L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

ICS3206 - Machine Learning, Expert Systems and Fuzzy Logic

Cristina Cutajar* (230802L)

*B.Sc. (Hons) Artificial Intelligence

Study-unit: **Machine Learning, Expert Systems and Fuzzy Logic**

Code: **ICS3206**

Lecturer: **Dr Kristian Guillaumier**

Plagiarism Form

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Cristina Cutajar
Student Name


Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICS3206
Course Code

Machine Learning, Expert Systems and Fuzzy Logic
Title of work submitted

11/01/2023
Date

Introduction

The aim of this project was to perform experimentations and evaluations on how different machine learning algorithms behave in predicting the gender of the speaker. In order for this to be achieved, a ready-made labelled dataset [1] was used which contains a total of 3,168 recorded voice samples, 1584 of which are of male speakers, with the remaining 1584 voice samples being of female speakers. Furthermore, five different algorithms were implemented on this dataset in order to learn to predict whether the speaker is male or female. The five algorithms which were implemented in this project are the following: Artificial Neural Networks (ANN), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Decision Trees and Logistic Regression.

Implementation

This implementation was carried out in Python. First, the dataset was imported and split into 80% training data and the remaining 20% test data. Initially, all the dataset's features were used during training. However, later on, experimentation was performed on these features to evaluate how useful some features are in identifying the gender of the speaker from a voice sample. The different machine learning algorithms were also implemented as follows:

- The Artificial Neural Network algorithm was implemented with the use of the sklearn's MLPClassifier function [2]. Initially, the function parameters which were used were random_state with a value of 1 and max_iter with a value of 400.

- The Support Vector Machine algorithm was implemented with the use of the sklearn's SVC function [3] with the kernel initially set to linear.

- The K-Nearest Neighbours algorithm was implemented with sklearn's KNeighborsClassifier function [4] with the parameters initially set to 15 neighbours, metric set to 'minkowski' and its power parameter was set to 1.

- The Decision Trees algorithm was implemented with the use of the sklearn's DecisionTreeClassifier function [5] with the random state initially set to 1.

- The Logistic Regression algorithm was implemented with the use of the sklearn's LogisticRegression function [6] with the maximum iterations initially set to 10000.

The algorithms were then individually trained with the use of the fit() function with the training set features and labels as the parameters. A prediction with the multi-layer perceptron classifier was then generated by calling the predict() function and passing in the test set without the label as the parameter. The score of each model was then generated by passing the test set with the labels in the score() function. For each model, the confusion matrix table was then constructed with the generated prediction and actual values to display the amount that the model predicted correctly and incorrectly. A classification report as well as the accuracy score, for each model, were generated with the use of the model's prediction along with the test set's labels. The time function was used to calculate the execution time taken for each algorithm to perform the training and generate a prediction.

The following metrics will also be calculated and used during experimentations in order to evaluate the algorithms: accuracy, precision, recall and F1-score.

Accuracy is the measure of how many predictions were correct from the total number of predictions. It is calculated by dividing the number of correct predictions from the number of total predictions and multiplying the answer by 100 to get the percentage. [13]

Precision is a measure of how many positive predictions were actually correct. It is calculated by taking the number of true positives divided by the sum of the number of true positives and false positives.[14]

Recall is the measure of how many from the positive observations the model manages to correctly predict. It is calculated by dividing the number of true positives by the sum of the number of true positives with the number of false negatives. [14]

The F1 score is the harmonic mean of recall and precision. This metric is calculated by dividing the multiplication of the precision with the recall by the sum of the precision and recall. The output is then multiplied by 2 to generate the final result. If both the precision and recall are high, the model will obtain a high F1 score and if both the precision and recall are low, the model will obtain a low F1 score. Meanwhile, if either the precision or the recall is low whilst the other is high, the F1 score will be medium. [13]

Experiments and Evaluation

Dataset Features

Experiment 1:

The initial training was performed using all of the dataset's features and each model was trained using the initial parameters as specified above. The accuracy percentages of the models, rounded to the nearest 2 decimal places, along with the time taken for each model to be trained and generate a prediction were:

- Artificial Neural Network: 94.32% in 13.9s
- Support Vector Machine: 91.01% in 2.67s
- K-Nearest Neighbours: 76.81% in 93.8ms
- Decision Tree: 96.21% in 31.2ms
- Logistic Regression: 90.38% in 859ms

As can be seen above, in this case, the most accurate model was the Decision Tree algorithm with an accuracy of 96.21% at predicting the correct gender from the voice samples. As can be seen in Figure 1, this model correctly identified 314 from 327 female voice samples and 296 out of 307 male voice samples. Apart from this, in the Decision Tree's classification report, which is shown in Figure 2, for the female voice samples, the model had a precision of 97%, a recall of 96% and an f1-score of 96%. For the male voice samples, the model had a precision of 96%, a recall of 96% and an f1-score of 96%. The Decision Tree algorithm was also the fastest at performing the training and generating a prediction.

Prediction / Actual	female	male	All
female	314	13	327
male	11	296	307
All	325	309	634

Figure 1: Decision Tree Confusion Matrix

Decision Tree Class report:

	precision	recall	f1-score	support
female	0.97	0.96	0.96	327
male	0.96	0.96	0.96	307
accuracy			0.96	634
macro avg	0.96	0.96	0.96	634
weighted avg	0.96	0.96	0.96	634

Figure 2: Decision Tree's classification report

Meanwhile, from the above accuracy scores, the least accurate model was the K-Nearest Neighbours model with an accuracy of 76.81% at predicting the correct gender. Figure 3 displays the confusion matrix of this algorithm where it correctly predicted out 238 of 327 female voice samples and 249 out of 307 male voice samples. The KNN's classification report displayed in Figure 4, shows that the model had a precision of 80%, recall of 73% and f1-score of 76% for the female voice samples and a precision of 74%, recall of 81% and f1-score of 77% for the male voice samples.

Prediction / Actual	female	male	All
female	238	89	327
male	58	249	307
All	296	338	634

Figure 3: K-Nearest Neighbours Confusion Matrix

KNN report:

	precision	recall	f1-score	support
female	0.80	0.73	0.76	327
male	0.74	0.81	0.77	307
accuracy			0.77	634
macro avg	0.77	0.77	0.77	634
weighted avg	0.77	0.77	0.77	634

Figure 4: K-Nearest Neighbours' classification report

Experiment 2:

Experimentation was performed on the dataset features to see which features are useful and which aren't. The meanfreq, sd, median, mode, centroid, meanfun, minfun, maxfun, meandom, mindom, maxdom, dfrange, modindx features were chosen for this experiment as they appeared more essential for recognising genders than the other features which were Q25, Q75, IQR, skew, kurt, sp.ent, sfm.

Upon performing the training using the algorithms with their initial parameters, the following results were obtained:

- Artificial Neural Network: 97.00% accuracy in 14.6s
- Support Vector Machine: 96.21% accuracy in 656ms
- K-Nearest Neighbours: 79.18% accuracy in 62.5ms
- Decision Tree: 95.58% accuracy in 31.2ms
- Logistic Regression: 90.38% accuracy in 250ms

In this case, the most accurate model was the Artificial Neural Network with an accuracy of 97%. With these chosen training features, this model got a greater accuracy than when the entire dataset was used as training data. However, despite there being less values to train on, the model took a bit longer to train and generate a prediction. The Support Vector Machine model also returned a greater accuracy in this experiment however it took much less time to perform the training and generate a prediction. Likewise, the K-Nearest Neighbours also got a greater accuracy in less time than when all the dataset features were used.

The Decision Tree model took the same amount of time to perform the training and generate a prediction however it got a slightly lower accuracy percentage with the chosen features rather than with all the dataset features.The Logistic Regression model generated the same accuracy percentage however it took less time to train and generate a prediction using the chosen features from the dataset rather than when all the dataset features were used.

From this experiment, it was noted that when certain features were removed from the training set, the majority of the models produced a higher accuracy. Since the algorithms got a higher accuracy result with less features given, except for the Decision Tree model which returned a slightly lower accuracy result and the Logistic Regression model which returned the same accuracy result, making use of all the dataset features may have led the model to be slightly

overfit. The above results also show that the Q25, Q75, IQR, skew, kurt, sp.ent and sfm are likely less important features.

Experiment 3:

Another experiment was then performed by unselecting the mode and centroid from the previously selected features. This was done to check whether the mode and centroid features were useful or not. Therefore, training was now performed with the meanfreq, sd, median, meanfun, minfun, maxfun, meandom, mindom, maxdom, dfrange, modindx features.
From this experiment, the following accuracy results were obtained with their respective execution time taken:

- Artificial Neural Network: 96.69% accuracy in 32.1s
- Support Vector Machine: 96.06% accuracy in 1.09s
- K-Nearest Neighbours: 81.23% accuracy in 46.9ms
- Decision Tree: 96.21% accuracy in 46.9ms
- Logistic Regression: 90.22% accuracy in 719 ms

Like the previous experiment, the Artificial Neural Network was the most accurate however in this experiment, this model was slightly less accurate, and it took more time to train and generate a prediction. The Support Vector Machine was also slightly less accurate with more time taken for the execution than the previous experiment. The K-Nearest Neighbours model performed better in this experiment than the previous experiment as it returned a higher accuracy percentage, and it took less time to train and generate the predictions. The Decision Tree model was more accurate, but it took slightly longer to execute than the previous experiment whilst the Logistic Regression was slightly less accurate and it took more time to execute than the previous experiment. 3 out of the 5 algorithms performed slightly better in this experiment than in the previous experiment, however, overall, this experiment provided a slightly better total average amongst the algorithms. This experiment had an average of 92.08% whilst the previous experiment had an average of 91.67%. Due to this, another 2 experiments were carried out to check if using only the mode or the centroid feature would provide higher accuracy results.

Experiment 4:

This experiment was carried out in order to check whether adding the mode feature to the previous experiment's chosen features would result in better results than those gathered from experiments 2 and 3. The following results were obtained:

- Artificial Neural Network: 96.37% accuracy in 35.1s
- Support Vector Machine: 94.79% accuracy in 1.58s
- K-Nearest Neighbours: 77.60% accuracy in 141ms
- Decision Tree: 96.06% accuracy in 62.5ms
- Logistic Regression: 91.01% accuracy in 391ms

As can be seen above, the most accurate model was the Artificial Neural Network with an accuracy of 96.37% in 35.1 seconds. In this experiment, all the models generated a smaller accuracy percentage than the previous experiment. Apart from this, all the models except for Logistic Regression, took more time to train and generate a prediction. Overall, this experiment had an

average of 91.17% accuracy which is less than the average accuracy generated in experiments 2 and 3. Therefore, one can safely conclude that adding only the mode feature to the previous experiment's chosen features was not beneficial as the algorithms ended up producing a less accurate prediction.

Experiment 5:

This experiment was carried out in order to check whether adding the centroid feature to experiment 3's chosen features would result in better results than those gathered from experiments 2 and 3. This experiment resulted in the following accuracy and time results:

- Artificial Neural Network: 96.69% accuracy in 28.8s
- Support Vector Machine: 93.38% accuracy in 734ms
- K-Nearest Neighbours: 80.76% accuracy in 62.5ms
- Decision Tree: 95.90% accuracy in 31.2ms
- Logistic Regression: 88.33% accuracy in 203ms

This experiment resulted in a 91.01% accuracy average which is slightly less than the accuracy average of experiment 4 and hence also less than the average accuracy of experiments 2 and 3. In this experiment, the Artificial Neural Network model provided the greatest accuracy. The Artificial Neural Network and K-Nearest Neighbours models generated slightly higher accuracy percentages in this experiment than in experiment 4. The other models however generated lower accuracy percentages in this experiment than in experiment 4. Therefore, making use of the centroid feature resulted in the algorithms being less accurate at predicting the gender from the voice samples.

In conclusion, Experiment 3 provided the best results and therefore in this case, the most useful features are the meanfreq, sd, median, meanfun, minfun, maxfun, meandom, mindom, maxdom, dfrange, modindx features. Experiments on the algorithms will now be performed with the use of these chosen dataset features.

Artificial Neural Network

An Artificial neural network is a computational network based on the structure of the human brain which is constructed with neurons linked to each other. This is because artificial neural networks also make use of neurons, known as nodes, connected to each other in the layers of the networks [10]. The most common type of artificial neural networks are the multilayer perceptrons (MLPs). This type of artificial neural network makes use of three types of layers, the input layer, the hidden layer and the output layer in order to perform supervised training to solve a wide variety of problems. [11]

The input is passed through the input layer and then, the hidden layers perform the needed calculations in order to derive patterns and hidden features from the input. In the hidden layers, the input is multiplied with its corresponding weights, which is usually the strength of the neurons' connection, and then the weighted input is summarised. Bias is added if the sum of the weighted input is equal to 0 in order for the output to be non-zero. Moreover, this algorithm makes use of a benchmarked maximum value in order to make sure that the output would be in the desired value range. [10] An activation function is used on the sum of the weighted input in order for the algorithm to be able to identify complex patterns from the input. There are different types of activation functions such as Logistic, Binary, Identity, Tanh, ReLU and Leaky ReLU. Finally, the result is then displayed with the use of the output layer.

The Logistic function is a very commonly used activation function in binary classifiers and multi-label classifiers where the probability needs to be predicted. This function however can cause the vanishing gradient problem due to small gradient values since it is continuously differentiable. The binary activation function is a threshold-based classifier which is used with binary classifiers. This function however cannot be used with multiclass classifiers as this function is unable to perform gradient-based optimisations. The identity activation function is mainly used for regression problems. This function does not alter the sum of the weighted inputs and due to this, it is insufficient alone when non-linearities are present. The Tanh function is mainly used for RNNs or classification problems as it is differentiable, anti-symmetrical and centred at zero. A disadvantage of this function is that it may result in the vanishing gradient problem. ReLU is the most popular function for deep learning models, such as MLP and CNN, since it introduces sparsity in the model by not activating the neurons at the same time, and hence, it overcomes the issue of vanishing gradients. This function however may result in dead neurons or slow learning. Leaky ReLU is an improved activation function from ReLU with the purpose of overcoming the issue of dead neurons by setting the derivative to be 0.01 instead of 0 for negative valued inputs. [12]

A solver, such as adam, lbfgs or sgd, is used for weight optimization. The adam solver is ideal when dealing with a large number of training samples as with large amounts of data, this solver provides the best validation score in the least amount of time. However, the lbfgs solver is ideal when dealing with small amounts of data as it performs better and converges faster. [2]

In order to experiment on this algorithm, the following parameters were chosen:

- hidden_layer_sizes; refers to the number of neurons present in the corresponding hidden layer.
- activation; refers to the activation function.
- solver; refers to the weight optimisation solver.
- random_state; refers to the number for weights and bias initialisation.
- max_iter; refers to the maximum number of iterations that the solver can perform.

	hidden_layer_sizes	activation	solver	random_state	max_iter	CPU Time	Accuracy
1	(100,)	relu	adam	1	400	35.1s	97.79%
2	(100,)	relu	adam	0	800	30.3s	97%
3	(100,)	relu	adam	0	1000	27.5s	97%
4	(150,100,50)	relu	adam	1	1000	32.7s	97.32%
5	(150,100,50)	relu	adam	1	800	32.3s	97.32%
6	(150,100,50)	relu	adam	1	400	32.7s	97.32%
7	(150,100,50)	relu	adam	1	300	32.3s	97.32%
8	(150,100,50)	relu	adam	0	300	27.2s	96.37%
9	(150,100,50)	relu	adam	0	800	28.7s	96.37%
10	(150,100,50)	relu	adam	2	300	29.4s	95.74%
11	(200,150,100)	relu	adam	1	300	32.4s	96.06%
12	(150,100,50)	logistic	adam	1	300	54s	96.84%
13	(100,)	logistic	adam	1	300	Failed to converge	
14	(150,100,50)	logistic	sgd	1	300	4.3s	48.26%
15	(150,100,50)	logistic	lbfgs	1	300	Failed to converge	
16	(150,100,50)	logistic	lbfgs	1	1000	Failed to converge	
17	(150,100,50)	logistic	lbfgs	0	1000	Failed to converge	
18	(100,)	logistic	lbfgs	1	800	Failed to converge	
19	(150,100,50)	tanh	lbfgs	1	800	Failed to converge	
20	(150,100,50)	tanh	adam	1	800	11.1s	97%
21	(150,100,50)	tanh	adam	1	300	9.91s	97%
22	(150,100,50)	tanh	adam	0	300	15.4s	96.53%

23	(100,)	tanh	adam	1	300	Failed to converge	
24	(150,100,50)	tanh	sgd	1	800	47.2s	88.96%
25	(150,100,50)	identity	sgd	1	300	18.4s	86.28%
26	(150,100,50)	identity	adam	1	300	6.45s	93.85%
27	(100,)	identity	adam	1	300	Failed to converge	
28	(150,100,50)	identity	lbfgs	1	300	Failed to converge	
29	(150,100,50)	identity	lbfgs	1	1000	41.3s	96.37%
30	(100,)	identity	lbfgs	1	1000	13.8s	96.21%

From the above experiments, it can be seen that experiment number 1 resulted in the greatest accuracy percentage in the least amount of time. From experiments number 4, 5 and 6, it can also be seen that upon decreasing the number of maximum iterations, the convergence time and accuracy percentage remained the same since the algorithm had managed to reach convergence before performing 300 iterations. From experiment 8, upon changing the random_state number from 1 to 0, the algorithm became less accurate however it took less time to reach convergence. Meanwhile, from experiment 1, the difference between having 1 hidden layer with 100 neurons and 3 hidden layers with 150, 100 and 50 neurons respectively, is shown. With 1 hidden layer, the algorithm took more time to converge and it was more accurate than when 3 hidden layers were used. As stated in [12], for the MLP classifier, the best activation function was the ReLU. Moreover, as can be seen in the above experiments, the adam solver performed the best. This could be due to the fact that the training dataset was not small and according to [2], the adam solver performs better than the other solvers when handling large amounts of data. In experiment number 21, the Tanh activation function with the adam solver also performed very well. It converged much faster than when the ReLU activation function was used in experiment 7. However, the accuracy percentage that resulted from the Tanh activation function, in experiment 21, was slightly less than the accuracy percentage from the ReLU activation function, in experiment 7.

The Tanh activation function however failed to converge, in experiment 23, when the adam solver was used with a maximum of 300 iterations and 1 hidden layer. Moreover, as can be seen in experiments 15 to 19, both the Tanh and Logistic activation functions failed to converge when making use of the lbfgs solver, even with 1000 iterations. The logistic activation function also failed to converge when it was used with 1 hidden layer, as seen in experiment 13 where the adam solver was used. The identity activation function however failed to converge when the maximum iteration value was set to below 1000 when making use of the lbfgs solver. This can be seen in experiment 28. Moreover, as can be seen in experiment 27, it also failed to converge when making use of 1 hidden layer and a maximum of 300 iterations with the adam solver.

Support Vector Machine

The support vector machine is a type of supervised learning algorithm which is most often used for classification tasks however it is also used to solve regression tasks. This algorithm works by separating the data into different classes by selecting the best hyperplane in the N-dimensional space. The type of hyperplane depends on the number of input features. The support vector machine can handle both linearly separable data as well as non-linearly separable data with the use of a kernel. The kernel will transform the low dimensional input space into a high dimensional space in order to convert the non-linearly separable data to linearly separable data. [15]

In order to experiment on this algorithm, the following parameters were chosen:

- kernel; refers to the type of kernel used.
- gamma; refers to the coefficient used for the rbf, poly and sigmoid kernels.
- cache_size; refers to the kernel's cache size in MB.

	kernel	gamma	cache_size	CPU Time	Accuracy
1	linear	NA	200	469ms	95.9%
2	linear	NA	500	469ms	95.9%
3	rbf	scale	200	469ms	63.25%
4	rbf	scale	500	469ms	63.25%
5	rbf	auto	200	422ms	70.82%
6	rbf	auto	500	422ms	70.82%
7	poly	scale	200	328ms	66.4%
8	poly	auto	200	516ms	82.49%
9	poly	auto	100	516ms	82.49%
10	sigmoid	scale	200	297ms	38.01%
11	sigmoid	auto	200	297ms	37.54%

From the above experiments, experiment number 1 resulted in the greatest accuracy percentage in the least amount of time. When comparing experiment 1 to experiment 2, it was noted that the increase in cache_size did not make a difference in convergence time or accuracy of the algorithm. This can also be seen when comparing experiment 8 to experiment 9 where the cache_size was decreased from 200 to 100 however the convergence time and generated accuracy remained the same.

The rbf kernel provided less accuracy than the linear kernel however, when the rbf made use of the auto gamma, it generated higher accuracy percentage in less time than when it made use of the scale gamma. This can be seen in experiments 3 to 6. Moreover, these experiments also show that upon increasing the cache_size, no difference was noted in the convergence time or accuracy percentage.

When making use of the poly kernel, convergence was faster when using the scale gamma rather than when the auto gamma was used. However, when using the scale gamma, the model resulted in low accuracy whilst when using the auto gamma, the model resulted in higher accuracy, however still not as high as when making use of the linear kernel. These can be seen in experiments 7 and 8.

From experiments number 10 and 11, it can be seen that when using the sigmoid kernel, convergence was fast however the accuracy was very low, especially when using the auto gamma.

K-Nearest Neighbors

The K-Nearest Neighbors algorithm is a supervised machine learning algorithm that is memory based and non-parametric. This algorithm handles classification and regression problems by generating classes or predictions from the given data with the use of a distance algorithm. It works by assuming that data points next to each other are similar and of the same class. For KNNs, the Euclidean distance is mostly used, however other distance metrics used include Manhattan distance and Minkowski distance.

Euclidean distance measures the straight line from two points whilst Manhattan distance measures the absolute value of the two points. Minkowski distance is the generalised form of the other two distance metrics, based on the given power parameter. When the power parameter is 1, Manhattan distance is used, when the power parameter is 2, the Euclidean distance is used. Moreover, when the power parameter is arbitrary, Minkowski distance is used.

In order to experiment on this algorithm, the following parameters were chosen:

- n_neighbors; refers to the number of neighbours used.
- weights; refers to the weight function that will be used for the prediction.
- algorithm; refers to the algorithm used to generate the nearest neighbours. The auto algorithm will automatically select the most ideal algorithm for the input data.
- leaf_size; refers to the leaf size of the algorithm which affects speed and memory.
- p; refers to the power parameter which determines which distance metric is used.

Moreover, during the experimentations, the metric parameter was kept to 'minkowski' as default.

	n_neighbors	weights	algorithm	leaf_size	p	CPU Time	Accuracy
1	15	uniform	auto	30	1	125ms	81.23%
2	5	uniform	auto	30	1	62.5ms	82.02%
3	5	uniform	auto	30	2	31.2ms	77.29%
4	15	uniform	auto	30	1	62.5ms	76.03%
5	5	distance	auto	30	1	46.9ms	82.97%
6	15	distance	auto	30	1	31.2ms	82.49%
7	10	distance	auto	30	1	31.2ms	83.12%
8	8	distance	auto	30	1	31.2ms	83.75%
9	7	distance	auto	30	1	31.2ms	82.33%

10	9	distance	auto	30	1	46.9ms	82.97%
11	8	uniform	auto	30	1	109ms	82.33%
12	12	distance	ball_tree	30	1	46.9ms	83.44%
13	8	distance	ball_tree	30	1	31.2ms	83.75%
14	8	distance	ball_tree	15	1	31.2ms	83.75%
15	8	distance	kd_tree	30	1	31.2ms	83.75%
16	8	distance	kd_tree	15	1	15.6ms	83.75%
17	8	distance	kd_tree	10	1	31.2ms	83.75%
18	5	distance	kd_tree	10	1	15.6ms	82.97%
19	10	distance	kd_tree	10	1	46.9ms	83.12%
20	10	distance	kd_tree	15	1	62.5ms	83.12%
21	8	distance	kd_tree	15	2	46.9ms	79.5%
22	8	distance	brute	15	1	109ms	83.75%
23	8	distance	brute	10	1	125ms	83.75%
24	10	distance	brute	15	1	109ms	83.12%
25	10	distance	brute	30	1	125ms	83.12%
26	8	uniform	brute	15	1	250ms	82.97%
27	5	uniform	brute	15	1	156ms	82.02%
28	8	uniform	auto	15	1	109ms	82.97%
29	8	uniform	auto	30	1	62.5ms	82.97%
30	8	uniform	auto	45	1	31.2ms	82.97%
31	8	distance	kd_tree	15	2	15.6ms	75.55%
32	8	distance	auto	15	1	15.6ms	82.18%
33	8	distance	auto	15	2	31.2ms	75.56%

From the above experiments, experiment 16 had the best accuracy and time results. However, whilst in experiment 16 the kd_tree algorithm was used, in experiments 22 and 23, the same accuracy was generated with the brute algorithm. The models in these two experiments however both took longer than experiment 16's model. Moreover, when comparing experiments 22 and 23, it can be noted that when using the brute algorithm, when the leaf_size was decreased from 15 to 10, the computational time increased.

When comparing experiments 15 and 16, it can be noted that for the kd_tree algorithm, when the leaf_size is decreased from 30 to 15, the accuracy remains the same however the computation time decreases from 31.2ms to 15.6ms. When comparing experiment 14 and 16, it shows that when using kd_tree, convergence was much faster than when using ball_tree despite the accuracy being the same. Moreover, in comparison of experiments 14 and 15, when using the kd_tree algorithm with leaf_size set to 30, the convergence and accuracy results were the same. When using the ball_tree algorithm, there was no difference in the convergence time or accuracy percentage results when changing the leaf_size from 30 to 15. This can be seen in experiments 13 and 14. In experiments 12 and 13, it can be seen that, from the distance weights with the ball_tree algorithm, when the number of neighbours was decreased from 12 to 8, the convergence time was decreased but the accuracy result increased slightly. Upon comparing experiments 16 and 21, when the p parameter was set from 1 to 2, the accuracy decreased, and the time taken was increased.

When making use of the uniform weight in experiments 26 and 28, with the brute algorithm, the convergence time was much higher than when the auto algorithm was used, the accuracy however remained the same. For the uniform weight with brute algorithm, when the n_neighbors parameter was decreased, both the time and the accuracy were decreased. This can be seen when comparing experiments 26 and 27. This was not the case when using the brute algorithm with the distance function as, as can be seen in experiments 22 and 24, when the n_neighbors parameter was increased, the time remained the same however the accuracy decreased. In experiments 29 and 30, when making use of the uniform weight and auto algorithm, when the leaf_size was increased from 30 to 40, the accuracy remained the same however the computational time decreased by half the time. This effect can also be seen between experiment 28 and 29.

When the auto algorithm was used, in experiment 32, instead of the kd_tree algorithm used in experiment 16, the computation time remained the same however it resulted in lower accuracy.

When comparing experiments 2 and 3, 16 and 31 as well as experiments 32 and 33, when the power parameter p was changed from 1 to 2, the model became less accurate. Therefore, when Manhattan distance was used, the model was more accurate than when Euclidean distance was used.

Decision Tree

Decision Tree is a supervised non-parametric machine learning algorithm used for regression and classification problems. It makes use of a tree structure with a root node as the initial state, internal nodes and leaf nodes connected together with branches [17]. This algorithm can handle both categorical and continuous data [18].

In order to experiment on this algorithm, the following parameters were chosen:

- criterion; refers to the function which is used to measure how accurate the split is.
- splitter; refers to the strategy which is used to perform the split for each node.
- max_depth; refers to the tree's maximum depth.
- random_state; refers to the value which is used to control the estimator's randomness.

	criterion	splitter	max_depth	random_state	CPU Time	Accuracy
1	gini	best	None	1	62.5ms	95.74%
2	gini	best	10	1	62.5ms	95.58%
3	gini	best	5	1	46.9ms	95.43%
4	gini	best	None	None	78.1ms	95.27%
5	gini	best	None	10	46.9ms	94.79%
6	gini	best	None	5	62.5ms	95.74%
7	gini	best	20	5	46.9ms	95.74%
8	gini	best	20	1	62.5ms	95.74%
9	gini	random	20	5	31.2ms	94.48%
10	gini	random	None	1	31.2ms	94.64%
11	entropy	best	None	1	46.9ms	94.64%
12	entropy	best	20	5	62.5ms	95.11%
13	entropy	best	None	5	46.9ms	95.11%
14	entropy	best	None	10	46.9ms	95.43%
15	entropy	best	None	15	46.9ms	94.48%
16	entropy	best	None	20	31.2ms	95.29%
17	entropy	random	None	10	15.6ms	94.01%
18	entropy	random	None	1	15.6ms	95.43%

From the above experiments, one can conclude that experiment 7 returned the best results. Other experiments such as experiments 1, 6 and 8 also returned the same accuracy percentage as experiment 7. However, the model in experiment 7 performed computation in less time than the other experiments which returned the same accuracy percentage.

Upon comparing experiments 1, 2 and 3, when max_depth was changed from None to 10, computation time remained the same however the accuracy decreased. However, when max_depth was changed from 10 to 5, both computation time and accuracy percentage decreased. In experiments 1 and 4, when random_state was changed from 1 to None, it was noted that the model took longer to converge and it produced less accuracy percentage.

When making use of the Gini criterion with the best splitter the following results were noticed. Upon comparing experiment 1 and 6, when max_depth was set to None and random_state was changed from 1 to 5, no change was noted, however, in experiment 5 where random_state was set to 10, the algorithm's computational time was faster but it was less accurate than in experiments 1 and 6. However, in experiments 7 and 8, change was noted in time when random_state was changed from 5 to 1, both cases having max_depth set to 20. In these two experiments, when random_state was set to 5, the algorithm was faster than when the random_state was set to 1. The accuracy remained the same for both experiments.

Moreover, when making use of the entropy criterion with the best splitter, slightly different results were noticed. In experiments 13 and 14, where random_state was changed from 5 to 10, computation time remained the same, but accuracy increased. However, in experiment 15, where random_state was changed to 15, computation time remained the same, but accuracy decreased and then in experiment 16, when random_state was changed to 20, the computation was faster than all three experiments. The accuracy percentage however was greater than the accuracy percentage generated in experiments 13 and 15 but less than the accuracy percentage generated in experiment 14. However, when the random splitter was used with the entropy criterion, in experiments 17 and 18, when random_state was changed from 10 to 1, time remained the same, but accuracy percentage increased.

Comparing experiments 7 and 9, the difference can be seen between best splitter and random, where the random splitter was faster but less accurate than the best splitter. This effect was also noticed when comparing experiment 14 to 17 where the random splitter resulted in faster computation but less accuracy percentage than the best splitter.

The following effects were noted when comparing experiments 1 to 11 as well as 7 to 12, when the criterion was changed from Gini to entropy:

- when the max_depth was set to None with random_state set to 1, the entropy criterion was faster but produced less accuracy than the Gini criterion.
- when the max_depth was set to 20 with random_state set to 5, the entropy criterion was slower as well as less accurate than the Gini criterion.

Finally, from the above experiments, it was noted that the Gini criterion was overall more accurate than the entropy criterion.

Logistic Regression

Logistic Regression is a statistical supervised algorithm used to estimate probabilities with a regression model to perform predictive analysis as well as classification on data. This algorithm is able to perform classification with the use of a decision threshold. The value of the decision threshold depends on the precision and recall values of the classification task. [18][19]

In order to experiment on this algorithm, the following parameters were chosen:

- penalty; refers to the penalty norm.
- solver; refers to the algorithm used for the optimization process.
- max_iter; refers to the number of possible maximum number of iterations that the solver can perform to converge.
- multi_class; refers to the option chosen to handle the data.

It is important to note that the solver depends on the type of penalty chosen as not all solvers can support all penalties. Furthermore, when the multi-class parameter is set to auto, the ovr is chosen for binary data or when the liblinear solver is being used, otherwise the mutlinomial option is chosen.

	penalty	solver	max_iter	multi_class	CPU Time	Accuracy
1	l2	lbfgs	100	auto	188ms	90.22%
2	l2	lbfgs	100	ovr	125ms	90.22%
3	l2	lbfgs	100	multinomial	328ms	93.38%
4	l2	lbfgs	1000	multinomial	328ms	93.38%
5	none	lbfgs	100	auto	Failed to converge	
6	none	lbfgs	1000	auto	1s	96.53%
7	none	lbfgs	1000	ovr	1s	96.53%
8	none	lbfgs	1000	multinomial	1.88s	96.53%
9	l2	liblinear	100	auto/ovr	15.6ms	90.38%
10	l1	liblinear	100	auto/ovr	125ms	95.58%
11	l2	newton-cg	100	auto	125ms	90.22%
12	l2	newton-cg	100	ovr	125ms	90.22%
13	l2	newton-cg	100	multinomial	312ms	93.38%
14	none	newton-cg	100	auto/ovr	234ms	96.53%

15	none	newton-cg	100	multinomial	766s	96.53%
16	l2	sag	100	auto	Failed to converge	
17	l2	sag	100	ovr	Failed to converge	
18	l2	sag	100	multinomial	Failed to converge	
19	l2	sag	1000	auto/ovr	297ms	90.06%
20	l2	sag	1000	multinomial	359ms	93.38%
21	none	sag	1200	auto/ovr	Failed to converge	
22	none	sag	1200	multinomial	Failed to converge	
23	l2	saga	1000	auto	Failed to converge	
24	l2	saga	1000	ovr	Failed to converge	
25	l2	saga	1000	multinomial	Failed to converge	
26	l2	saga	1200	auto	500ms	90.06%
27	l2	saga	1200	ovr	500ms	90.06%
28	l2	saga	1200	multinomial	609ms	93.38%
29	none	saga	1200	auto/ovr	Failed to converge	
30	none	saga	1200	multinomial	Failed to converge	
31	l1	saga	1200	auto/ovr	Failed to converge	
32	l1	saga	1200	multinomial	Failed to converge	

From the above experimentation results, experiment 14 returned the highest accuracy result.

In experiments 1,2 and 3, when the multinomial multi_class was used, the model was less fast but more accurate than when the auto or ovr multi_class parameters were used. This was also the case for experiments 12 and 13. For these mentioned experiments (1, 2, 3, 12, and 13), the l2 penalty was used, however, the results differed when no penalty was used. In experiments 7 and 8 as well as in experiments 14 and 15, when the multi_class parameter was set to multinomial, the model took longer to compute the training and prediction, however, the accuracy percentage remained the same as when the ovr multi_class parameter was used. Therefore, when no penalty was chosen, multinomial and ovr produced the same accuracy results.

When comparing experiments 3 and 4, it was noted that upon increasing max_iter, from 100 to 1000, no change was noted. This is because the model managed to converge in less than 101 iterations. However, in experiment 5, the model failed to converge in 100 iterations but upon increasing the max_iter value to 1000 in experiment 6, the model converged and returned high accuracy results.

The auto multi_class automatically selects either the ovr multi_class or the multinomial multi_class based on the given training data. From the above experiments, it was noted that for this dataset, when the auto multi_class was used, the model was making use of the ovr multi_class. This can be seen when comparing experiments 2 and 7, 11 and 12, 16 and 17, 26 and 27, where the results when choosing auto and ovr were all the same. This was the case for the lbfgs, newton-cg, sag and saga solvers. For the liblinear solver, the multinomial multi_class cannot be used so therefore the auto multi_class will always use the ovr multi_class for this solver.

Upon comparing experiment 9 with experiment 10, for the liblinear solver when the penalty was changed from l2 to l1, the computation time increased but so did the accuracy percentage.

From experiments 12 and 13, when the penalty was set to l2, it can be seen that the multinomial multi_class returned higher accuracy but took longer to converge than the ovr multi_class. However, when the penalty was set to none in experiments 14 and 15, the multinomial multi_class returned the same accuracy percentage as the ovr multi_class, but the model using the multinomial multi_class took longer to converge. Overall, the accuracy generated when no penalty was used was higher than when the l2 penalty was used.

When making use of the sag solver the following effects were noticed. In experiments 17 and 18, when the value of max_iter was set to 100, the model failed to converge for both the ovr and multinomial mutli_class, however then, in experiments 19 and 20, the max_iter value was increased to 1000 and the model converged for both multi_class options. From experiments 19 and 20, it was noted that the multinomial multi_class model took longer to perform the training and predictions but was more accurate than the model using the ovr multi_class. Moreover, when no penalty was used in experiments 21 and 22, the model failed to converge even with 1200 iterations. The same effects were noticed when the saga solver was used, and these can be seen in experiments 27 to 30. Furthermore, in experiments 31 and 32, when using the saga solver, the model did not converge when the penalty was set to l1.

When the saga solver was used, the model converged only when performing 1200 iterations with the l2 penalty. This can be seen in experiments 23 to 32. In experiments 23, 24 and 25, the model failed to converge at 1000 max iterations with the l2 penalty, however when max_iter was then increased to 1200 in experiments 26, 27 and 28, the model converged. In experiments 29 to 32, the model did not converge with 1200 iterations when no penalty was used or when the penalty was set to l1.

Final Experiments

After the most ideal parameters were identified in the previous experiments, experimentation was then performed on the chosen dataset features as well as on the entire dataset to see the difference in results.

Experiment 6:

Experimentation was first performed on the chosen dataset features which included the meanfreq, sd, median, meanfun, minfun, maxfun, meandom, mindom, maxdom, dfrange, and modindx features. This experiment resulted in the following accuracy percentages and computation times:

- Artificial Neural Network: 97.00% in 32.7s
- Support Vector Machine: 94.79% in 1.14s
- K-Nearest Neighbours: 83.60% in 46.9ms
- Decision Tree: 96.37% in 46.9ms
- Logistic Regression: 96.53% in 531ms

As can be seen in the above results, the most accurate model in this experiment was the Artificial Neural Network with an accuracy of 97%. The fastest models however were the Decision Tree and K-Nearest Neighbor models with a computation time of 46.9ms. This experiment had an average accuracy of 93.66%. In this experiment, a higher accuracy percentage resulted from all the models when compared to results gathered from experiment 3, except for the Support Vector Machine model. However, this model's parameter values remained the same from experiment 3.

Experiment 7:

The final experiment was performed with all the dataset features to see the difference in results from experiment 1. The following accuracy percentages and computation times were gathered from this experiment:

- Artificial Neural Network: 96.06% in 35.7s
- Support Vector Machine: 95.58% in 1.08s
- K-Nearest Neighbours: 84.70% in 46.9ms
- Decision Tree: 96.69% in 46.9ms
- Logistic Regression: 96.21% in 734ms

From the above results, the most accurate model in this experiment was the Decision Tree with an accuracy of 96.69%. Like in experiment 6, the fastest models were the Decision Tree and K-Nearest Neighbor models, both having a computation time of 46.9ms. This experiment had an average accuracy of 93.85% which was slightly higher than the average accuracy gathered from experiment 6. In this experiment, all the models generated higher accuracy percentages when compared to results gathered from experiment 1. This shows that the chosen parameter values truly improved the performance of all the models.

Conclusion

Overall, the best algorithms were the Artificial Neural Network and Decision Tree. The Artificial Neural Network performed better with the chosen dataset features whilst the Decision Tree model performed better with the whole dataset. The fastest models were the Decision Tree and K-Nearest Neighbors. Therefore, overall, the Decision Tree model was the most ideal since it provides high accuracy in just a few milliseconds.

Since the average accuracy results of experiments 6 and 7 were very close, it is safe to say that the features which were not chosen were not that useful for the model to identify the gender of the speaker from the voice samples. This is due to the fact that if they were needed, the algorithms would have resulted in lower accuracy without them. Therefore, the important features were the meanfreq, sd, median, meanfun, minfun, maxfun, meandom, mindom, maxdom, dfrange, and modindx features. The other features: mode, centroid, Q25, Q75, IQR, skew, kurt, sp.ent, and sfm were less important for the aim of this project.

Statement of Completion

	Completed:
Implemented artificial neural network	Yes
Implemented support vector machine	Yes
Implemented k-nearest neighbours	Yes
Implemented decision tree learning	Yes
Implemented logistic regression	Yes
Evaluated artificial neural network	Yes
Evaluated support vector machine	Yes
Evaluated k-nearest neighbours	Yes
Evaluated decision tree learning	Yes
Evaluated logistic regression	Yes
Overall comparison of methods and discussion	Yes

References

[1] K. Becker. “Gender Recognition by Voice” kaggle.com. https://www.kaggle.com/datasets/primaryobjects/voicegender (Accessed Dec. 19, 2022)

[2] “sklearn.neural_network.MLPClassifier” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (Accessed Dec. 19, 2022)

[3] “sklearn.svm.SVC” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (Accessed Dec. 19, 2022)

[4] “sklearn.neighbors.KNeighborsClassifier” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier (Accessed Jan. 9, 2022)

[5] “sklearn.tree.DecisionTreeClassifier” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier (Accessed Dec. 19, 2022)

[6] “sklearn.linear_model.LogisticRegression” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (Accessed Dec. 19, 2022)

[7] K. Leung. “Micro, Macro & Weighted Averages of F1 Score, Clearly Explained” towardsdatascience.com. https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f (Accessed Jan. 9, 2022)

[8] S. Aryan. “Sex predn using 5 different models | Beginner” kaggle.com. https://www.kaggle.com/code/shabareesharyan/sex-predn-using-5-different-models-beginner/notebook (Accessed Dec. 19, 2022)

[9] A. Nair. “A Beginner’s Guide To Scikit-Learn’s MLPClassifier” analyticsindiamag.com. https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/ (Accessed Jan. 9, 2022)

[10] “Artificial Neural Network Tutorial” javatpoint.com. https://www.javatpoint.com/artificial-neural-network (Accessed Jan. 9, 2022)

[11] “Artificial Neural Network” sciencedirect.com. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/artificial-neural-network (Accessed Jan. 9, 2022)

[12] A. Ranjith. “Activation functions for Artificial Neural Networks (ANN)” medium.com. https://medium.com/@anuvarshini.ranjith/activation-functions-for-artificial-neural-networks-ann-cc033608b0f5 (Accessed Jan. 9, 2022)

[13] J. Korstanje. “The F1 score” towardsdatascience.com. https://towardsdatascience.com/the-f1-score-bec2bbc38aa6 (Accessed Jan. 12, 2022)

[14] R. Vickery. “8 Metrics to Measure Classification Performance” towardsdatascience.com. https://towardsdatascience.com/8-metrics-to-measure-classification-performance-984d9d7fd7aa (Accessed Jan. 12, 2022)

[15] “Support Vector Machine Algorithm” geeksforgeeks.org. https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (Accessed Jan. 12, 2022)

[16] “What is the k-nearest neighbors algorithm?” ibm.com. https://www.ibm.com/topics/knn (Accessed Jan. 14, 2022)

[17] “What is a Decision Tree?” ibm.com. https://www.ibm.com/topics/decision-trees (Accessed Jan. 14, 2022)

[18] “What is logistic regression?” ibm.com. https://www.ibm.com/topics/logistic-regression (Accessed Jan. 15, 2022)

[19] “Understanding Logistic Regression” geeksforgeeks.org. https://www.geeksforgeeks.org/understanding-logistic-regression/ (Accessed Jan. 15, 2022)