CSI Coursework Report

Part 1

This first section is a report on 4 statistical tests run against data from an experiment involving phone call conversations between men and women in different combinations. The calls are split up into sections representing different actions such as “laughter” or “back channel”.

Laughter and Filler Frequency

Hypothesis

Two tests were conducted to try and address the query of whether female subjects perform certain actions more frequently than males during the experimental phone conversations that the data is sourced from. The first statistical test was for if laughter is used more frequently by females (1), and the second was for if filler is used more frequently (2).

(1) Laughter

Research hypothesis: Female subjects laugh more frequently than male ones by a statistically significant amount.

Null hypothesis: Female subjects do not laugh more frequently that male ones by a statistically significant amount.

(2) Filler

Research hypothesis: Female subjects use fillers more frequently than male ones by a statistically significant amount.

Null hypothesis: Female subjects do not use fillers more frequently that male ones by a statistically significant amount.

Data

Test	Action counts	
	Male	Female
1 (laughter)	519	748
2 (filler)	1935	1602

Methodology

A Chi-Square test was used here to try to determine whether the variables gender and laughter frequency were related or independent amongst the test population. This is a one-tailed test, as we are only considering one direction - females having a higher frequency than males.

In order to calculate this statistic, first we needed to create a contingency table of observed values, and also work out what the expected values would be for these actions for both genders. First, the total talk durations for both females and males were calculated by summing the durations of all data for each gender in the set excluding “silence” instances. This was then used to calculate the fraction of total talk time spoken by females $p_f = 0.45$, and of males $p_m = 0.55$ (to 2 decimal places.).

In both the laughter and filler case, the expected frequencies could now be calculated by multiplying the total frequency of the action by the p value for each gender, like so:

$$Frequency_{(gender)} = N \times p_{gender}$$

The observations and expectations were then fed into a function to calculate the Chi Squared value which sums the comparisons of each observation with its related expectation like so:

$$\chi^2 = \sum_{k=1}^{L} \frac{(O_k - E_k)^2}{E_k},$$

Where O_k is an observation, E_k is it’s expected value, and L is the number of observations (in this case, 2 for both experiments). Also, we can take a degrees of freedom value here equal to L-1, which we can use to look up a sampling distribution for Chi Square to gain a p value from (in this case, the degrees of freedom will be 1, so the critical value for χ^2 in order to achieve a confidence better than 0.05 is 3.85).

Results

Test	Observations		Expectations (to 1 d.p.)	
	Male	Female	Male	Female
1 (laughter)	519	748	696.8	570.2
2 (filler)	1935	1602	1945.2	1591.8

It is fairly obvious to see by looking at the observed values compared with the expected ones, that there is a marked difference in frequencies for laughter, but little difference for filler.

Test	χ^2 value (3 d.p.)	p value (3 s.f.)	Null hypo. decision
1 (laughter)	100.800	1.02×10^{-23}	Reject
2 (filler)	0.119	0.731	Retain

The results of the Chi Square solidify these suspicions. The p value for laughter is extremely low, and well within the confidence boundary of $\alpha = 0.05$, giving a confidence level of above 99% for rejecting the null hypothesis, so we can say that there is probably a statistically significant increase in laughter frequency for females compared with males. For filler however, our p value is nowhere near the required level in order to reject the null hypo. In this case, we must retain the null hypothesis, as there is not a statistically significant difference for this data.

Laughter and Filler Length Difference

Hypothesis

Again, two further tests were conducted; this time to try and address the query of whether there is any statistically significant difference in the duration of each gender's talking actions. The first statistical test was for if there is a laughter length difference (3), and the second was for if there is a filler length difference (4).

(3) Laughter

Research hypothesis: There is a statistically significant laughter length difference between male and female subjects in either direction.

Null hypothesis: There is not a statistically significant laughter length difference between male and female subjects in either direction.

(4) Filler

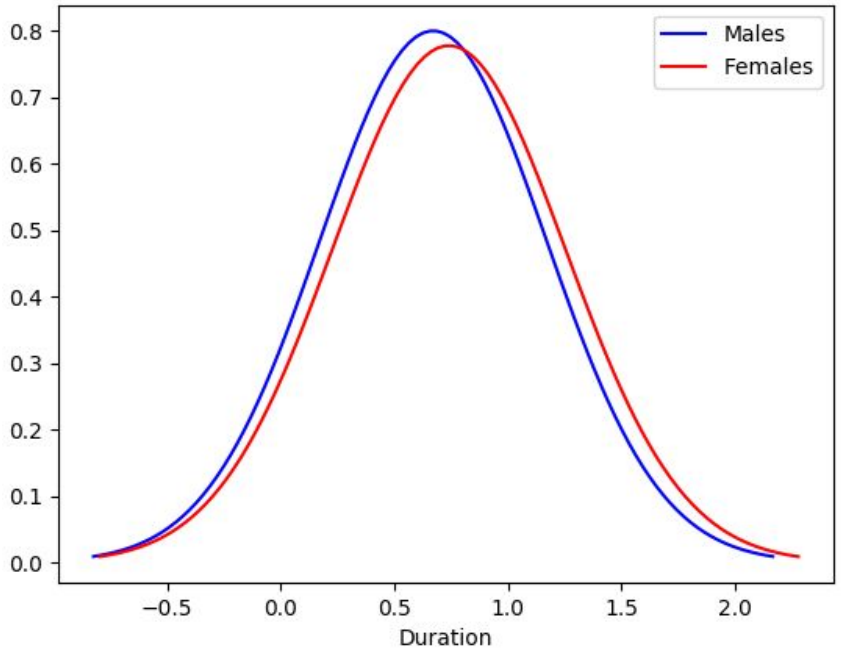
Research hypothesis: There is a statistically significant filler length difference between male and female subjects in either direction.

Null hypothesis: There is not a statistically significant filler length difference between male and female subjects in either direction.

Data

Presented below are comparisons of the normal distributions of laugh/filler duration per gender.

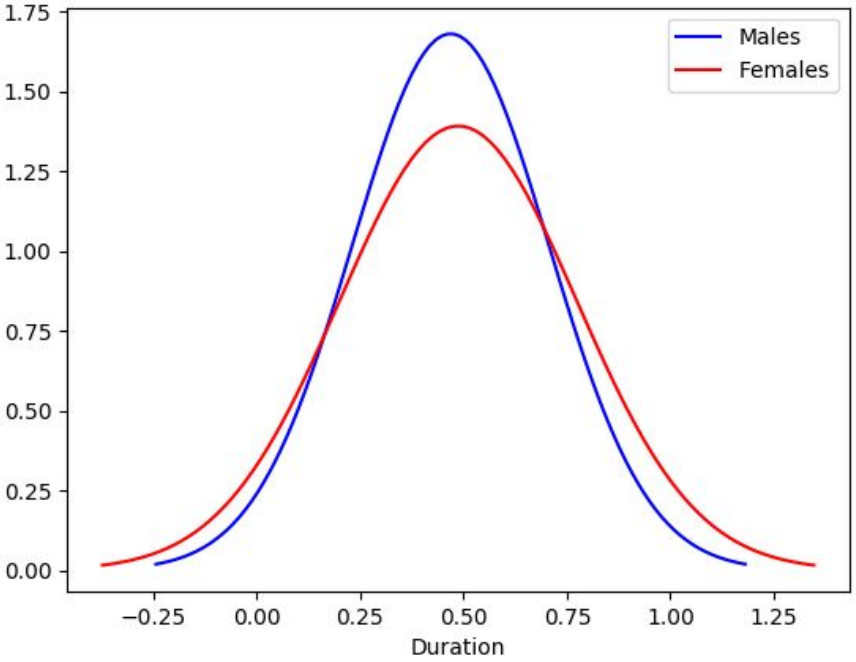
Test 3 (Laughter)



Males: $\mu = 0.672$, $\sigma^2 = 0.249$

Females: $\mu = 0.741$, $\sigma^2 = 0.263$

Test 4 (Filler)



Males: $\mu = 0.469$, $\sigma^2 = 0.056$

Females: $\mu = 0.487$, $\sigma^2 = 0.082$

Methodology

A t-test was used for these statistical analyses because they are both two-tailed in nature, and also we are comparing samples from 2 demographics which can be represented by a gaussian, which can therefore be compared with a t-test. This is because the research hypothesis does not imply a specific direction, we are only looking for a difference in the durations of these actions for each gender.

I have provided both my own implementation of the t-test function, and also used the scipy.stats.ttest_ind() function to check against the result.

Results

Test	t value (3 d.p.)	p value (3 s.f.)	Null hypo. decision
1 (laughter)	2.407	0.0162	Reject
2 (filler)	2.054	0.0401	Reject

From these results we can relatively safely reject the null hypothesis for both tests. In the case of laughter with about 98% confidence, and for filler 96% confidence. We would conclude in this case that females on average have longer lengths of both their laughs and their fillers.

However, I am not sure how sensible these results are, as the original analysis of the data shows that the gendered distributions for both tests were quite similar in both cases. Therefore I would expect that the t-test would judge the distributions to not be different to statistically significant extent, whereas they have both been seen to be different enough in these results for us to reject the null hypothesis. I am therefore quite skeptical of these results.

Part 2

Problem Introduction

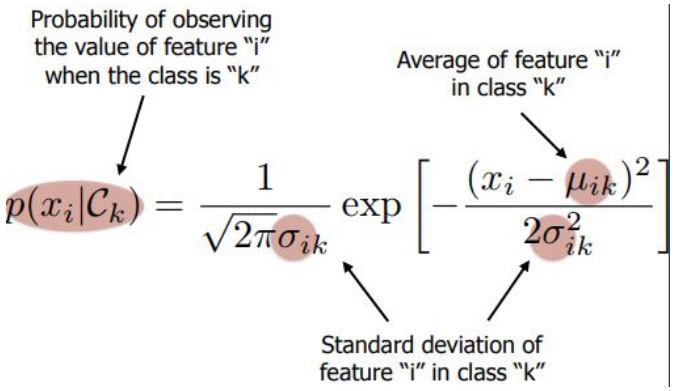
Facial expression analysis can be a difficult task, as it is hard to describe facial behaviour as interpreted by human observation in a way that can be used in the context of machine learning.

A way to solve this issue is to employ the facial movement measurement techniques created by Ekman and Friesen, grouped under the name of the “Facial Action Code”. This method maps visible facial movements made by specific muscular actions to “Action Units”, with a value representing the degree to which that AU is being used in the current expression. This allows us to record facial expressions as feature vectors, where the features are values corresponding to each of the AUs.

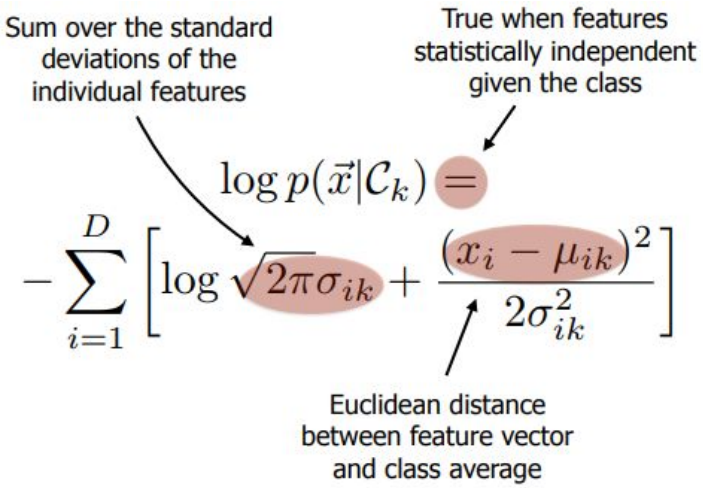
Gaussian Discriminant Functions

In this task, a classifier was designed based on the concepts of Gaussian Discriminant Functions.

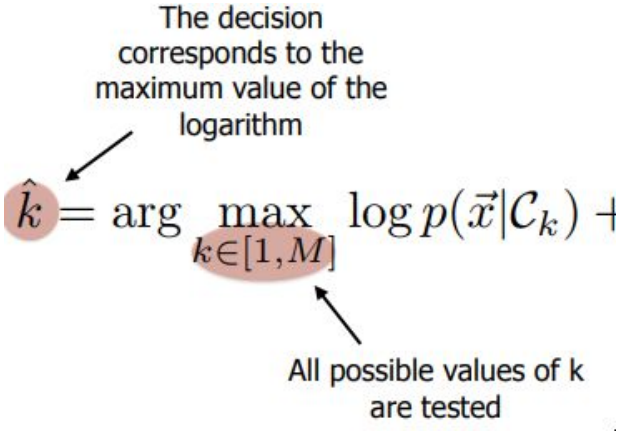
This method utilises the following formula derived from Bayes rule for calculating the likelihood of observing a particular value of a feature x_i when it’s class is C_k using the mean μ_{ik} and standard deviation σ_{ik}, across all values of of that feature for that class in the training set.



We are able to use this formula by assuming that all of our features for this task are independent of one another. We can modify this formula to give us the log likelihood of observing values across a whole vector of features like so:



We can compare the log likelihood achieved for each available class (in this case “smile” or “frown”), and choose the one which maximises the likelihood value and is therefore most likely to be the correct classification for the given vector. This next formula mathematically formalises this decision:



Methodology

In this scenario, as stated above, it can be assumed that the prior probabilities are essentially equal, as there are only 2 classes, and we have no reason to believe that a smile is more likely than a frown or vice versa. We were also able to treat the features as independent of one another, as we were using the FAC’s Action Units approach, allowing us to make use of the GDF method.

The classifier was trained using the K-Folds approach. This involves splitting the data into a number of sections, and then removing one of these to use as testing data, whilst the remaining sections act as training data. This is repeated k times, and the error rate for each attempt is compared in order to choose the best (the one with the lowest error rate, and therefore best classification rate). The gaussian discriminant function and vectors of feature means and variances from this best attempt then become the final “model”.

In order to improve the values gained from training, the feature vectors of each class were distributed evenly across all folds.

The error rate is calculated using a Zero-One loss function:

$$\pi(\beta_i|\mathcal{C}_j) = \left\{ \begin{matrix} 0 \; i = j \\ 1 \; i \neq j. \end{matrix} \right. \qquad i, j = 1, \ldots, M \Big\}.$$

Each mis-classified vector counts as an error, and the error rate is calculated as the number of errors divided by the total number of attempted classifications.

After training, the selected model was then run over the whole dataset and the error rate recorded as a final metric for performance of this method.

Results

Fold	Error rate
0	7.692%
1	0.0%
2	7.692%
3	0.0%

Here we can see the performance of each fold in terms of error rate. Fold 1 was chosen as the final “model” to use, and it achieved an error rate of 0.0% when tested afterwards on the whole data set. It’s fairly safe to say then that this is a successful model for this data.

This shows that even with a small data sample like the 52 feature vectors used in this scenario, facial expressions can be predicted to a fairly high accuracy using a GDF method. It would be better to train and test this on a larger data set however, before this method is declared entirely reliable.

Appendix

Part 1 - Code

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

data = pd.read_csv("data-part-1.csv", header=None, names=["Fnn", "Topic", "Event", "Start time", "End
time"])

#clean data

cleaned_data = pd.DataFrame(columns=["Fnn", "Action", "Role", "Gender", "Duration"]);
cleaned_data_list = []
for index, row in data.iterrows():
        # remove any extra events after the first
        event_contents = row['Event'].split(" ")[0]
        # if the event field is 2 strings separated by an underscore
        event_contents = event_contents.split("_")
        if len(event_contents) == 2:
                # and it has a relevant action type
                if event_contents[0] == "laughter" or event_contents[0] == "filler" or
event_contents[0] == "bc":
                        # and it has both a role code and gender code
                        if len(event_contents[1]) == 2:
                                # and these codes are valid
                                if event_contents[1][0] == "r" or event_contents[1][0] == "c":
                                        if event_contents[1][1] == "M" or event_contents[1][1] ==
"F":
                                                # add this row to the cleaned data
                                                duration = float(row["End time"]) - float(row["Start
time"])
                                                cleaned_data_list.insert(0, {"Fnn": row["Fnn"],
"Action": event_contents[0], "Role": event_contents[1][0], "Gender": event_contents[1][1], "Duration":
duration})

cleaned_data = pd.concat([cleaned_data, pd.DataFrame(cleaned_data_list)], ignore_index=True,
sort=True)

#############################################################

def GetChiSquareResults(obs, exp):
        # Get chi square and p value
        chi_sq_results = stats.chisquare(f_obs=obs, f_exp=exp)

        chi_square = chi_sq_results[0]
        p_value = chi_sq_results[1]

        print("Chi square:", chi_square, "P value:", p_value)


# Part 1 - Test 1
print("\nTest 1")

# Count the number of observed laughs
female_laughs = sum((cleaned_data["Gender"] == "F") & (cleaned_data["Action"] == "laughter"))
```

```
male_laughs = sum((cleaned_data["Gender"] == "M") & (cleaned_data["Action"] == "laughter"))
total_laughs = female_laughs + male_laughs

# Calculate total talk time and total female talk time
total_duration_uncleaned = 0.0
female_duration_uncleaned = 0.0
for index, row in data.iterrows():
        event_contents = row['Event'].split(" ")[0].split("_")
        if len(event_contents) > 0 and event_contents[0] != "silence":
                total_duration_uncleaned += float(row["End time"]) - float(row["Start time"])
                if len(event_contents) > 1:
                        if len(event_contents[1]) == 2:
                                if event_contents[1][1] == "F":
                                        female_duration_uncleaned += float(row["End time"]) -
float(row["Start time"])
                        else:
                                if event_contents[1] == "F":
                                        female_duration_uncleaned += float(row["End time"]) -
float(row["Start time"])

print("Total duration from all data =", total_duration_uncleaned, "Female duration from all data =",
female_duration_uncleaned)

# Calculate expected female laughs and male laughs
all_females = cleaned_data[cleaned_data["Gender"] == "F"]

total_duration = total_duration_uncleaned
female_duration = female_duration_uncleaned
# alternate method - ignores data that isn't laugh, filler or bc
#female_duration = all_females["Duration"].sum()
#total_duration = cleaned_data["Duration"].sum()

fraction_female = female_duration / total_duration
print("p_f =", fraction_female)
print("p_m =", 1 - fraction_female)

expected_female_laughs = fraction_female * total_laughs
expected_male_laughs = total_laughs - expected_female_laughs
print("Observations: Female =", female_laughs, "Male =", male_laughs)
print("Expectations: Female =", expected_female_laughs, "Male =", expected_male_laughs)

GetChiSquareResults([female_laughs, male_laughs], [expected_female_laughs, expected_male_laughs])


# Part 1 - Test 2
print("\nTest 2")

# Count the number of observed laughs
female_fillers = sum((cleaned_data["Gender"] == "F") & (cleaned_data["Action"] == "filler"))
male_fillers = sum((cleaned_data["Gender"] == "M") & (cleaned_data["Action"] == "filler"))
total_fillers = female_fillers + male_fillers

# Calculate expected female laughs and male laughs
expected_female_fillers = fraction_female * total_fillers
expected_male_fillers = total_fillers - expected_female_fillers
print("Observations: Female =", female_fillers, "Male =", male_fillers)
print("Expectations: Female =", expected_female_fillers, "Male =", expected_male_fillers)

GetChiSquareResults([female_fillers, male_fillers], [expected_female_fillers, expected_male_fillers])

############################################################
import matplotlib.pyplot as plt
```

```
import matplotlib.mlab as mlab
import math

def GetStudentsTResults(vec_x, vec_y, alpha):
        #Calculate the variance with N-1
        variance_x = vec_x.var(ddof=1)
        variance_y = vec_y.var(ddof=1)

        mean_x = vec_x.mean()
        mean_y = vec_y.mean()

        n_x = len(vec_x)
        n_y = len(vec_y)

        print("Female mean =", mean_x)
        print("Male mean =", mean_y)
        print("Female variance =", variance_x)
        print("Male variance =", variance_y)

        # show graphed distributions
        sigma_x = math.sqrt(variance_x)
        sigma_y = math.sqrt(variance_y)
        x = np.linspace(mean_x - 3*sigma_x, mean_x + 3*sigma_x, 100)
        y = np.linspace(mean_y - 3*sigma_y, mean_y + 3*sigma_y, 100)
        plt.plot(x, stats.norm.pdf(x, mean_x, sigma_x), 'r', label='Females')
        plt.plot(y, stats.norm.pdf(y, mean_y, sigma_y), 'b', label='Males')
        plt.xlabel("Duration")
        plt.legend()
        plt.show()


        # Calculate the t-statistic
        t = (mean_x - mean_y) / (np.sqrt( (variance_x / n_x) + (variance_y / n_y) ))

        deg_of_freedom = n_x + n_y - 2

        #p-value after comparison with the t
        p = 1 - stats.t.cdf(t, df=deg_of_freedom)

        print("My implementation: t =", t)
        print("My implementation: p =", 2*p) # Multiply the p value by 2 because it's a 2 tail t-test

        if p > alpha:
                print('Accept null hypothesis')
        else:
                print('Reject the null hypothesis')

        # Cross Checking with the internal scipy function
        t2, p2 = stats.ttest_ind(vec_x, vec_y, equal_var=False)
        print("SciPy implementation: t = ", t2)
        print("SciPy implementation: p = ", p2)


# Part 1 - Test 3
print("\nTest 3")

all_males = cleaned_data[cleaned_data["Gender"] == "M"]

all_female_laughs = all_females[all_females["Action"] == "laughter"]
all_male_laughs = all_males[all_males["Action"] == "laughter"]

female_laughs_lengths = all_female_laughs["Duration"]
```

```
male_laughs_lengths = all_male_laughs["Duration"]

GetStudentsTResults(female_laughs_lengths, male_laughs_lengths, 0.05)


# Part 1 - Test 4
print("\nTest 4")

all_female_fillers = all_females[all_females["Action"] == "filler"]
all_male_fillers = all_males[all_males["Action"] == "filler"]

female_fillers_lengths = all_female_fillers["Duration"]
male_fillers_lengths = all_male_fillers["Duration"]

GetStudentsTResults(female_fillers_lengths, male_fillers_lengths, 0.05)
```

Part 2 - Code

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

data = pd.read_csv("data-part-2.csv")

#Evenly spread smiles and frowns (for better folds)
reordered_data_list = []
for i in range(0,26):
        reordered_data_list.append(data.iloc[i,:].values)
        reordered_data_list.append(data.iloc[-(i+1),:].values)

reordered_data = np.vstack(reordered_data_list)

############################################################

def SplitTrainAndTest(features_matrix, classes_vector, k_folds, fold_no):
        vectors_count = features_matrix.shape[0]

        test_length = int(round(vectors_count / k_folds))

        split_start = test_length * fold_no
        split_end = split_start + test_length

        X_train = np.append(features_matrix[:split_start,:], features_matrix[split_end:,:], axis=0)
        y_train = np.append(classes_vector[:split_start], classes_vector[split_end:], axis=0)

        X_test = features_matrix[split_start:split_end,:]
        y_test = classes_vector[split_start:split_end]

        return X_train, y_train, X_test, y_test

def CalculateLogLikelihood(feature_values, means, variances):
        euclid_dist = np.power((feature_values - means),2)
        divisor = variances * 2
```

```
        std_dev_term = variances * 2*np.pi
        std_dev_term = np.sqrt(std_dev_term)
        st_dev_term = np.log(std_dev_term)

        log_likelihoods = st_dev_term + (euclid_dist / divisor)

        log_lh = -np.sum(log_likelihoods)

        return log_lh

def MakePredictions(X, y, c1_means, c2_means, c1_variances, c2_variances):
        predictions = []
        for row in range(0, X_test.shape[0]):
                frown_log_lh = CalculateLogLikelihood(X_test[row,:], c1_means, c1_variances)
                smile_log_lh = CalculateLogLikelihood(X_test[row,:], c2_means, c2_variances)

                if(frown_log_lh > smile_log_lh):
                        predictions.append("frown")
                else:
                        predictions.append("smile")

        # Now check to see how many predictions we got wrong
        errors = 0
        for i in range (0, len(predictions)):
                if predictions[i] != y_test[i]:
                        errors += 1

        return predictions, errors


# Part 2

X = np.array(reordered_data[:,:-1], dtype="float64")
y = reordered_data[:,-1]
k = 4

best_errors = np.iinfo(np.int32).max
for fold in range(0,k):
        X_train, y_train, X_test, y_test = SplitTrainAndTest(X, y, k_folds=k, fold_no=fold)

        X_frowns = X_train[np.where(y_train == "frown")]
        X_smiles = X_train[np.where(y_train == "smile")]

        frowns_means = np.mean(X_frowns, axis=0)
        frowns_variances = np.var(X_frowns, axis=0)
        smiles_means = np.mean(X_smiles, axis=0)
        smiles_variances = np.var(X_smiles, axis=0)

        predictions, errors = MakePredictions(X_test, y_test, frowns_means, smiles_means,
frowns_variances, smiles_variances)

        if(errors < best_errors):
                best_errors = errors
                best_fold = fold
                c0_means = frowns_means
                c0_variances = frowns_variances
                c1_means = smiles_means
                c1_variances = smiles_variances

        print("Fold no. =", fold, "Error =", np.around(errors/len(predictions)*100, 3), "%")
```

```
print("\nBest fold =", best_fold)


predictions, errors = MakePredictions(X, y, c0_means, c1_means, c0_variances, c1_variances)

print("Test Error =", np.around(errors/len(predictions)*100, 3), "%")
```