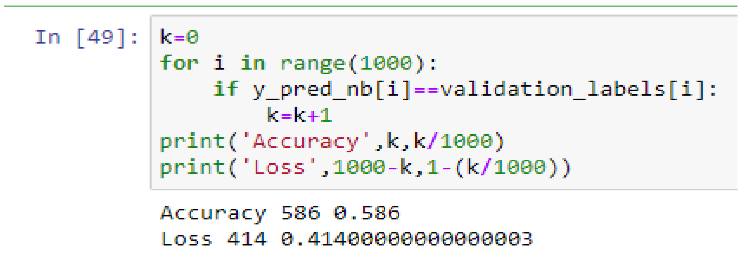
```
In [48]: confusion_matrix = np.zeros((2,2))
         for i, y in enumerate(y_pred_svm):
                 confusion_matrix[validation_labels[i]][y] += 1
         confusion_matrix

Out[48]: array([[258., 214.],
                [151., 377.]])
```

We will now take a look at another model, Naïve Bayes:



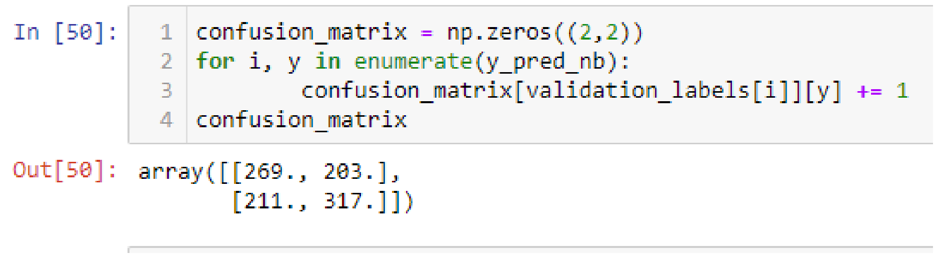
```
In [23]: from sklearn.naive_bayes import MultinomialNB#the naive bayes model from sklearn

         naive_bayes_model = MultinomialNB()#create the model
         naive_bayes_model.fit(train_fft, train_labels)#train the model
         y_pred_nb = naive_bayes_model.predict(validation_fft)#the results from validation prediction
```

And the accuracy and loss for the validation is:



```
In [49]: k=0
         for i in range(1000):
             if y_pred_nb[i]==validation_labels[i]:
                 k=k+1
         print('Accuracy',k,k/1000)
         print('Loss',1000-k,1-(k/1000))

         Accuracy 586 0.586
         Loss 414 0.41400000000000003
```

And the confusion matrix:

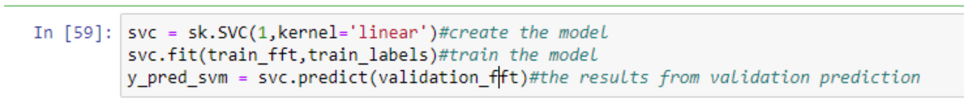


```
In [50]: confusion_matrix = np.zeros((2,2))
         for i, y in enumerate(y_pred_nb):
                 confusion_matrix[validation_labels[i]][y] += 1
         confusion_matrix

Out[50]: array([[269., 203.],
                [211., 317.]])
```

So, with the default options, the SVM model seems to be performing better on the validation test.

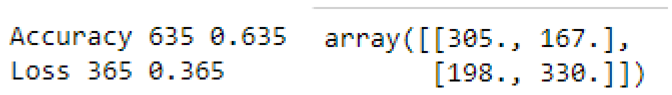
Let’s try to change the hyperparameters

Changed the kernel to a linear one:



```
In [59]: svc = sk.SVC(1,kernel='linear')#create the model
         svc.fit(train_fft,train_labels)#train the model
         y_pred_svm = svc.predict(validation_fft)#the results from validation prediction
```

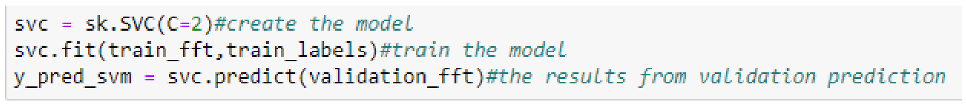
And the result:



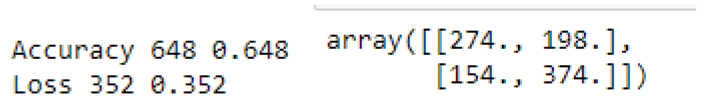
```
Accuracy 635 0.635   array([[305., 167.],
Loss 365 0.365              [198., 330.]])
```

We can see that, while the accuracy is the same, the confusion matrix is different

Let’s try the default kernel with C=2:

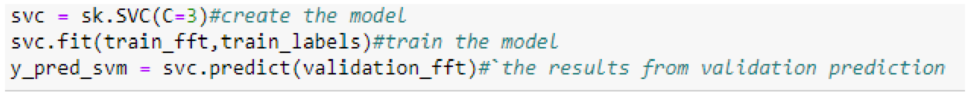


```
svc = sk.SVC(C=2)#create the model
svc.fit(train_fft,train_labels)#train the model
y_pred_svm = svc.predict(validation_fft)#the results from validation prediction
```

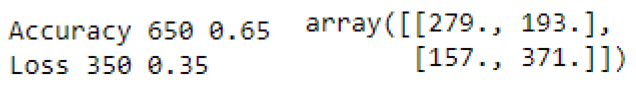


```
Accuracy 648 0.648     array([[274., 198.],
Loss 352 0.352                [154., 374.]])
```

The accuracy seems to be improving, let's try C=3

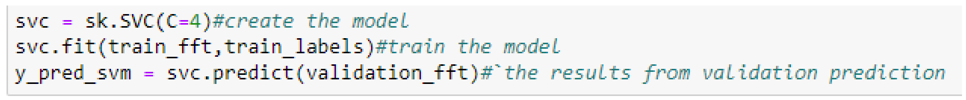


```
svc = sk.SVC(C=3)#create the model
svc.fit(train_fft,train_labels)#train the model
y_pred_svm = svc.predict(validation_fft)#`the results from validation prediction
```

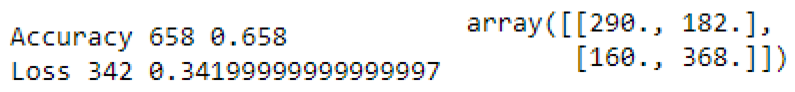


```
Accuracy 650 0.65     array([[279., 193.],
Loss 350 0.35                [157., 371.]])
```

Still improving, C=4:

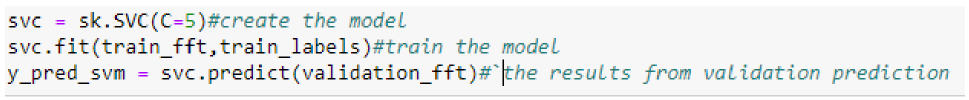


```
svc = sk.SVC(C=4)#create the model
svc.fit(train_fft,train_labels)#train the model
y_pred_svm = svc.predict(validation_fft)#`the results from validation prediction
```

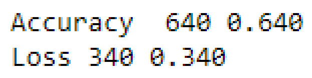


```
Accuracy 658 0.658                    array([[290., 182.],
Loss 342 0.34199999999999997                 [160., 368.]])
```

C=5



```
svc = sk.SVC(C=5)#create the model
svc.fit(train_fft,train_labels)#train the model
y_pred_svm = svc.predict(validation_fft)#`the results from validation prediction
```



```
Accuracy  640 0.640
Loss 340 0.340
```

Here we can see that the accuracy is lowering.

So, for our final model we will chose the SVM with C=4