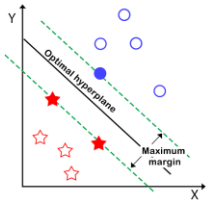
Support Vector Machine Experiment Report

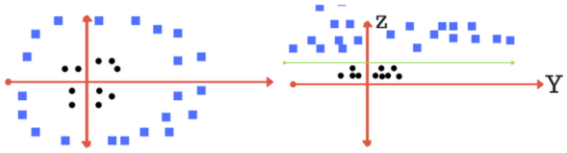
Nisarg Negi
Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
nisargne@buffalo.edu

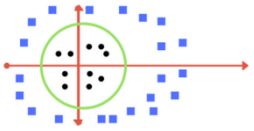
1. Introduction



Support Vector Machine (SVM) algorithm is a supervised machine learning algorithm used for regression classification analysis. In SVM, if we have a labelled dataset, data is divided into classes by drawing a hyper plane between them. SVM finds the maximum distance hyperplane from the datapoints, each side of the hyperplane forms 2 separate classes. In two-dimensional data, this hyperplane forms a line dividing the data into 2 different classes.

When we have more complicated data in which we cannot draw a separating line as in fig 1.2, we transform the data and add one more dimension say z-axis (Fig 1.3). We can observe that the points around the origin in xy plane will be clearly separated by z-axis.



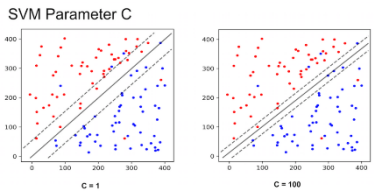


For the datapoints in z plane we can consider the points in a circle with the equation $w = x^2 + y^2$ and manipulate these points as a function of distance from the z axis. If we transform them back to XY plane, we will get a circular boundary as in figure 1.4.
These transforms are known as kernels.

2. Parameters

a. C Parameters

It controls the distance between the data points and the hyperplane.
-large C: distance is small, it can overfit
-small C: distance is large, it can underfit



b. Kernel

as stated before, SVM uses kernel trick to transform nonlinear data. There are 4 types of kernel which we can use:

Radial Basis Function Kernel	K(x,xi) = exp(-gamma * sum((x – xi^2))	Good local performance
Polynomial Kernel	K(x,xi) = 1 + sum(x * xi)^d	Global kernels
Sigmoid Kernel	K(x,xi) = tanh(αxi.xj +β)	Neural Net
Linear Kernel	K(x, xi) = sum(x * xi)	When sample is separable in low dimension space

c. Gamma Parameter

It defines how distant values from hyperplane must be considered in rbf, polynomial or sigmoid kernel.
-Low Gamma: far away points are considered
-High Gamma: only close points are considered

3. Dataset

We are using asteroseismology data which has the distinct features of oscillations of 6008 Kepler stars and using it to predict whether it is a Red Giant or a Helium burning star.

Feature:

POP: Target Variable, if 0: Red giant star & 1: Helium burning star
Dnu: Mean large frequency separation of modes with the same degree and consecutive order
Numax: Frequency of maximum oscillation power
Epsilon: Location of the l=0 mode

	POP	Dnu	numax	epsilon
0	1	4.44780	43.06289	0.985
1	0	6.94399	74.07646	0.150
2	1	2.64571	21.57891	0.855
3	1	4.24168	32.13189	0.840
4	0	10.44719	120.37356	0.275

4. Experiment

From the asteroseismology dataset we will use the features to predict whether the star is a Red Burning Giant or Helium Burning Star.

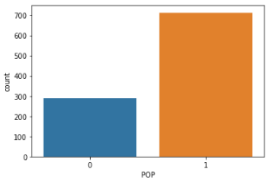
We will read the dataset from csv and store it into a data frame. Then we will perform our initial analysis of the data. We can see that all the features are integer or float so will not require any labeling or one hot encoding them.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001 entries, 0 to 1000
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   POP      1001 non-null   int64
 1   Dnu      1001 non-null   float64
 2   numax    1001 non-null   float64
 3   epsilon  1001 non-null   float64
dtypes: float64(3), int64(1)
memory usage: 31.4 KB
```

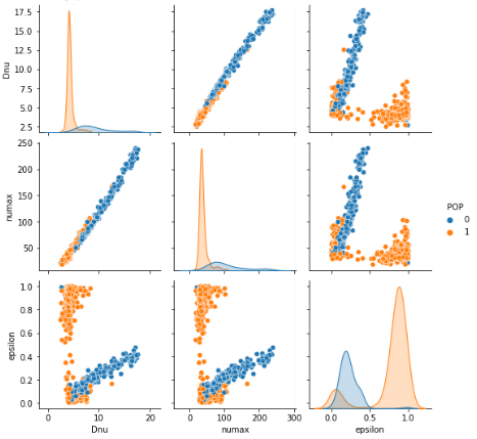
Then we will look for null values. We can tell that the data set does not have any null values and will not require imputation.

```
POP        0
Dnu        0
numax      0
epsilon    0
dtype: int64
```

We will perform bivariate analysis using pair plot. Pairplot helps us plot pairwise bivariate distributions in a dataset which helps us summaries the data visually. The x & y axis have all the features plotting against the target variable POP and the diagonal has the distribution of each element against the target variable POP.



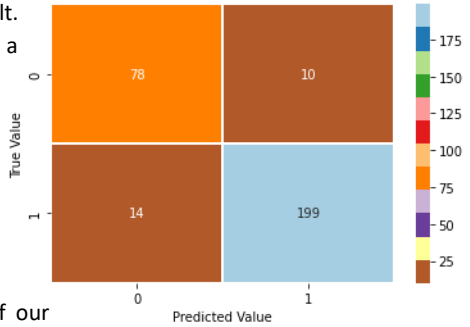
Then we will check for the count of each classification in the target variable POP to check if the data is not highly imbalanced. Our data is not highly imbalanced.



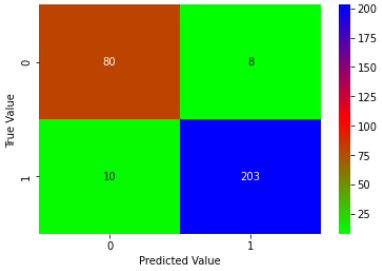
We will then plot the correlation for each variable in the dataset. This helps us find highly correlated features which we can remove from our model training as they will not affect our training by much.

We will then split our dataset into test(30%) and training(70%)

We will then fit our Training data into an SVM classifier and predict the result. We will compare this result with the actual values from the dataset by using a confusion matrix.



We are getting an accuracy of 92.03%. This accuracy is without any normalization and parameter tuning.
We can get better result



We will then normalize our data and check if our accuracy increases.
Normalizing the data increases our accuracy to 94.02%.
We can increase this further by parameter optimization.

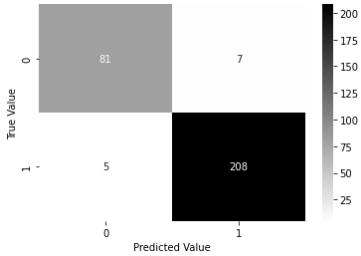
We will now perform parameter optimization by using grid search. For this will run SVM model prediction on different values of C, Gamma and Kernels.
C: 0.1, 1, 10, 100
Gamma: 1, 0.1, 0.01, 0.001
Kernel: rbf, polynomial, sigmoid, linear.
Using GridSearchCV we will run SVM for all the above parameters and select the best ones that fit.

```
print("Best Parameters: ",grid.best_params_)
```

```
Best Parameters:  {'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

We find out that the best parameters are C = 10, gamma = 1 and kernel being rbf.

Now, finally we will train our model using the best parameter that we found out and build our confusion matrix and test for the accuracy score.



	precision	recall	f1-score	support
0	0.94	0.92	0.93	88
1	0.97	0.98	0.97	213
accuracy			0.96	301
macro avg	0.95	0.95	0.95	301
weighted avg	0.96	0.96	0.96	301

We are now getting the accuracy of 96.01% which is greater than the previous conditions.
This tells us how using normalization and parameters optimization we can increase our accuracy.

4. References

1.https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

2.https://medium.com/@cdabakoglu/what-is-support-vector-machine-svm-fd0e9e39514f

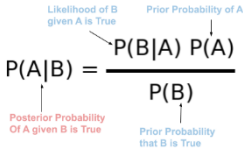
3. https://cdsarc.cds.unistra.fr/viz-bin/cat/J/MNRAS/469/4578#/article

Naïve Bayes Classifier Experiment Report

1. Introduction

Naïve Bayes Classifier is based on Bayes theorem, it is a series of simple probabilistic classifiers which use Bayes theorem. It is a classification algorithm. Classification algorithm divides the data into separate categories or classes.
The algorithm is called "Naïve" because it assumes that the features in the data have strong independence i.e. they are unrelated to other features in other classes. For e.g. In case of attrition in a cellphone operator, we assume that the customer not getting good service and the cost of the cellular plan affecting him to decide to switch are both independently contribute to the customer leaving.

Conditional probability: Chances of an event occurring given that another event has already occurred.
Bayes Theorem: Conditional probability of an event, given that another event has already occurred is equal to the probability of second event multiplied by probability of the first event.



Bayes rule can further be extended for more than 2 events as:

$$
\begin{aligned}
& p(y, X_1, X_2, \cdots, X_n) \\
& \propto p(y)p(X_1, X_2, \cdots, X_n|y) \\
& \propto p(y)p(X_1|y)p(X_2, \cdots, X_n|y, X_1) \\
& \propto p(y)p(X_1|y)p(X_2|y, X_1)p(X_3, \cdots, X_n|y, X_1, X_2) \\
& \cdots \\
& \propto p(y)p(X_1|y)p(X_2|y, X_1)p(X_3|y, X_1, X_2) \cdots p(X_n|y, X_1, X_2, \cdots, X_{n-1})
\end{aligned}
$$

Conditional Distribution: As events are assumed conditionally independent the distribution can be expressed as:

$$p(y|X_1, X_2, \cdots, X_n) = \frac{1}{Z} p(y) \prod_{i=1}^{n} p(X_i|y)$$

Since the characteristic variables are constant, z is dependent only on the features. It increases the stability of the model.

Classifier construction: We commonly select the maximum posterior probability decision criterion to build our classifier. We assume y has K categories, our equation will be:

$$\hat{y} = \underset{k \in [1, \cdots, K]}{\arg\max}\, p(y_k) \prod_{i=1}^{n} p(X_i|y_k)$$

2. Dataset

We will now use a dataset which contains the data of a social networking website users who have purchased a product by clicking on the add or not. It has the following features:

User ID: Unique id of the user
Gender: Gender of the user
Age: Age of the user
EstimatedSalary: Estimated salary of the user

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Purchased: 1 if purchased after seeing the advertisement else 0

3. Experiment

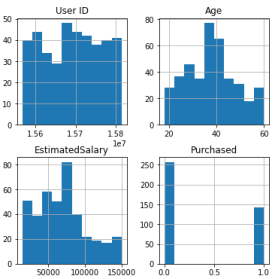
We will use the social networking website dataset to build a classification model using naïve bayes classifier to predict whether a user will purchase the product after clicking on the advertisement. This model can be used to target users and thereby reduce the costs associated with marketing.

We will read the dataset from csv and store it into a data frame. Then we will perform our initial analysis of the data. We can see that all the features are integer or float so will not require any labeling or one hot encoding them.

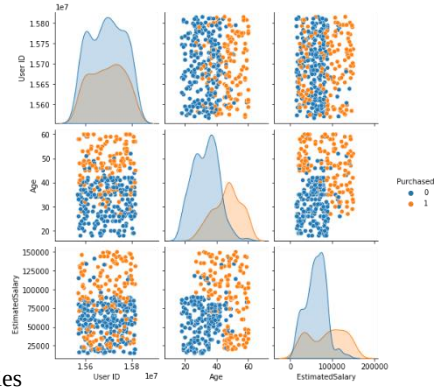
```
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

Then we will look for null values. We can tell that the data set does not have any null values and will not require imputation



We will then plot histograms of our dataset.

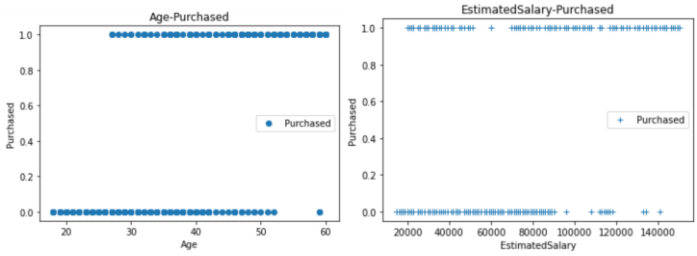


We will perform bivariate analysis using pair plot. Pairplot helps us plot pairwise bivariate distributions in a dataset which helps us summaries the data visually. The x & y axis have all the features plotting against the target variable Purchased and the diagonal has the distribution of each element against the target variable Purchased.

	User ID	Age	EstimatedSalary	Purchased
User ID	1.000000	-0.000721	0.071097	0.007120
Age	-0.000721	1.000000	0.155238	0.622454
EstimatedSalary	0.071097	0.155238	1.000000	0.362083
Purchased	0.007120	0.622454	0.362083	1.000000

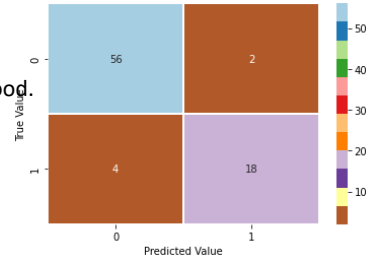
We will also perform correlation analysis. The correlation coefficient gives us the direction and degree of the relationship. Age and Purchased correlation are the same direction and have a good correlation. Age and User ID attributes are in inverse direction and have negligible correlation. Correlation is done to see in which direction the dependent variable will change when the independent variable changes.

We will then plot Age and Estimated salary against the target variable Purchased to see the distribution.



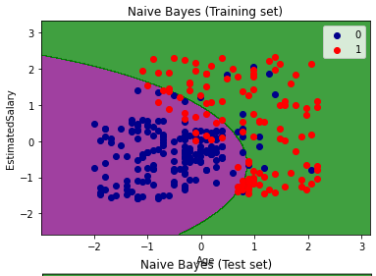
We can tell that both the features affect the user's decision to purchase a product.

We will store our features in X and target variable in Y and then split our dataset into test(25%) and training(75%).
The we will perform feature scaling and then fit our training data to the Naive Bayes classifier.
The we will run prediction on our test data and plot it into a confusion matrix:

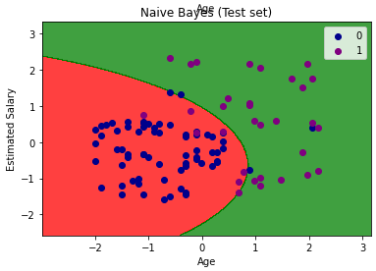


Visually from the confusion matrix we can tell that our model accuracy is pretty good.

Additionally, we can map our training set classification:



As well as our prediction classification:



From sklearn.metrics we can find the accuracy of our model as below:

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95        58
           1       0.90      0.82      0.86        22

    accuracy                           0.93        80
   macro avg       0.92      0.89      0.90        80
weighted avg       0.92      0.93      0.92        80

[[56  2]
 [ 4 18]]
ACC:  0.925
```

4. References

https://towardsdatascience.com/introduction-to-naive-bayes-classifier-f5c202c97f92
https://medium.com/analytics-vidhya/everything-you-need-to-know-about-na%C3%AFve-bayes-9a97cff1cba3

Neural Network Backpropagation Experiment Report

1. Introduction

Neural networks are algorithms inspired from how the human brain functions. It works on processing the data in a way similar to how neurons process our sensory observations in our brain. It takes in data and recognizes patterns, draws out references and gives out an output.

They are also called Artificial Neural Networks (ANN) as they perform functions like human brain neurons but are not natural. They are made to artificially mimic the functions of neuron. An ANN is made up of a large number of neurons which work together to solve a problem.

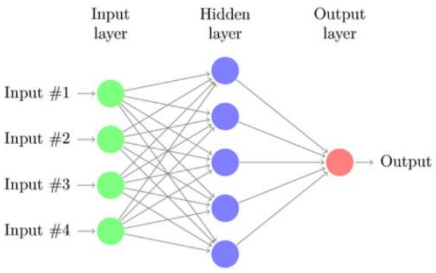
ANN learn by making observations like humans. The are configured by making them learn for various problems like classification, pattern recognition, Image recognition, etc. by using examples.

Layers: ANN consists of 3 types of layers usually:

Input unit: This layer takes raw input from the data.

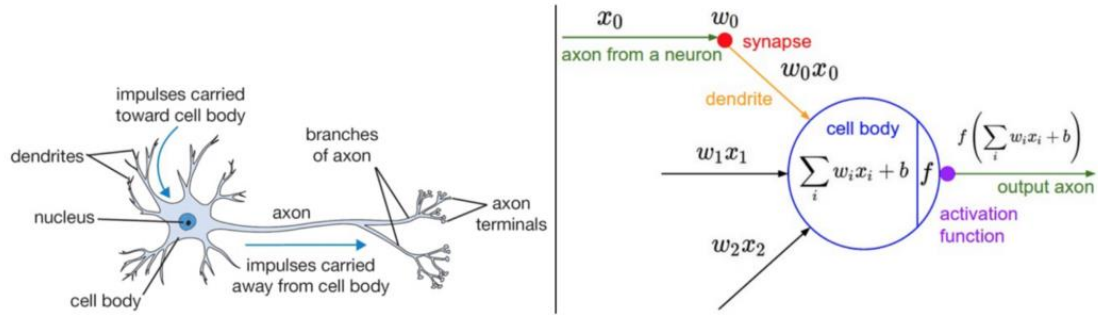
Hidden unit: All the processing happens in the hidden unit using the raw data from the input unit. The functioning depends on the input unit and the weights on the connection from the input unit.

Output unit: It functions depending on the problem statement, for eg for classification it will display the different classes. Its functioning depends on the hidden unit and the weights on the connection from the hidden unit.



In a simple neural network the hidden layers are free to create their own representation of data. The weights between the input and hidden layer determine when the hidden layer will be active.

Neuron: each hidden layer is made up of neurons. They are similar to neurons in Human and are also called nodes or units. The neuron receives an input, learns and computes from it and sends an output. Every input node has a weight(w) associated with it based on its importance. Hidden node applies a function(f) to the weighted sum of the inputs.



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

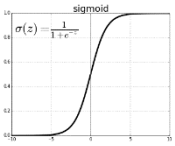
The above image compares a biological neuron with a computation neuron. x_1 and x_2 are inputs with weights w_1 & w_2 associated with the inputs. There is one more input 1 with a weight n associated with it.

Activation function:

The above neuron computes the output Y. The function used here, f, is a non linear function called activation function. The activation function is used to introduce non-linearity to the neuron output. This conversion is necessary as in reality as real data is not usually linear.

Each activation function takes a single input and performs mathematical operations. Some of the commonly used activation functions are:

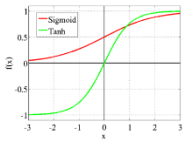
a. Sigmoid: It transforms the real valued input into between 0 & 1.
σ(x) = 1 / (1 + exp(−x))



b. Softmax: It is an activation function which transforms the outputs into probabilities which sum up to 1.It basically takes a real vector and transforms it into values between 0 & 1 such that the total sum is 1.
Probability (A) + Probability (B) = 1

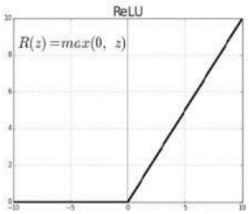
c. Tanh: It transforms a real values within a range of -1 & 1

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

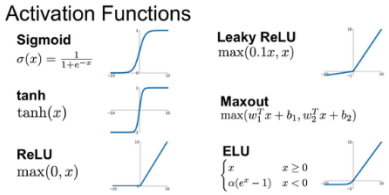


Both the sigmoid and Tanh activation functions with graphs compared

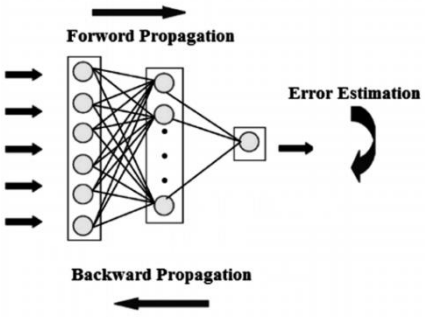
d. ReLu: Rectified Linear Unit takes in a real input and transforms it to a threshold at zero.



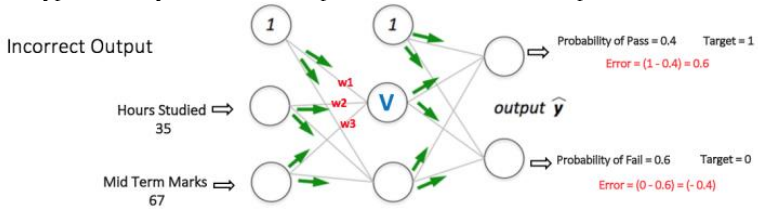
f(x) = max(o, x)



Every neuron has two methods of propagation:



Forward Propagation:
In this, the weights are randomly assigned.
Lets assume weights to be w1,w2,w3.
Input: x2,x3 say 35,67 hours of study
Target: [1,0] where 1 is pass and 0 is fail
The output(V) from the node have activation f calculated as:
V = f (1*w1 + x2*w2 + x3*w3)
= f (1*w1 + 35*w2 + 67*w3)
Output from other nodes in hidden layer are also similarly calculated. Two nodes with these calculations then feed to the output layer which helps us to calculate output of one node from two different hidden nodes.
Lets hypothetically make an assumption that two nodes in output are 0.4 & 0.6.



These values are far off from the target of 0 or 1. Therefore, the network formed in the above image is false. To correct this error we implement back propogation.

Backward Propagation:

After forward propagation node output is found out to be incorrect, errors are measured, and these errors are sent back to the hidden layer using back propagation to calculate gradients.
Then we adjust all weights using optimization techniques like Gradient descend to adjust the weights aiming to reduce the error in output unit/layer.
Formula's needed for back propagation algorithm:

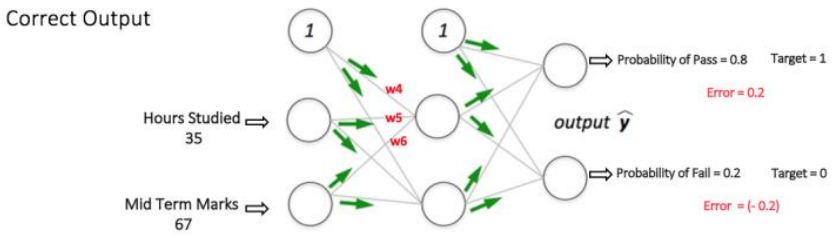
1.	$\frac{\partial E_d}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}$	For the partial derivatives
2.	$\delta_1^m = g_o'(a_1^m)\,(\hat{y}_d - y_d)$	For the final layer's error term
3.	$\delta_j^k = g'(a_j^k) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}$	For the hidden layers' error terms
4.	$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}(\hat{y}_d - y_d)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ij}^k}$	For combining the partial derivatives for each input-output pair
5.	$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}(\hat{y}_d - y_d)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ij}^k}$	For updating the weights

General Algorithm:
Step 1: Backward phase calculation:
Step a: For every input-output pair $(\vec{x_d}, y_d)$, store the calculated values in $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k while connecting node i in layer k-1 another node j in layer k proceeding from output layer to the layer 1.
Step b: calculate the error from the final layer δ_1^m by the second equation
Step c: backpropagate the error terms in the hidden layer δ_j^k, from final hidden layer k = m-1 and before, repeatedly using third equation.
Step d: Calculate partial derivatives of each error E_d w.r.t w^k_{ij} by the help of first equation.

Step 2: Combining individual gradient:
For every input-output $\frac{\partial E_d}{\partial w_{ij}^k}$ pair, compute total $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ gradient for all sets of input output pairs by using fourth equation.

Step 3: Updating the weights: Update the weights using the learning rate α and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ using the fifth equation.

From the example taken in forward propagation which gave us incorrect answer, we will now apply backward propagation. The weights will now be adjusted to minimize the error. As shown in the below image the output will now reduce to [0.2,-0.2] from [0.6,-0.4] previously found. This is closer to [1,0] and now our error has been reduced.



We will repeat this till forward and back propagation our output nodes come to [1,0].

2. Dataset

This is a dataset of electrical impulse measurement of freshly excised tissues samples of breast collected by NEB-Instituto de Engenharia Biomédica, Porto, Portugal. It has the following features:

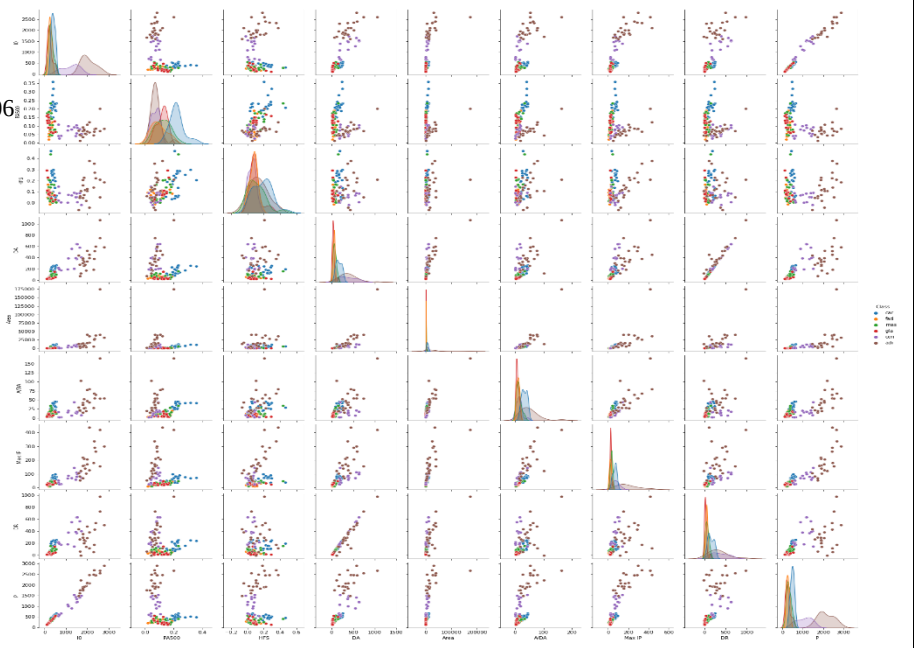
I0 Impedivity (ohm) at zero frequency
PA500 phase angle at 500 KHz
HFS high-frequency slope of phase angle
DA impedance distance between spectral ends
AREA area under spectrum
A/DA area normalized by DA
MAX IP maximum of the spectrum
DR distance between I0 and real part of the maximum frequency point
P length of the spectral curve Class(Classes:car(carcinoma), fad (fibro-adenoma), mas (mastopathy), gla (glandular), con (connective), adi (adipose))

3. Experiment

We will use the breast tissue dataset to classify the data into different classes using Neural net with the help of back propagation. The target variable here is class.

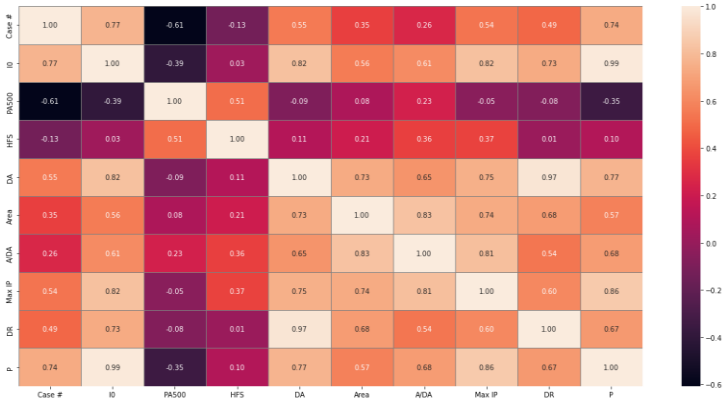
Case #	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	P	Class
77	650.0	0.041015	0.145211	216.811330	427.534068	1.971918	33.765163	214.165979	528.699233	con
31	272.0	0.091455	0.004887	63.789380	718.946310	11.270627	20.085556	60.690729	286.920220	fad
86	1800.0	0.034208	0.042586	301.060351	4406.154331	14.635452	67.625328	293.366920	1742.375702	adi
51	310.0	0.174707	0.165457	98.509961	2741.032044	27.824923	49.327862	85.270010	388.977808	mas
95	1800.0	0.069115	0.157080	385.564704	13831.724890	35.873940	157.570007	351.897477	1823.032364	adi

For the initial analysis we will look up for null or missing values in the dataset. We will move on to the analysis of the data. The data has 106 instances of electrical impedance measurements of freshly excised breast tissues. There are 9 features and 1 target class data column. There are a total of 6 classes.



From the pairplot we can analyse That our target variable P & IO are In a direct linear relationship.

From the correlation plot we can tell that IO is highly correlated with our target variable. We can also observe that DR & DA are highly correlated and hence one of them can be removed from our analysis as removing one of them will decrease complexity without much affect to the result.
Hence, the most optimum features we can use are IO, DA, A/DA, MAX IP, DR.



	Case #	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	P
count	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000
mean	53.500000	784.251618	0.120133	0.114691	190.568642	7335.155161	23.473784	75.381258	166.710575	810.638127
std	30.743563	753.950075	0.068596	0.101347	190.801448	18580.314212	23.354672	81.345838	181.309580	763.019135
min	1.000000	103.000000	0.012392	-0.066323	19.647670	70.426239	1.595742	7.968783	-9.257696	124.978561
25%	27.250000	250.000000	0.067413	0.043982	53.845470	409.647141	8.180321	26.893773	41.781258	270.215238
50%	53.500000	384.936489	0.105418	0.086568	120.777303	2219.581163	16.133657	44.216040	97.832557	454.108153
75%	79.750000	1487.989626	0.169602	0.166504	255.334809	7615.204968	30.953294	83.671755	232.990070	1301.559438
max	106.000000	2800.000000	0.358316	0.467748	1063.441427	174480.476200	164.071543	436.099640	977.552367	2896.582483

For more observations we can also use the describe function for data frame.

We will then normalize our dataset to stabilize the gradient descend which will allow us to use larger learning rate and also help the model to converge faster.

I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	P
0.019281	0.463673	0.346405	0.018483	0.001976	0.055790	0.041734	0.038905	0.021590
0.034112	0.366297	0.508497	0.017067	0.001479	0.044130	0.063193	0.022854	0.038826
0.088476	0.230071	0.262745	0.062652	0.007456	0.089389	0.049191	0.090385	0.093864
0.015202	0.312815	0.210458	0.000000	0.000000	0.012240	0.023736	0.017052	0.012771
0.042052	0.304743	0.252288	0.032527	0.001204	0.022382	0.034620	0.058543	0.032614

We will then labialize out target variable different classes from 'car','fad','mas','gla','con','adi' to 0,1,2,3,4,5 respectively. Then we shall remove the 'case#' variable split our data into test and training in the ration of 25% & 75% respectively.

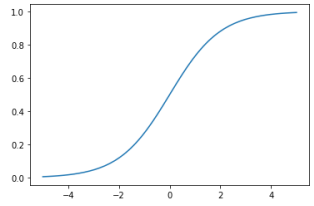
Next, we will build 6 vectors for our 6 classes in the target columns and create a vector 'target' for the results of the values in train data frame.
Next we need to implement the ANN model for which we choose
Input layer neuron: number of features for training = 9
Hidden layer neuron: 12 using the formula in the image
Output layer neuron: number of classes = 6

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.
N_o = number of output neurons.
N_s = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

We will assign random weights w1 & w2 initially using random function.

Next we will run forward propagation using the current random weights applying the sigmoid activation function discussed above. Sigmoid plot is shown in the figure.

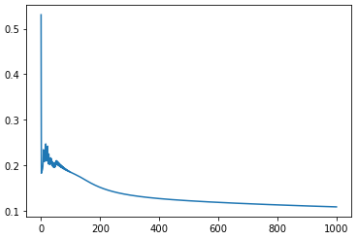


We will then train the model by using an alpha rate of 0.2 and epoch limit of 1000 and adding back propagation to the results from the forward propagation initialized values.
We can see that our error after applying back propagation is 0.108

```
print('Error:', er)
```

```
Error: 0.10883462678585987
```

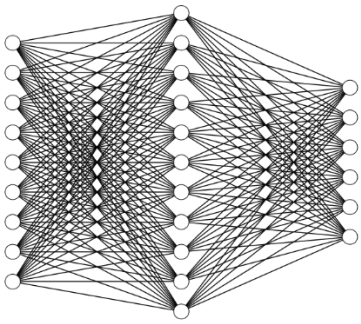
Plotting the error:



Next, we will run our prediction on the dataset and check for its accuracy

	Class	Prediction
4	car	car
6	car	car
8	car	car
15	car	car
25	fad	mas
28	fad	fad
29	fad	fad
30	fad	mas
31	fad	fad
35	fad	gla
36	mas	gla
37	mas	gla
38	mas	mas
46	mas	gla
47	mas	mas
49	mas	mas
50	mas	car
51	mas	car
53	mas	car
55	gla	gla
46	mas	gla
47	mas	mas
49	mas	mas
50	mas	car
51	mas	car
53	mas	car
55	gla	gla
56	gla	gla
58	gla	gla
59	gla	gla
64	gla	fad
67	gla	gla
72	con	con
74	con	con
78	con	con
81	con	con
87	adi	adi
97	adi	adi
98	adi	adi

```
Correct: 22 / 32 : 68.75 %
```



We can see that we have an accuracy of 68.75% buy using backpropogation algorithm on ANN with learning rate of 0.2 and 11 neurons on the hidden layer. Our simple ANN model can be represented as in the figure above with 9 Input nodes, 11 hidden nodes and 6 output nodes

4. References:

1. https://brilliant.org/wiki/backpropagation/#:~:text=Backpropagation%2C%20short%20for%20%22backward%20propagation,to%20the%20neural%20network%27s%20weights
2. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
3. https://towardsdatascience.com/a-step-by-step-guide-to-building-a-multiclass-classifier-for-breast-tissue-classification-5b685d765e97
4. https://purnasaigudikandula.medium.com/a-beginner-intro-to-neural-networks-543267bda3c8

Gaussian Mixture Model Experiment Report

1. Introduction

Gaussian Mixture Model is a type of clustering algorithm. It is a probabilistic model used on normally distributed clusters of data within a dataset. It doesn't require knowing the which cluster a data point belongs to for training, it learns this on its own and hence it is classified as an unsupervised clustering algorithm.
GMM is similar to k-means clustering, we can also say that K-means is a part of GMM.

Mixture model: mixture model is a combination of two different models. GMM is a mixture model of two Gaussian Distributions with their weights. The sum of weights

One dimensional GMM:

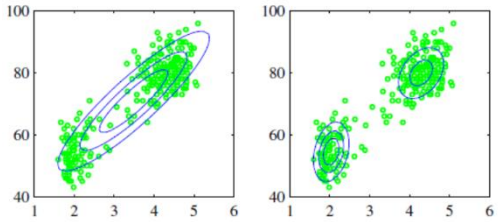
$$p(x) = \sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i)$$
$$\mathcal{N}(x \mid \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$
$$\sum_{i=1}^{K} \phi_i = 1$$

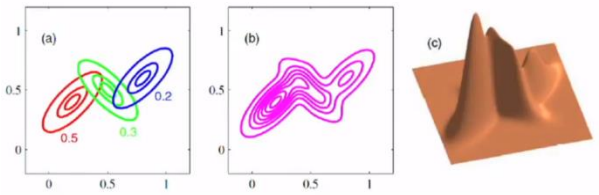
pi is the weight mixing coefficient of the model
Here the sum of pi should be 1 as probability cannot be more than 1.

Multi-dimensional GMM:

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$
$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^{\mathrm{T}} \Sigma_i^{-1} (\vec{x}-\vec{\mu}_i)\right)$$
$$\sum_{i=1}^{K} \phi_i = 1$$

When there are two or more Gaussian's in data, each of the gaussians have their own mean parameter and covariance. It can be seen in the figure.
In the figure on the left we can see that if we fit one gaussian, the distribution isn't proper around the cluster. We need two gaussians for the cluster to be dense around the mean.





We can also overlap gaussians in a GMM. The numbers in the graph on left are the weights. Image in the right shows density in a 3D graph.

Expectation Maximization: This is done to find maximum likelihood of a hidden variable.

Algorithm:

Step1: Initialize the variables and compute the log likelihood.

Step2: We find the current estimator parameter by evaluating the posterior distribution of Z

$$Z:\ p(Z|X, \vartheta^{(m-1)})$$

Step3: Using this distribution of Z we evaluate the complete likelihood.

$$\mathcal{Q}(\vartheta, \vartheta^{(m-1)}) = \sum_{Z} \underbrace{p(Z|X, \vartheta^{(m-1)})}_{\substack{\text{distribution of Z} \\ \text{assuming } \vartheta^{(m-1)}}} \underbrace{\log p(X, Z|\vartheta)}_{\substack{\text{complete data likelihood} \\ \text{unknown } \vartheta}}$$
$$= \mathbb{E}_{Z|X,\vartheta^{(m-1)}} \log p(X, Z|\vartheta)$$

Step4: Maximizing the Q function from above:

$$\vartheta^{(m)} = \text{argmax}_{\vartheta}\, \mathcal{Q}(\vartheta, \vartheta^{(m-1)})$$

Using previous guess, we computed the expected complete log likelihood

EM for GMM:
We assume that the hidden variables and latent variables (present in data) are equal. We need to find is the covariance, mixing coefficient and mean.

$$\text{E Step}\quad \mathcal{Q}(\vartheta, \vartheta^{(m-1)}) = \mathbb{E}_{Z|X,\vartheta^{(m-1)}} \log p(X, Z|\vartheta)$$

$$\text{M Step}\quad \vartheta^{(m)} = \text{argmax}_{\vartheta}\, \mathcal{Q}(\vartheta, \vartheta^{(m-1)})$$

We calculated the log likelihood of expected value above.

$$\begin{aligned}
\mathcal{Q}(\vartheta, \vartheta^{(m-1)}) &= \mathbb{E}_{Z|X,\vartheta^{(m-1)}} \log p(X, Z|\vartheta) \\
&= \mathbb{E}_{Z|X,\vartheta^{(m-1)}} \left[\sum_{n=1}^{N} \log p(x_n, z_n|\vartheta)\right] \\
&= \sum_{n=1}^{N} \mathbb{E}_{Z|X,\vartheta^{(m-1)}} [\log \prod_{k=1}^{K} (\pi_k p(x_n|\theta_k))^{\mathbb{I}(z_n=k)}] \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{E}_{Z|X,\vartheta^{(m-1)}} [\mathbb{I}(z_n = k)] \log (\pi_k p(x_n|\theta_k)) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} p(z_n = k|X, \vartheta^{(m-1)}) \log (\pi_k p(x_n|\theta_k)) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log (\pi_k p(x_n|\theta_k)) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log \pi_k + \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log p(x_n|\theta_k)
\end{aligned}$$

For M step we need to differentiate the equation based on the guessed parameter:

$$\begin{aligned}
\frac{\partial \mathcal{Q}}{\partial \mu_k} &= \frac{\partial}{\partial \mu_k}\left\{\sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log \pi_k + \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log p(x_n|\theta_k)\right\} \\
&= \frac{\partial}{\partial \mu_k}\left\{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log p(x_n|\theta_k)\right\}, \quad k = 1, \ldots, K \\
&= \frac{\partial}{\partial \mu_k}\left\{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}} \log \left[\frac{1}{(2\pi_k)^{p/2}} \frac{1}{|\Sigma_k|^{1/2}} \exp\{-\frac{1}{2}(x_n - \mu_k)^T \Sigma_k^{-1}(x_n - \mu_k)\}\right]\right\}
\end{aligned}$$

(Use $\frac{\partial}{\partial \mu_k}(x_n - \mu_k)^T \Sigma_k^{-1}(x_n - \mu_k) = -2\Sigma_k^{-1}(x_n - \mu_k)$ to simplify and equate to zero)

$\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}}(x_n - \mu_k) = 0 \implies$

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}} x_n}{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}}}, \quad k = 1, \ldots, K$$

Here the responsibility becomes constant:

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}} (x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}}} \quad k = 1, \ldots, K$$

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta^{(m-1)}}}{N}, \quad k = 1, \ldots, K$$

We will continue similarily for other parameters:

- Initialize v^0, calculate $l^0 = \log p(X|v^0)$
- $\gamma(z_{nk})_{|\vartheta^{(m-1)}} = p(Z|X, \vartheta^{(m-1)}) = \frac{\pi_k^{(m-1)} p(x_n|\theta_k^{(m-1)})}{\sum_{j=1}^{K} \pi_j^{(m-1)} p(x_n|\theta_j^{(m-1)})}$
- v^m:

- $\mu_k^m = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta(m-1)} x_n}{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta(m-1)}}$

- $\sum_k^{(m)} = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta(m-1)} (x_n - \mu_k^{(m-1)})(x_n - \mu_k^{(m-1)})^T}{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta(m-1)}}$

- $\pi_k^{(m)} = \frac{\sum_{n=1}^{N} \gamma(z_{nk})_{|\vartheta(m-1)}}{N}, \quad k = 1, \ldots, K$

- We look for convergence and stop if $l^m - l^{(m-1)} < \varepsilon$

AIC: Akaike's Information Criteria consists of log likelihood, we set it at maximum to find a good model and the number of parameters. We consider the number of parameters ad ignore a better log likelihood as increasing the number of parameter will increase the accuracy but will tend to over fit the data.
We need this number of parameter to use it to know how many gaussians we can fit in ou

$$\mathbf{AIC_i = -2LogL_i + 2P_i}$$

BIC: Bayesian Information Criteria is similar to AIC but instead of number of parameter, it has weights. We consider the best fit for number of cluster in a BIC curve as to the point after which the graph is relatively flat.

$$\mathbf{BIC_i = -2LogL_i + P_i.logn}$$

2. Dataset:

We will use a dry bean dataset which has observations from images of 7 different types of beans. This Multivariate dataset has been created by Murat KOKLU, Faculty of Technology, Selcuk University, TURKEY.
Attributes of the dataset:
1. Area (A): The area of a bean zone and the number of pixels within its boundaries.
2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4. Minor axis length (l): Longest line that can be drawn from the bean while standing perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10.Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11.Roundness (R): Calculated with the following formula: (4piA)/(P^2)
12.Compactness (CO): Measures the roundness of an object: Ed/L
13.ShapeFactor1 (SF1)
14.ShapeFactor2 (SF2)
15.ShapeFactor3 (SF3)
16.ShapeFactor4 (SF4)
17.Class (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira)

3. Experiment:

We will use the dry beans dataset to cluster the data into different classes of dry beans using Gaussian mixture model. We will load the csv data into a dataframe and then print some rows of the tables.

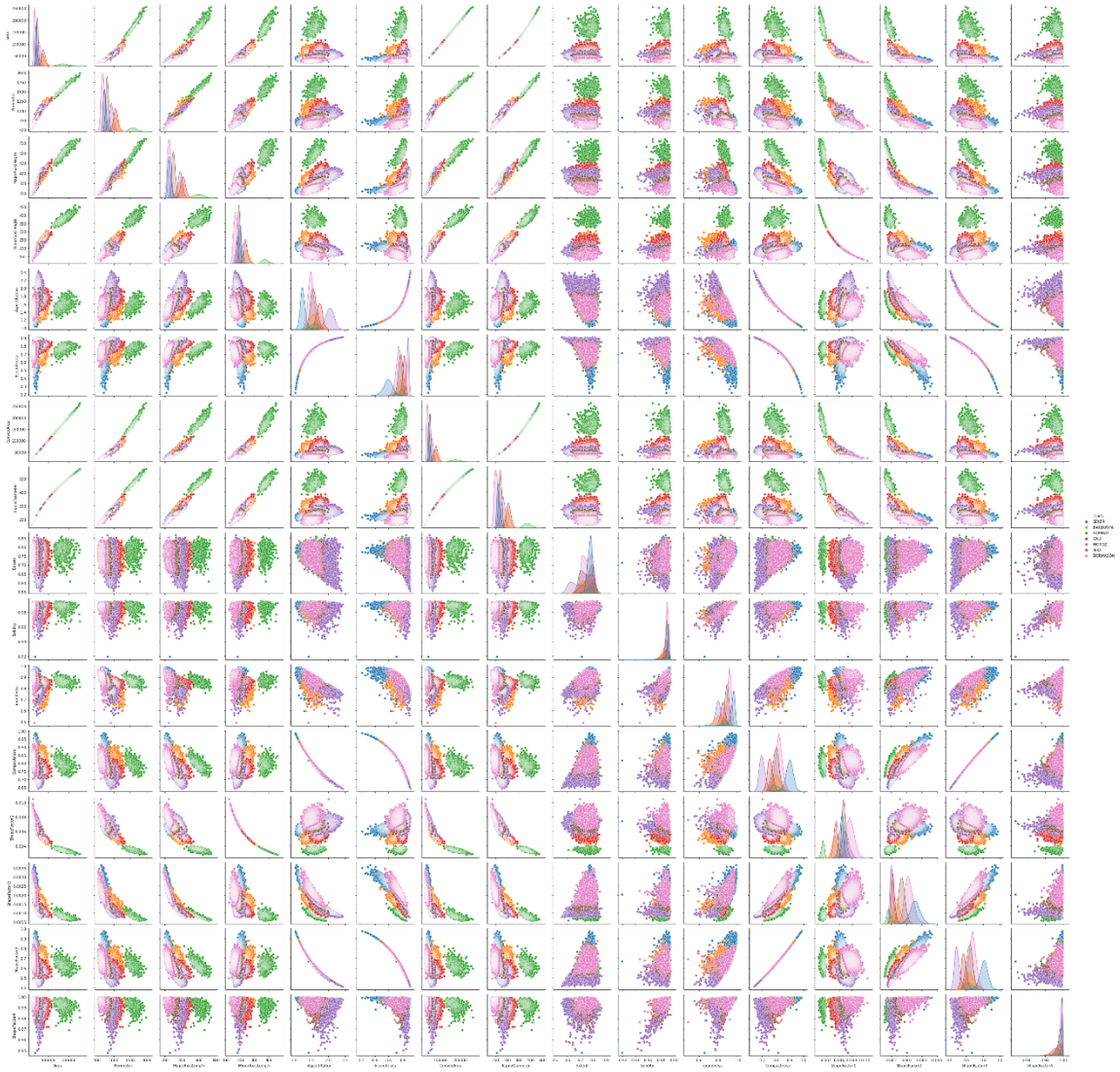
```
   Bean ID   Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  \
0        1  28395    610.291       208.178117       173.888747      1.197191
1        2  28734    638.018       200.524796       182.734419      1.097356

   Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  roundness  \
0      0.549812       28715     190.141097  0.763923  0.988856   0.958027
1      0.411785       29172     191.272751  0.783968  0.984986   0.887034

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4  Class
0     0.913358      0.007332      0.003147      0.834222      0.998724  SEKER
1     0.953861      0.006979      0.003564      0.909851      0.998430  SEKER
```

For the initial analysis we will lookup for null or missing values in the dataset.

```
Bean ID           0
Area              0
Perimeter         0
MajorAxisLength   0
MinorAxisLength   0
AspectRation      0
Eccentricity      0
ConvexArea        0
EquivDiameter     0
Extent            0
Solidity          0
roundness         0
Compactness       0
ShapeFactor1      0
ShapeFactor2      0
ShapeFactor3      0
ShapeFactor4      0
```

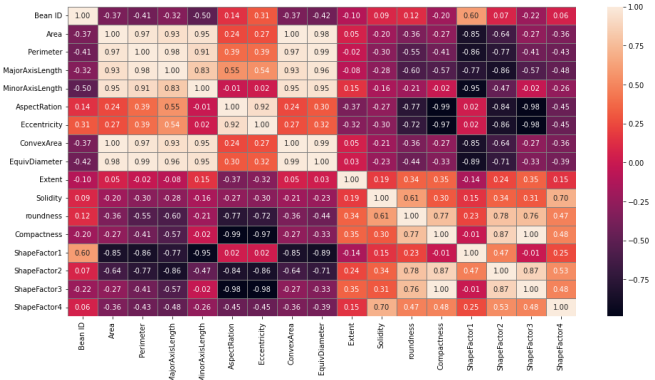
We will move on to the analysis of the data. The data has 13611 instances observation of dry beans. There are 17 features and 1 target class data column i.e. Class. There are a total of 7 classes.

Building pair plot to observe relationship of features:



From the pairplot we can analyse that our target variable most of the seed dimension fields are in a linear relationship.

We will drop the 'Bean ID' column. The we will build a correlation matrix.
Highly correlated features(we can ignore one from the pair):
-Area & Perimeter
-Area & EquivDiameter
-Perimeter & MajorAxisLenght
-Perimeter & EquivDiameter
-Eccentrivity & AspectRation
removing one of them from each pair will decrease complexity without much affect to the result.



We will then normalize our dataset to bring the numeric columns to a common scale:

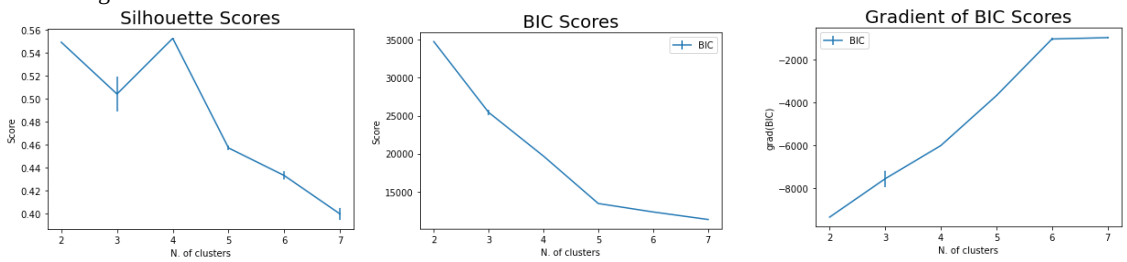
```
          0         1         2         3         4         5         6   \
0 -0.303995 -0.147572 -0.200680 -0.229340 -0.110783 -0.274705 -0.383647
1 -0.243312 -0.116507 -0.142463 -0.196135 -0.061043 -0.276769 -0.517914

          7         8         9         10        11        12        13  \
0 -0.147695 -0.186642  0.050742  0.064525  0.249923  0.322810  0.119495
1 -0.116073 -0.146720  0.098000 -0.065042  0.032465  0.350628  0.051702

         14        15        16
0  0.421640  0.338011  0.147154
1  0.435697  0.377922  0.108350
```

Note: We may or may not standardize the data for GMM model as the optimization parameter can learn and fit variance.

We then convert the normalized data to a data frame and perform PCA to reduce dimensionality. We will select the components which give us maximum explanation of the variance. Here 2 PCA explains enough data so we select it.

	P1	P2
0	-0.710019	0.543729
1	-0.604036	0.636805

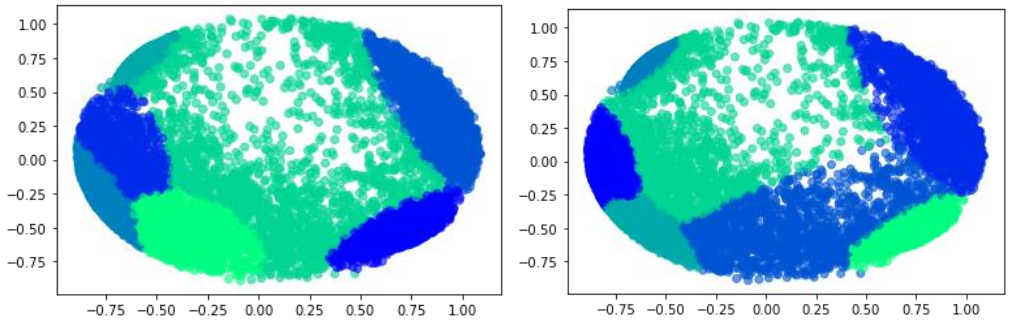
For finding the best fit cluster size we will use Silhouette score and BIC test:



From Silhouette: The highest peak is the best cluster size, i.e. 4
From BIC plot the best cluster size is the lowest, so the best number of cluster is 7.

We will initialize a GMM model with the data and the parameter n = 4 and n =7 for 4 & 7 clusters respectively. As GMM does not on its own find the number of clusters, we have to always use other tests like silhouette, AIC/BIC to learn the best number of clusters.

We can now visualize our GMM clusters on the data:



K=4 (clusters) K=7 (clusters)

4. References:

1. https://medium.com/swlh/ml-gmm-em-algorithm-647cf373cd5a
2. https://data.world/makeovermonday/2021w14/workspace/file?filename=Dry_Bean_Dataset.txt
3. https://ryanwingate.com/intro-to-machine-learning/unsupervised/gaussian-mixture-model-examples/