**Machine Learning – project:**

# Gender Classification By Voice Records


**Name:**

Moran Oshia ID: 313292633

Amit Bibi ID: 203262647


**Moderator:**

Dr. Lee-Ad Gottlieb

# Table of contents:

# Introduction:

In this paper, we will summarize our ML final project.

We will describe the techniques, database, libraries, code we used and the difficulties we had during the process, what we did to solve them and the results of the project.

We decided to choose for our final project in the Machine Learning course “Gender Recognition by Voice Records”, our models are built to classify between men and women.

We used 5 techniques: CNN, KNN, logistic regression, decision tree and SVM.

The main purpose of machines is to predict according to the 20 characteristics of the record whether it is male or female.

# Database:

The Database is taken from Kaggle.

The Database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers, divided equally between male and female.

The database is containing 21 acoustic properties of each voice:

- meanfreq: mean frequency (in kHz)
- sd: standard deviation of frequency
- median: median frequency (in kHz)
- Q25: first quantile (in kHz)
- Q75: third quantile (in kHz)
- IQR: interquantile range (in kHz)
- skew: skewness (see note in specprop description)
- kurt: kurtosis (see note in specprop description)
- sp.ent: spectral entropy
- sfm: spectral flatness

- mode: mode frequency
- centroid: frequency centroid.
- peakf: peak frequency (frequency with highest energy).
- meanfun: average of fundamental frequency measured across acoustic signal.
- minfun: minimum fundamental frequency measured across acoustic signal.
- maxfun: maximum fundamental frequency measured across acoustic signal.
- meandom: average of dominant frequency measured across acoustic signal.
- mindom: minimum of dominant frequency measured across acoustic signal.
- maxdom: maximum of dominant frequency measured across acoustic signal.
- dfrange: range of dominant frequency measured across acoustic signal.
- modindx: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range.
- label: male or female 1 for man and 0 for woman

**print(data.describe()):**

| | meanfreq | sd | median | Q25 | Q75 | IQR | skew | kurt | sp.ent |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.059781 | 0.064241 | 0.032027 | 0.015071 | 0.090193 | 0.075122 | 12.863462 | 274.402906 | 0.893369 |
| 1 | 0.066009 | 0.067310 | 0.040229 | 0.019414 | 0.092666 | 0.073252 | 22.423285 | 634.613855 | 0.892193 |
| 2 | 0.077316 | 0.083829 | 0.036718 | 0.008701 | 0.131908 | 0.123207 | 30.757155 | 1024.927705 | 0.846389 |
| 3 | 0.151228 | 0.072111 | 0.158011 | 0.096582 | 0.207955 | 0.111374 | 1.232831 | 4.177296 | 0.963322 |
| 4 | 0.135120 | 0.079146 | 0.124656 | 0.078720 | 0.206045 | 0.127325 | 1.101174 | 4.333713 | 0.971955 |

| sfm | ... | centroid | meanfun | minfun | maxfun | meandom | mindom | maxdom | dfrange | dfrange | modindx | label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.491918 | ... | 0.059781 | 0.084279 | 0.015702 | 0.275862 | 0.007812 | 0.007812 | 0.007812 | 0.000000 | 0.000000 | 0.000000 | male |
| 0.513724 | ... | 0.066009 | 0.107937 | 0.015826 | 0.250000 | 0.009014 | 0.007812 | 0.054688 | 0.046875 | 0.046875 | 0.052632 | male |
| 0.478905 | ... | 0.077316 | 0.098706 | 0.015656 | 0.271186 | 0.007990 | 0.007812 | 0.015625 | 0.007812 | 0.007812 | 0.046512 | male |
| 0.727232 | ... | 0.151228 | 0.088965 | 0.017798 | 0.250000 | 0.201497 | 0.007812 | 0.562500 | 0.554688 | 0.554688 | 0.247119 | male |
| 0.783568 | ... | 0.135120 | 0.106398 | 0.016931 | 0.266667 | 0.712812 | 0.007812 | 5.484375 | 5.476562 | 5.476562 | 0.208274 | male |

Link to the Database:
https://www.kaggle.com/primaryobjects/voicegender

# Project Description:

We used the Panda library to read the data from the CSV database file.

We split our data into groups: Y and X, Y which holds the label and X which holds all the 20 acoustic properties.

Afterward, we splitted these groups to 2 parts of train and test sets, with the function **train_test_split** import from sklearn.model_selection, we chose to split it to 20% test and 80% train.

We ran in for 100 rounds and splitted each round to train and test, afterward we sent the train and test group to all of the different techniques we chose.at the end we calculate the average result for the training for all of the techniques.

# The techniques:

## SVM:

There are several kernel types that can be sent to the machine, each type is cutting the information in a different way.

The SVM model is known as a model capable of handling high-dimensional samples, even when the feature dimension is greater than the number of samples as in our case with voice subtleties between men and women.

We sent 4 types of kernel: with random state =1, Linear, polyand and rbf.

The results:

```
achine.
_____________________________________SVM_____________________________________

Svm Accuracy:  97.73974763406939
Linear Svm Accuracy:  97.41798107255518
Poly Svm Accuracy:  97.83596214511039
Rbf Svm Accurancy:  97.73974763406939
____________________________________________________________________________
```

## KNN:

The k-nearest neighbour's algorithm assumes that similar things exist nearby. In other words, similar things are near to each other.

The K-NN is a non-parametric method proposed for classification and regression.

Search for a nearby neighbour, get the classification according to the nearest neighbours.

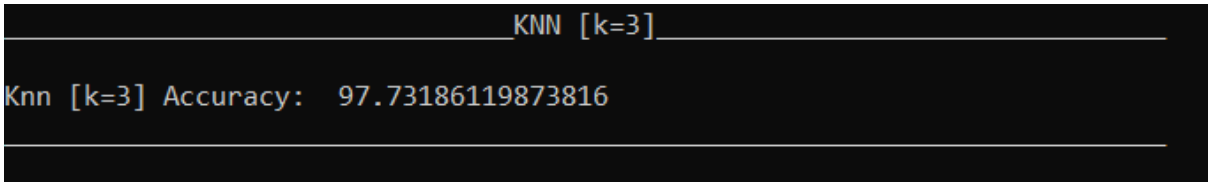
We implemented the model twice with the library Sklearn.

The first implementation is when k = 3 we run like the others technique 100 times and take the average of the training test we got.

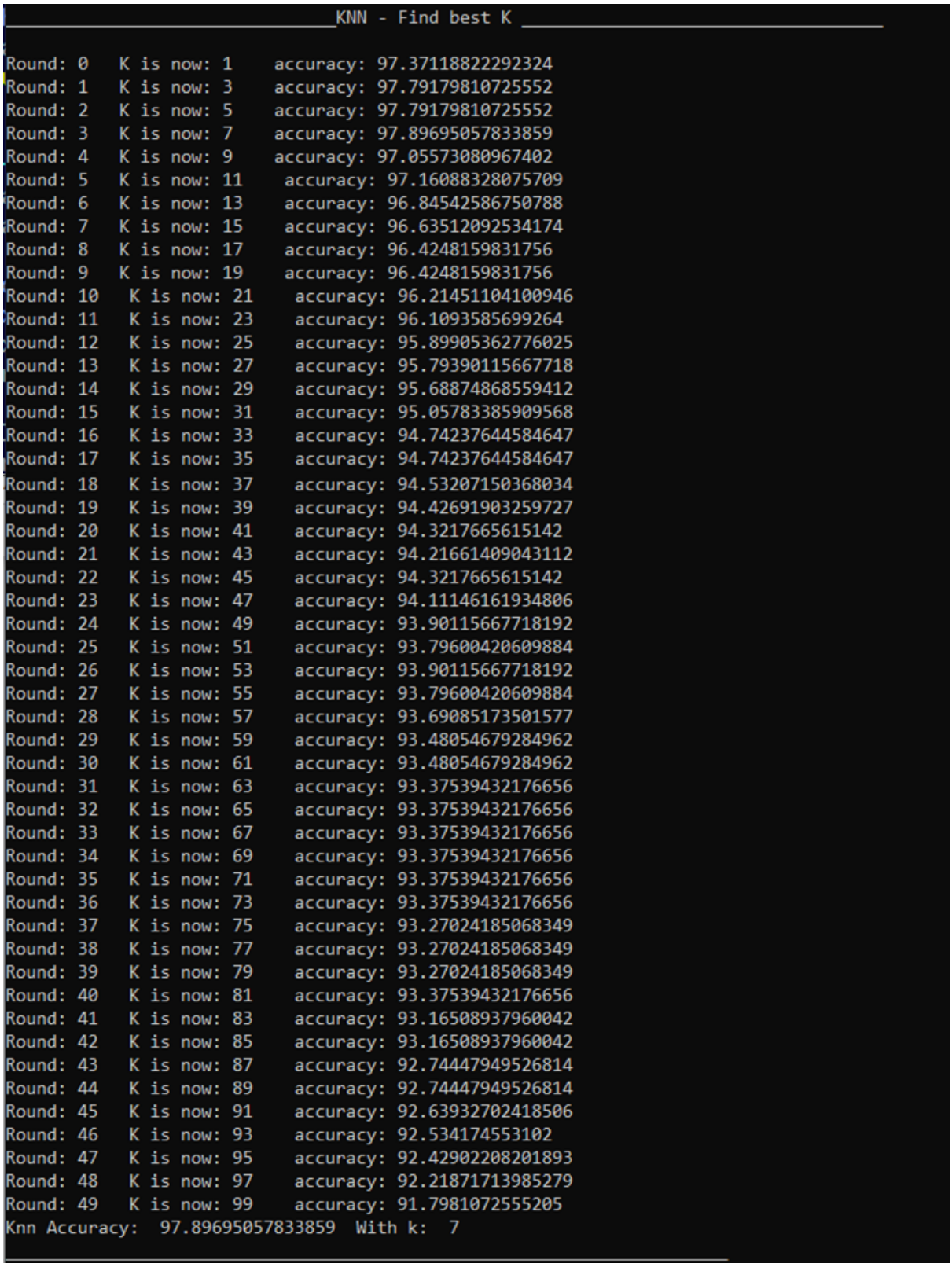
The second implementation is BestKnn which runs with k =1 to 100 for 50 rounds and found out which of the k has the best result. We got that the best result was when k = 7

The results:

K = 3:



Find the best k:



## Logistic Regression:

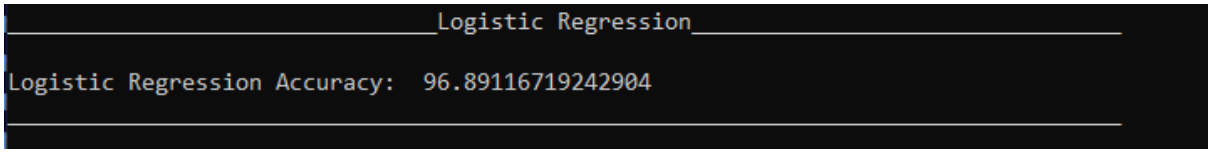
Logistic regression is a function that translates the input into one of two categories, the “classic” application of logistic regression model is a binary classification.

You can think of logistic regression as an on-off switch.

It can stand alone, or some version of it may be used as a mathematical component to form switches, or gates, that relay or block the flow of information.

We implemented the model with the library called Sklearn as well like the KNN model.

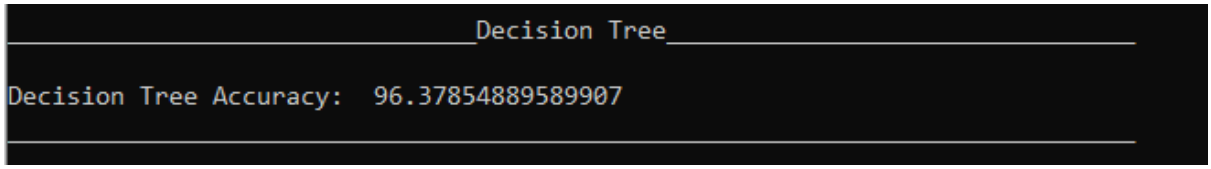
The results:




## Decision Tree:

The goal is to create a model that predicts the value of a target variable by learning simple decision rules derived from data attributes.

The results:



## CNN:

A Convolutional Neural Network (ConvNet / CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.

The results:

Cnn:

```
Epoch 1/100
56/56 [==============================] - 0s 7ms/step - loss: 0.4075 - accuracy: 0.8472 - auc: 0.9387 - val_loss: 0.2106 - val_accuracy: 0.9392 - val_auc: 0.9931
Epoch 2/100
56/56 [==============================] - 0s 3ms/step - loss: 0.1516 - accuracy: 0.9611 - auc: 0.9916 - val_loss: 0.1016 - val_accuracy: 0.9707 - val_auc: 0.9957
Epoch 3/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0944 - accuracy: 0.9707 - auc: 0.9946 - val_loss: 0.0750 - val_accuracy: 0.9797 - val_auc: 0.9967
Epoch 4/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0767 - accuracy: 0.9746 - auc: 0.9959 - val_loss: 0.0690 - val_accuracy: 0.9865 - val_auc: 0.9967
Epoch 5/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0717 - accuracy: 0.9763 - auc: 0.9960 - val_loss: 0.0605 - val_accuracy: 0.9820 - val_auc: 0.9973
Epoch 6/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0654 - accuracy: 0.9797 - auc: 0.9964 - val_loss: 0.0570 - val_accuracy: 0.9842 - val_auc: 0.9976
Epoch 7/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0629 - accuracy: 0.9797 - auc: 0.9966 - val_loss: 0.0526 - val_accuracy: 0.9887 - val_auc: 0.9978
Epoch 8/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0566 - accuracy: 0.9825 - auc: 0.9969 - val_loss: 0.0503 - val_accuracy: 0.9887 - val_auc: 0.9981
Epoch 9/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0529 - accuracy: 0.9842 - auc: 0.9977 - val_loss: 0.0565 - val_accuracy: 0.9865 - val_auc: 0.9974
Epoch 10/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0524 - accuracy: 0.9825 - auc: 0.9972 - val_loss: 0.0634 - val_accuracy: 0.9842 - val_auc: 0.9969
Epoch 11/100
56/56 [==============================] - 0s 3ms/step - loss: 0.0490 - accuracy: 0.9836 - auc: 0.9983 - val_loss: 0.0550 - val_accuracy: 0.9865 - val_auc: 0.9973
```

```
30/30 [==============================] - 0s 1ms/step - loss: 0.0684 - accuracy: 0.9800
```

Cnn 2D:

```
Epoch 1/100
56/56 [==============================] - 0s 8ms/step - loss: 0.6543 - accuracy: 0.6520 - auc: 0.7486 - val_loss: 0.6051 - val_accuracy: 0.7455 - val_auc: 0.8369
Epoch 2/100
56/56 [==============================] - 0s 4ms/step - loss: 0.5365 - accuracy: 0.7998 - auc: 0.8813 - val_loss: 0.4722 - val_accuracy: 0.8041 - val_auc: 0.9096
Epoch 3/100
56/56 [==============================] - 0s 3ms/step - loss: 0.3874 - accuracy: 0.8697 - auc: 0.9427 - val_loss: 0.3433 - val_accuracy: 0.8626 - val_auc: 0.9599
Epoch 4/100
56/56 [==============================] - 0s 3ms/step - loss: 0.2818 - accuracy: 0.9013 - auc: 0.9660 - val_loss: 0.2658 - val_accuracy: 0.8874 - val_auc: 0.9721
Epoch 5/100
56/56 [==============================] - 0s 3ms/step - loss: 0.2230 - accuracy: 0.9160 - auc: 0.9764 - val_loss: 0.2660 - val_accuracy: 0.8829 - val_auc: 0.9773
Epoch 6/100
56/56 [==============================] - 0s 3ms/step - loss: 0.1940 - accuracy: 0.9301 - auc: 0.9801 - val_loss: 0.1944 - val_accuracy: 0.9144 - val_auc: 0.9811
Epoch 7/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1766 - accuracy: 0.9318 - auc: 0.9827 - val_loss: 0.1737 - val_accuracy: 0.9347 - val_auc: 0.9842
Epoch 8/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1633 - accuracy: 0.9380 - auc: 0.9848 - val_loss: 0.1629 - val_accuracy: 0.9392 - val_auc: 0.9852
Epoch 9/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1490 - accuracy: 0.9391 - auc: 0.9873 - val_loss: 0.1504 - val_accuracy: 0.9369 - val_auc: 0.9882
Epoch 10/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1402 - accuracy: 0.9425 - auc: 0.9885 - val_loss: 0.1462 - val_accuracy: 0.9392 - val_auc: 0.9882
Epoch 11/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1333 - accuracy: 0.9453 - auc: 0.9897 - val_loss: 0.1403 - val_accuracy: 0.9392 - val_auc: 0.9904
Epoch 12/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1298 - accuracy: 0.9515 - auc: 0.9899 - val_loss: 0.1275 - val_accuracy: 0.9414 - val_auc: 0.9909
Epoch 13/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1238 - accuracy: 0.9515 - auc: 0.9908 - val_loss: 0.1323 - val_accuracy: 0.9324 - val_auc: 0.9908
Epoch 14/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1202 - accuracy: 0.9504 - auc: 0.9910 - val_loss: 0.1231 - val_accuracy: 0.9505 - val_auc: 0.9929
Epoch 15/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1132 - accuracy: 0.9549 - auc: 0.9925 - val_loss: 0.1119 - val_accuracy: 0.9527 - val_auc: 0.9933
Epoch 16/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1090 - accuracy: 0.9633 - auc: 0.9925 - val_loss: 0.1084 - val_accuracy: 0.9572 - val_auc: 0.9935
Epoch 17/100
56/56 [==============================] - 0s 4ms/step - loss: 0.1063 - accuracy: 0.9605 - auc: 0.9931 - val_loss: 0.1056 - val_accuracy: 0.9505 - val_auc: 0.9939
Epoch 18/100
56/56 [==============================] - 0s 4ms/step - loss: 0.0991 - accuracy: 0.9628 - auc: 0.9937 - val_loss: 0.0997 - val_accuracy: 0.9640 - val_auc: 0.9947
Epoch 19/100
```

```
30/30 [==============================] - 0s 1ms/step - loss: 0.1095 - accuracy: 0.9590
```

## Challenges:

Our difficulty was mostly figuring out how to make the division properly so that everything would work. We wanted the reading of the data to be done another time and to fit all types of functions.

We worked on it for quite some time, read online and tried, until the distribution was successful. Once we understood how to make the division and how to deal with it in the libraries the work on the database began to become clearer.

Understanding the sklearn library and using it properly and its functions was also challenging as it is a library with many options which we are not familiar with.

In addition, writing and implementing CNN's own functions would have been challenging, we tried to implement CNN with the help of TensorFlow and very quickly realized that this is a difficult task that may not bear fruit, and we started to investigate and discovered more about TensorFlow library and it was helpful because we started to understand how to install and use it.

# Conclusions:

Result for all techniques:

| Techniques | Result |
|---|---|
| svm | 97.739 |
| Svm Linear | 97.417 |
| Svm Poly | 97.835 |
| Svm Rbf | 97.739 |
| Knn k=3 | 97.731 |
| Knn - Find best K (k = 7) | 97.896 |
| Logistic regression | 96.891 |
| Decision tree | 96.378 |
| Cnn | 98.002 |
| Cnn 2D | 95.901 |

We did our seminarian presentation on "How AI and ML help to fight the Coruna virus" and we decided to take this project seriously because our final project is in ML.

Our understanding of the methods grew and expanded.
Undoubtedly it can be seen that the methods of deep learning are much better than different methods of ML.

In the Knn method it can be seen that the number of neighbours in the KNN technique affects the success rate of the machine. As the number of neighbours in the machine increases, the success rates increase. I.e. people who are similar to each other in terms of characteristics.
Also, in the Svm, we can see that the results are very similar for the different kernel, so it can be concluded that the kernel is not very effected on the result.

All our techniques came out with fairly close results.
In our opinion, possible reasons that there is no significant technique that is the best is because the database is relatively small and contains only about 3K data.