Neural Network Training Using Back-propagation

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

Introduction

A neural network training is a type of supervised (data that has labels) learning which is modelled after the human brain in the sense that it has layers and neurons and each of the neurons and layers connect with each other in order to process data. The three layers in a neural network are a hidden layer, output layer, and an input layer. One way to train the data inside a neural network is through a process called Backpropagation and in Backpropagation values from each neuron are retrieved and those values include the activation, bias, and the weights. An activation of a neuron is the output of a neuron given a set of inputs which will be the output of multiple neurons and the outputs of those neurons are determined by something called the sigmoid function and the sigmoid function is

$$\frac{1}{1+e^{-a}}$$

and what a represents in the equation is an activation from a connected layer so for example if the output layer is connected to the hidden layer the activation value from the hidden layer is involved in the activation of the output layer from the sigmoid function and if the hidden layer is connected to the input layer then the activation from the input layer is put into the sigmoid function of the hidden layer. When the sigmoid function computes a value, this value is an activation number between 0 and 1. This is needed due to the fact that the neural network will take non-linear data and there needs to be a universal unit, and this sigmoid function is the way to do it. Just like how the brain is always updating information so will the neural network and the neural network will update weights, activations, and biases repeatedly in order to find the optimum values and also recognize patterns as efficient as possible.

Updating Values

In order to do update the values of the neurons in each of the layer what must be done is calculate the error which is the percent of the network unrecognizing the pattern and this pattern would be outputs in a set of data. For example, if there's a training set with inputs and those inputs are mapped to the output then it's up to the neural network to correctly recognize the outputs given a set of inputs. This error is $\delta = \partial E/\partial z_j = \partial E/\partial y_j * dy_j/dz_j$ with $\partial E/\partial z_j$ representing the partial derivative of the error, $\partial E/\partial y_j$ representing the partial derivative of the total net input and the total net input is

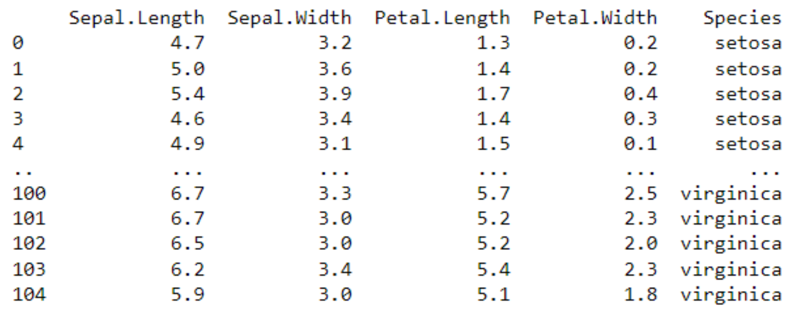
$$\sum_{j=1}^{n} w_j x_j$$

with w representing the weight of a neuron and x representing the input of a neuron and n representing the number of neurons. After the error is calculated then the values of the neurons in each of the three layers such as weights, biases, and inputs will be times by the error and then will repeated any amount of times to get the optimum values. Essentially, with each repetition of the error process what is supposed to be done is for the neural network to be able to recognize the outputs given a set of inputs in a training set much more accurately. Using Python will be able to go about this process.

Neural Network in Python

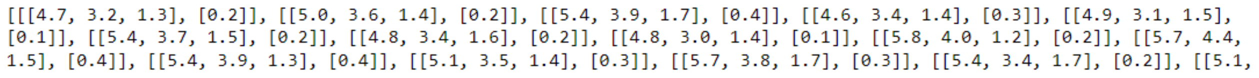
Iris Training Dataset

The neural network will be tested on an iris dataset which will having the following information



```
     Sepal.Length  Sepal.Width  Petal.Length  Petal.Width    Species
0             4.7          3.2           1.3          0.2     setosa
1             5.0          3.6           1.4          0.2     setosa
2             5.4          3.9           1.7          0.4     setosa
3             4.6          3.4           1.4          0.3     setosa
4             4.9          3.1           1.5          0.1     setosa
..            ...          ...           ...          ...        ...
100           6.7          3.3           5.7          2.5  virginica
101           6.7          3.0           5.2          2.3  virginica
102           6.5          3.0           5.2          2.0  virginica
103           6.2          3.4           5.4          2.3  virginica
104           5.9          3.0           5.1          1.8  virginica
```

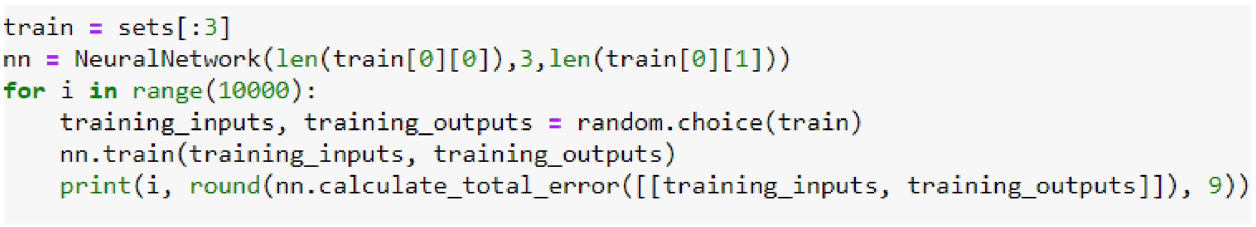
The information that will be obtained in order to test the neural network is the sepal length, sepal width, petal length, and petal width with the x-values being the sepal length, sepal width, and petal length while the petal width will be the y-values. A portion of the training set will look something like this.



```
[[[4.7, 3.2, 1.3], [0.2]], [[5.0, 3.6, 1.4], [0.2]], [[5.4, 3.9, 1.7], [0.4]], [[4.6, 3.4, 1.4], [0.3]], [[4.9, 3.1, 1.5],
[0.1]], [[5.4, 3.7, 1.5], [0.2]], [[4.8, 3.4, 1.6], [0.2]], [[4.8, 3.0, 1.4], [0.1]], [[5.8, 4.0, 1.2], [0.2]], [[5.7, 4.4,
1.5], [0.4]], [[5.4, 3.9, 1.3], [0.4]], [[5.1, 3.5, 1.4], [0.3]], [[5.7, 3.8, 1.7], [0.3]], [[5.4, 3.4, 1.7], [0.2]], [[5.1,
```

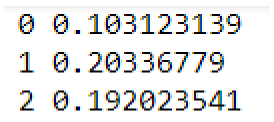
Accuracy of the Neural Network

The way the neural network was designed is to repeat the error process 10,000 times and this is the method to go about it:

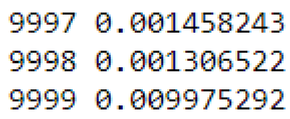


```
train = sets[:3]
nn = NeuralNetwork(len(train[0][0]),3,len(train[0][1]))
for i in range(10000):
    training_inputs, training_outputs = random.choice(train)
    nn.train(training_inputs, training_outputs)
    print(i, round(nn.calculate_total_error([[training_inputs, training_outputs]]), 9))
```

In the method above train is the first three rows of the dataset, nn represents the neural network with backpropagation, training_inputs and training outputs represent the x values and training outputs represent the number that is trying to be recognized by the algorithm, and the method nn.train represents the training of the neural network to update all respective values given the training dataset, and the last line is the probability of wrong outputs that the neural network got.



```
0 0.103123139
1 0.20336779
2 0.192023541
```



```
9997 0.001458243
9998 0.001306522
9999 0.009975292
```

On the left the first three times the algorithm went through the error process the amount the neural network got wrong is between 10 to 20 percent, on the other hand, during the 9,998th through the 10,000th the neural network got the identification of the outputs wrong only 0.14% through 0.09% wrong.

Conclusion

In conclusion, a neural network is a type of supervised learning that is designed based upon the human brain and has uses such as pattern recognition and accurately recognize training set data. The neural network is designed in a way in which through past errors and past values it can train itself to be better in the purpose of recognition and through Python this was able to be simulated. From neural networks, things such as face recognition, credit card fraud detection, and medical diagnosis can be incorporated into everyday society.

References

[1] https://builtin.com/machine-learning/backpropagation-neural-network

[2] Neural Networks-Changyou Chen-Department of Computer Science and Engineering SUNY Buffalo, State University of New York

Utilization of Support Vector Machines on Datasets

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

What is a Support Vector Machine?

A Support Vector Machine is a specific linear classifier that has a good amount of applied and conceptual properties that can be mapped for non-linear classification based on one or several features. Some of the real-world applications from the utilization of a Support Vector Machine is in handwriting recognition, intrusion detection, face detection, email classification, and in webpages. To start the algorithm of a Support Vector Machine is to create hyperplanes (region of a problem space that help classify the data points by classes linearly) and in order to create a hyperplane the data points are used in the formula $wx+b=0$ or $wx+b=-1$ or $wx+b=+1$ in which w is a vector normal to the hyperplane and b represents the bias value or a constant that helps skew a model in a particular way. -1 indicates the hyperplane is in a negative boundary and +1 indicates the boundary is in a negative boundary. After the hyperplanes are formed the best hyperplane is the one that has the maximum margin or the maximum distance between data points of the classes. Mathematically, the Support Vector Machine algorithm may be too difficult to do so by hand and that is when programming languages such as Python could be of great aid.

Maximum Margin

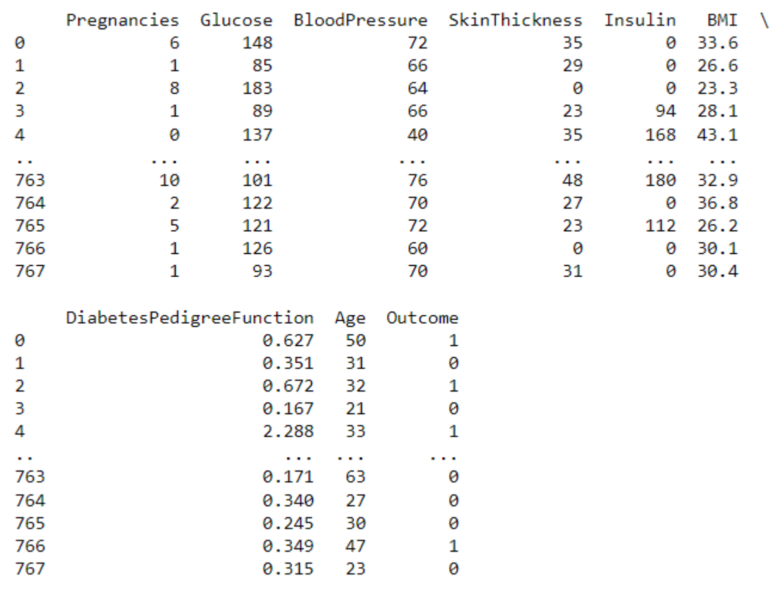
The formula for the maximum margin is

$$2\gamma = \frac{2}{||w||}$$

and the formula above represents the distance between two lines in a hyperplane and the goal of finding the best hyperplane is to find the one with the most maximum margin.

Diabetes Dataset

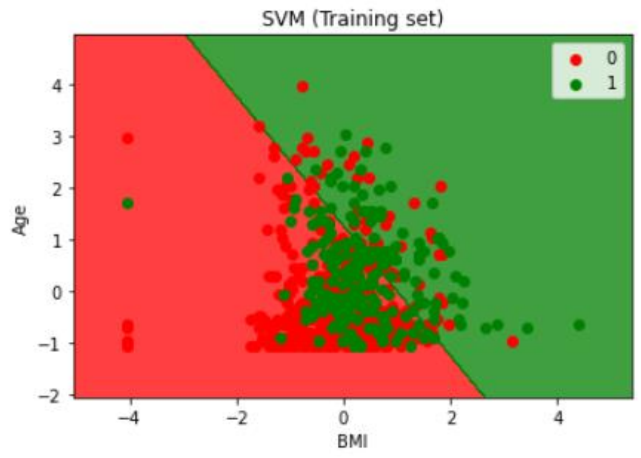
The dataset that will be used is a dataset involving diabetes and in this dataset are several factors that may play a role in a person having diabetes and this would be factors such as glucose, blood pressure, insulin, blood pressure, and age. An outcome of 0 or 1 is in this dataset and what 0 means is the person doesn't have diabetes while 1 means the person has diabetes.



```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

The data values that will be used in the support vector machine is age and BMI which will be the x-values and outcome will represent the y values and in python importation of SVC libraries and confusion matrix will be used in order to visualize the values from the dataset and also predict the values as well.

Visualization and What it Means



As can be seen the two features that were used age and BMI are in two axes and there is a map with the two outcomes with no for diabetes represented in red and yes for diabetes represented in green. Also, what is shown are negative hyperplanes and positive hyperplanes and for the graph above it is shown the hyperplane with 3 on the x-axis is the best hyperplane as that is the one with the greatest separation between multiple age and BMI values.

References

[1]https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[2]https://www.freecodecamp.org/news/svm-machine-learning-tutorial-what-is-the-support-vector-machine-algorithm-explained-with-code-examples/#:~:text=SVMs%20are%20used%20in%20applications,linear%20and%20non%2Dlinear%20data.

[3] http://www.adeveloperdiary.com/data-science/machine-learning/support-vector-machines-for-beginners-linear-svm/

Naïve Bayes Classification in Supervised Learning

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

What is the Naïve Bayes Classification Algorithm?

Naïve Bayes Classification is an algorithm used in machine learning which its algorithm is based upon Bayes Theorem and is used for the solutions of classification-based problems. Mainly this algorithm is used in text classification and like other machine learning models make predictions. These predictions are probability based as this algorithm does observation of inputs and from that observation does one or several probability distributions. In simple terms, a probability distribution is a statistical function that outlines all the potential values and probabilities for a random variable within a given range. The steps and math behind the Naïve Bayes Classification algorithm is largely tied to the meaning of Naïve Bayes itself.

Etymology of Naïve Bayes

Naïve

What Naïve means in this algorithm is that the event of a certain feature in the algorithm is independent of other features used in the algorithm. For example, features such as shape, color, and taste would all be independent of one another, and no feature would influence the value of another feature. These features would be known as independent values or the x values.

Bayes

What Bayes mean in this algorithm is from the essence of the Bayes' Theorem. Bayes Theorem describes the probability of an event based on prior circumstances surrounding an event and is given by the formula $P(A|B)=\frac{\mathrm{P(B|A)P(A)}}{\mathrm{P(B)}}$. In the Bayes Theorem, what P(A|B) represents is the Posterior probability or the probability of assumption A on the observed event B, P(B|A) represents the Likelihood probability or the probability of the evidence given that the probability of an assumption is true, P(A) represents the Prior Probability or the probability of an assumption before actually observing the evidence, and P(B) represents the Marginal Probability or the Probability of Evidence. From the Bayes Theorem, predicted probabilities of events can be done and also can be continuously updated.

Demonstration of Naïve Bayes Classification on a Dataset

Rain Dataset Table

The dataset that will be used to demonstrate Naïve Bayes Classification is a dataset about whether or not it will rain in multiple locations in Australia. Python will be used to use this algorithm on the dataset and before the table of the data from the dataset is given, a specific parameter from the dataset was if the value of rain droplets was 1 millimeters or more then that meant it rained that day so a conversion of that was done by having a condition of 1 millimeter or more meant that it rain so that value is represented by 'Yes' and if not then the value would be ‘No’. Yes or No will be the y-values in the algorithm. When all of those parameters or met this is the resulting data table:

	location	minTemp	maxTemp	WindGustDir	Rain?
0	Albury	13.4	22.9	W	No
1	Albury	7.4	25.1	WNW	No
2	Albury	12.9	25.7	WSW	No
3	Albury	9.2	28.0	NE	No
4	Albury	17.5	32.3	W	Yes
...	...	...	...	...	...
135129	Launceston	18.3	24.5	E	No
135130	Launceston	16.4	33.6	E	No
135131	Launceston	17.0	33.9	NNW	Yes
135132	Launceston	18.4	21.8	N	Yes
135133	Launceston	15.1	20.3	SE	No

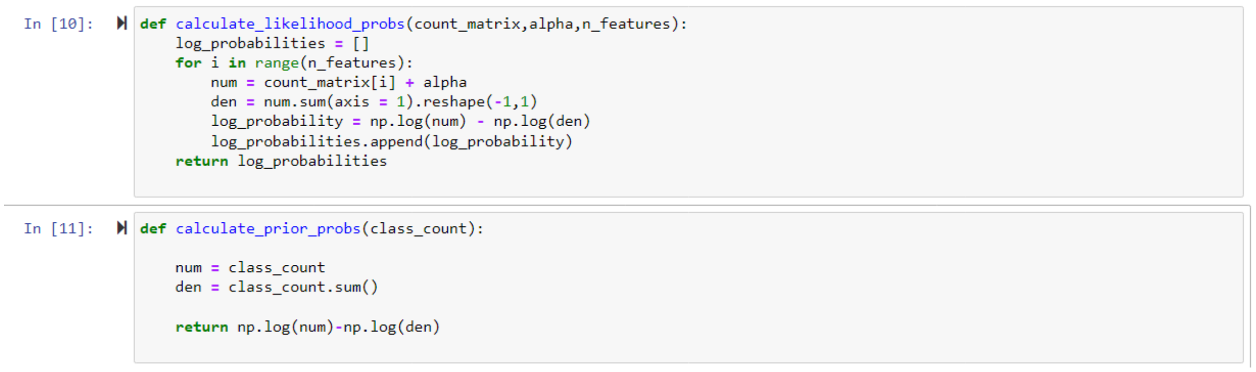
The x-values are location, minTemp(the minimum temperature throughout the day), maxTemp(the maximum temperature throughout the day), WindGust which is the direction the wind was that day, and the y-value ‘Rain?’ which indicates whether or not it rained that day.

Processing the Data

The two classes that are yes and no so now what must be done is based on the occurrence of the values of several futures compute the count of each of them relative to ‘Yes’ or ‘No’. From those calculations a count matrix will be done and a portion of the count matrix would look like this

```
[array([[2245, 2072, 2384, 2383, 2311, 2413, 2245, 2160, 2762, 2556, 2134,
         2237, 2460, 2436, 1920, 2661, 2255, 2400, 2531, 2359, 2200, 2240,
          984, 2125, 2043, 2724, 2416, 2407, 2652, 2364, 2113, 2348, 2300,
         1963, 2429, 2260, 2335, 2281, 2357, 2231, 2047, 2163, 2387, 2362,
         2168],
        [ 948,  968,  656,  626,  729,  627,  948,  880,  674,  453,  875,
          772,  580,  757,  183,  532,  754,  609,  478,  681,  840,  799,
          594,  879,  966,  285,  593,  632,  541,  645,  896,  661,  709,
         1038,  915,  749,  705,  758,  652,  775,  962,  846,  622,  678,
          841]], dtype=int64),
```

After the count matrix is calculated then the log probabilities must be done and the methods shown below will show the math behind it



```
In [10]:  def calculate_likelihood_probs(count_matrix,alpha,n_features):
              log_probabilities = []
              for i in range(n_features):
                  num = count_matrix[i] + alpha
                  den = num.sum(axis = 1).reshape(-1,1)
                  log_probability = np.log(num) - np.log(den)
                  log_probabilities.append(log_probability)
              return log_probabilities
```

```
In [11]:  def calculate_prior_probs(class_count):

              num = class_count
              den = class_count.sum()

              return np.log(num)-np.log(den)
```

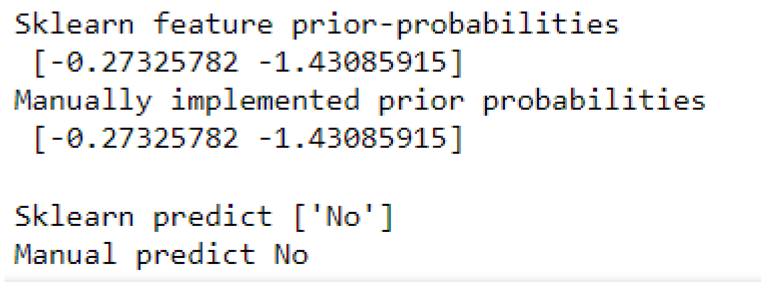
When the methods are used the log probabilities are computed and an example of what some of the log probabilities would look like is shown down below

```
[array([[-3.82429576, -3.90444978, -3.76424749, -3.76466686, -3.79533366, -3.75216149, -
3.82429576, -3.86287554, -3.61712957, -3.69461192, -3.87497997, -3.827864 , -3.73287884, -
3.74267884, -3.98060073, -3.65436889, -3.81985328, -3.7575613 , -3.70443711, -3.77478499, -
3.84453481, -3.82652442, -4.64856025, -3.87920433, -3.91853794, -3.63097818, -3.75091951, -
3.75465009, -3.65775554, -3.77266859, -3.88486473, -3.77945691, -3.8001028 , -3.9584634 , -
3.74555536, -3.81763942, -3.78500655, -3.80839436, -3.77563281, -3.83054857, -3.91658291, -
3.86148825, -3.76299042, -3.77351461, -3.85918038], [-3.52914596, -3.50829014, -3.89687074, -
3.94360821, -3.79151022, -3.94201459, -3.52914596, -3.60349713, -3.86984206, -4.26645756, -
3.60918866, -3.73427571, -4.019804 , -3.75387137, -5.169619 , -4.10603333, -3.757837 , -3.9710958
, -4.21285416, -3.8595251 , -3.64996309
```

From these log probabilities the algorithm in Python could predict future outcomes.

Predictions

Using Naïve Bayes Classification and the importation of Sklearn, Python tested to see if another set of prior probabilities were calculated based on a new set of x-values if it would accurately give a correct predicted y-value and this happened to be the result.



```
Sklearn feature prior-probabilities
 [-0.27325782 -1.43085915]
Manually implemented prior probabilities
 [-0.27325782 -1.43085915]

Sklearn predict ['No']
Manual predict No
```

As can be seen the algorithm predicted that it will rain based on a new set of values of the features that correspond to whether or not it rained and it accurately predicted it.

Conclusion

In conclusion, Naïve Bayes Classification is an algorithm used in machine learning which its algorithm is based upon Bayes Theorem and is used for the solutions of classification-based problems. Applications of this algorithm could be done in Python and this algorithm in Python was done on a rain dataset of Australia. When calculating a binary outcome based on probabilities and features Naïve Bayes Classification could be the go to solution due to its compatibility with text-based classification problems and does well in multi-class predictions compared to other supervised learning algorithms.

References

[1] https://www.javatpoint.com/machine-learning-naive-bayes-classifier

[2]https://www.simplilearn.com/tutorials/statistics-tutorial/what-is-probability-distribution#:~:text=A%20probability%20distribution%20is%20a,variable%20within%20a%20given%20range.

[3] https://www.kaggle.com/code/gautigadu091/categorical-naive-bayes-from-scratch-in-python/notebook

A Type of Data Model: Gaussian Mixture

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

Introduction

In Machine Learning there are two main types of learning: Supervised and Unsupervised. Supervised learning is when the data points and features associated with the data points are labelled while unsupervised learning is when any of data points and the features associated with any of the data points are unlabeled. An example of unsupervised learning would be clustering which tries to find common traits among clusters of points. Something like a Gaussian Mixture Model would be perfect for clustering and the main idea of a Gaussian Mixture Model is it contains several Gaussians (normal distributions) and the parameters of a Gaussian are a mean, covariance, and a mixing probability that determines how large the Gaussian function will be. If any of the parameters in the function are latent (non-observable) then an algorithm would be used in the Gaussian Mixture Model called the Expectation Maximization Algorithm. The beauty of the Gaussian Mixture Model algorithm is the value of the parameters throughout the model do not always have to be observable and from the Expectation Maximization Algorithm the values of parameters can be estimated and this will repeat throughout the algorithm until the best estimate is reached.

Expectation Maximization Algorithm

Maximum Likelihood Estimation

Maximum Likelihood Estimation is a method of estimating the parameters of a joint probability.

The formula for Maximum Likelihood Estimation is $P(Dc|\theta c)$ = $\prod_{x \varepsilon D_c}$ $P(x|\theta c)$ with Dc representing the data belonging to the cth class and θc representing the model parameter associated to the cth class. Now in order to find θc what must be done is finding the log likelihood of θc which is given by the formula $LL(\theta c)$ = $log\ P(Dc|\theta c)$ = $\sum_{x \varepsilon D_c} log P(x|\theta_c)$ and then the final step in order to find θc is the formula θc = $arg\theta cmaxLL(\theta c)$ which basically is finding the maximum θc out of all the θcs.

E-Step

This part of the algorithm is responsible for adjusting the parameters in each repetition of the algorithm and that is by solving the question of which Gaussian generated each datapoint. A distribution over all possibilities of probabilities must be done in order to solve the question of which Gaussian generated each datapoint and that is represented by

$\gamma_k^{(n)} = p(z^{(n)} = k|x^{(n)}; \pi, \mu, \sum)$ with each

$\gamma_k = p(z = k|x) = \frac{p(z = k)p(x|z = k)}{p(x)}$ and that represents a conditional probability using Bayes rule.

M-Step

In this part of the algorithm what is done is getting a certain amount of posterior probability for each Gaussian for each datapoint repeatedly. What must be satisfied is the derivative of log-likelihood must be 0 and the formula for log-likelihood is

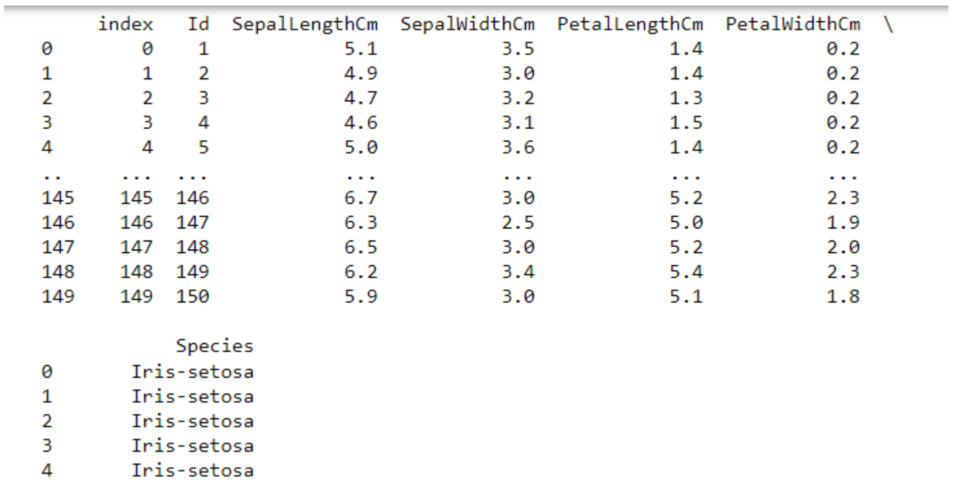
$$lnp(X|\pi, \mu, \sum) = \sum_{n=1}^{K} ln(\sum_{k=1}^{K} \pi_k N(x^{(n)}|\mu_k, \sum_k)$$

. From that, derived closed form updates happens to all the parameters.

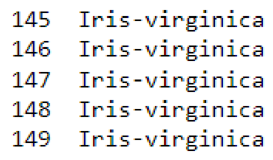
Gaussian Mixture Algorithm in Python

Iris Dataset

An iris dataset will be used in order to demonstrate the Gaussian Mixture Algorithm in Python. What this dataset has is features such as SepalLength, SepalWidth, PetalLength, PedalWidth, and three types of species: Iris-virginica, Iris-versicolor, and Iris-setosa. This is how a portion of the dataset looks like.



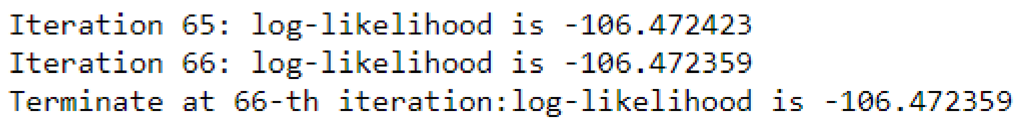
```
     index   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0        0    1            5.1           3.5            1.4           0.2
1        1    2            4.9           3.0            1.4           0.2
2        2    3            4.7           3.2            1.3           0.2
3        3    4            4.6           3.1            1.5           0.2
4        4    5            5.0           3.6            1.4           0.2
..     ...  ...            ...           ...            ...           ...
145    145  146            6.7           3.0            5.2           2.3
146    146  147            6.3           2.5            5.0           1.9
147    147  148            6.5           3.0            5.2           2.0
148    148  149            6.2           3.4            5.4           2.3
149    149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
```



```
145   Iris-virginica
146   Iris-virginica
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica
```

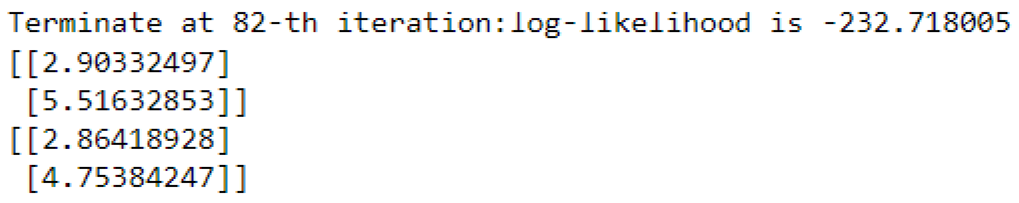
Iteration to Find Maximum Likelihood Estimation

The formula for finding the maximum likelihood was previously mentioned under the Maximum Likelihood Estimation sub section of the section Expectation Maximization Algorithm. This formula is incorporated into python and also what is done in python is repeat the formula for a parameter using the mean and covariance until the maximum log likelihood is reached and this is reached when the value of the maximum log likelihood doesn’t change in the current repetition.



```
Iteration 65: log-likelihood is -106.472423
Iteration 66: log-likelihood is -106.472359
Terminate at 66-th iteration:log-likelihood is -106.472359
```

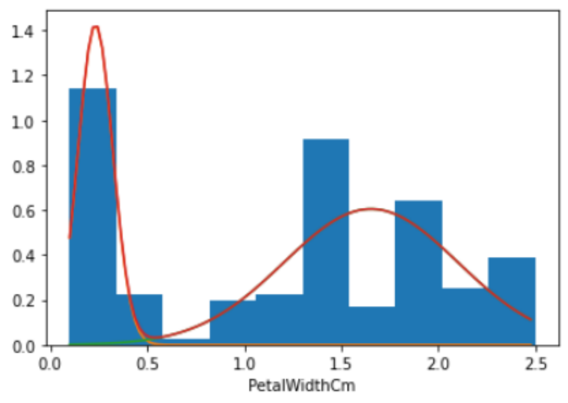
As seen above it took the algorithm 66 times in order to get the maximum log-likelihood. -106.472359 is the log-likelihood for the PetalWidthCm parameter. Now, the Maximum Likelihood Estimation must be done for SepalWidthCm and PetalLengthCM and for those parameters it takes 82 times in order to get the maximum log-likelihood.



```
Terminate at 82-th iteration:log-likelihood is -232.718005
[[2.90332497]
 [5.51632853]]
[[2.86418928]
 [4.75384247]]
```

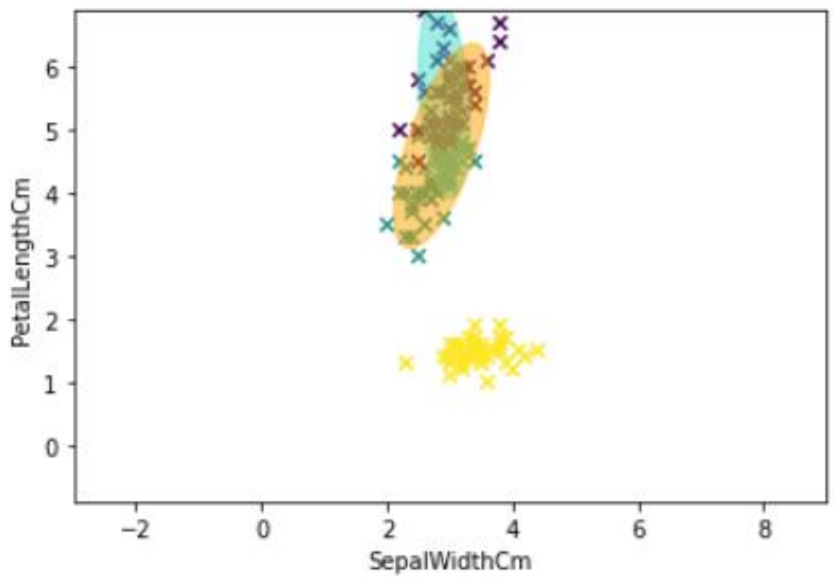
Gaussian Mixture Model Visualization

A.) PetalWidthCm



Estimated values for PetalWidthCm

B.) SepalWidthCm and PetalLengthCm



Estimated values for SepalWidthCm and PetalLengthCm

Conclusion

In conclusion, unsupervised learning is when any of data points and the features associated with any of the data points are unlabeled and an example of unsupervised learning is clustering. The Gaussian Mixture Model is an example of what clustering can undergo and this model was used on an iris dataset. Mathematical principles such as Maximum Likelihood Estimation, Bayes rule, and the Expectation Maximization Algorithm apply to the Gaussian Mixture Model in order to estimate parameters that aren't observable or also known as latent. With the Gaussian Mixture Model, every data point does not to be labeled and can learn in seemingly unorganized data.

References

[1]https://towardsdatascience.com/gaussian-mixture-models-explained 6986aaf5a95#:~:text=A%20Gaussian%20Mixture%20is%20a,%CE%A3%20that%20defines%20its%20width.

[2] https://github.com/cchangyou/Gaussian-Mixture-Model

[3] Bayesian Methods-Changyou Chen-Department of Computer Science and Engineering SUNY Buffalo, State University of New York