Hidden Markov Models Experiment Report

Nisarg Negi

Department of Computer Science and Engineering

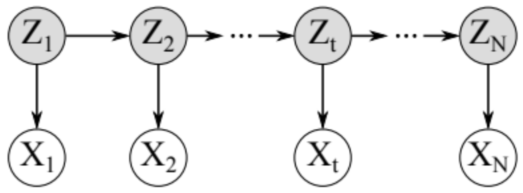
University at Buffalo, Buffalo, NY 14260

nisargne@buffalo.edu

1. Introduction

Hidden Markov model(HMM) is a discrete sequential data modelling method which works on the assumption that the observed events depend on a trait which are not directly observable. This is where the hidden part of the model comes from.
As for the Markov part, it comes from how we model the hidden states. We consider the Markov property from probability theory which is also known as the memoryless, i.e. the next hidden states only depends on the current state and is not affected by any other states.
HMM's are applied in speech recognition, activity recognition from video, gesture tracking, gene tracing.

HMM is used as a tool to represent probability distributions of sequential data. In HMM we take a stochastic process which has an observation Xt at a time t, here we have an unknown hidden state Zt. We assume that Zt satisfies the Markov property and Zt at time t is hence only dependent on the previous state Zt-1 at time t-1. This is the first order of Markov model. Hence, nth Markov model would depend on the previous n states.
Below is an image of a Bayesian network representing the first order HMM. Shaded states are the hidden states.



Hence, Joint distribution of a sequence of states and order for first order HMM:

$$P(X_{1:N}, Z_{1:N}) = P(Z_1) \prod_{t=2}^{N} P(Z_t|Z_{t-1}) \prod_{t=1}^{N} P(X_t|Z_t)$$

2. Dependence

HMM is characterized by the below five elements:

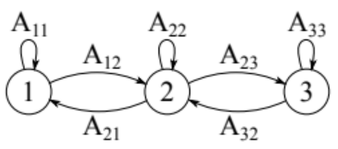
a. Number of states(N): The number of states that a hidden Markov model has is also related to the problem being modeled. We will encode the state Zt at time t as a K x 1 vector of binary numbers.

b. Number of Distinct observations(Ω): Observations are the number of disctinct observations in the experiment. We will encode Xt at a time t as Ω x 1 vector of binary numbers, where the only non-zero element is the only lth element.

c. State transition model(A): This is a N x N matrix with elements Aij containing the probability of transitioning from Zt-1,i to Zt,j in one time steps. It is written as:

$$A_{ij} = P(Z_{t,j} = 1|Z_{t-1,i} = 1)$$

Row sum of each row in the State transition matrix sums to 1, hence it is a stochastic matrix.



Above is a 3 state stochastic matrix for which $\sum_j A_{ij} = 1$
If in one jump a state can be reached from another state the value of A_{ij} will be > 0 else it will be zero. Hence, the state transition model for the above diagram will be

$$\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix}$$

Conditional probability can be written as

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{K}\prod_{j=1}^{K} A_{ij}^{Z_{t-1,i}Z_{t,j}}$$

We can take its log and write it as :

$$\begin{aligned} logP(Z_t|Z_{t-1}) &= \sum_{i=1}^{K}\sum_{j=1}^{K} Z_{t-1,i}Z_{t,j}\log A_{ij} \\ &= Z_t^\top \log(\mathbf{A})Z_{t-1}. \end{aligned}$$

d. Observation model(B): It is a matrix that describes the emission probabilities of the observable process(with elements $X_{t,k}$. Elements of this matrix are given as $B_{k,j}$ at state $Z_{t,j}$, and the matrix B is calculates as B = Ω× K.

Elements of B i.e. $B_{k,j}$ can be written as: $B_{kj} = P(X_t = k|Z_t = j)$
Conditional probabilities can be written as:

$$P(X_t|Z_t) = \prod_{j=1}^{K}\prod_{k=1}^{\Omega} B_{kj}^{Z_{t,j}X_{t,k}}$$

We can rake its log and write it as:

$$\begin{aligned} logP(X_t|Z_t) &= \sum_{j=1}^{K}\sum_{k=1}^{\Omega} Z_{t,j}X_{t,k}\log B_{kj} \\ &= X_t^\top \log(\mathbf{B})Z_t. \end{aligned}$$

e. Initial state distribution(π): Initial distribution of the states(π) is used for modeling where $\pi_i = P(Z_{1i=1}|\ \pi) = \prod_{i=1}^{K}\pi_i^{Z_{1i}}$

These 5 parameters can be used to specify the HMM model as, λ = (A, B, π).

3. Three classes of Problems that can be solved using HMM

a. Known a set of observations Ω and 3 model parameters $\boldsymbol{\pi}$, A and B, we can find out the probability of occurrence of X. We can find this by using forward filtering.

b. Known a set of observations Ω and 3 model parameters $\boldsymbol{\pi}$, A and B, we can find out the optimal set of hidden states Z that result in X. We can find this by using Viterbi algorithm.

c. Known a set of observations Ω we can find out the optimal parameters $\boldsymbol{\pi}$, A and B using Baum-Welch algorithm.

4. Algorithms

a. Forward Backward Algorithm: Forward-backward algorithm is a dynamic programming algorithm which uses belief propagation. It computes the filtered and smoothed marginals that are used to perform inference, sequence classification, MAP estimation, anomaly detection and clustering.

Algorithm 1 Forward algorithm

```
1: Input: $\boldsymbol{A}$, $\psi_{1:N}$, $\boldsymbol{\pi}$
2: $[\boldsymbol{\alpha}_1, C_1]$ = normalize($\psi_1 \odot \boldsymbol{\pi}$) ;
3: for $t = 2 : N$ do
4:     $[\boldsymbol{\alpha}_t, C_t]$ = normalize($\psi_t \odot (\boldsymbol{A}^\top \boldsymbol{\alpha}_{t-1})$) ;
5: Return $\boldsymbol{\alpha}_{1:N}$ and $\log P(X_{1:N}) = \sum_t \log C_t$
6: Sub: $[\boldsymbol{\alpha}, C]$ = normalize($\boldsymbol{u}$): $C = \sum_j u_j$; $\alpha_j = u_j/C$;
```

Algorithm 2 Backward algorithm

```
1: Input: $\boldsymbol{A}$, $\psi_{1:N}$, $\boldsymbol{\alpha}$
2: $\beta_N = 1$;
3: for $t = N-1 : 1$ do
4:     $\beta_t$ = normalize($\boldsymbol{A}(\psi_{t+1} \odot \beta_{t+1})$) ;
5: $\boldsymbol{\gamma}$ = normalize($\boldsymbol{\alpha} \odot \boldsymbol{\beta}$, 1)
6: Return $\boldsymbol{\gamma}_{1:N}$
```

b. Viterbi Algorithm: To compute the set of most probable sequence of hidden states(problem b) we will use the Viterbi algorithm. It uses the trellis diagram of HMM to compute the shortest path. Trellis diagram connects each state of the next time step in the model. Next, we use the forward-backward algorithm but using max-product instead of sum-product algorithm.

Algorithm 3 Viterbi algorithm

```
1: Input: $X_{1:N}$, $K$, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\pi}$
2: Initialize: $\delta_1 = \boldsymbol{\pi} \odot B_{X_1}$, $a_1 = \mathbf{0}$;
3: for $t = 2 : N$ do
4:     for $j = 1 : K$ do
5:         $[a_t(j),\ \delta_t(j)] = \max_i(\log \delta_{t-1}(:) + \log \mathbf{A}_{ij} + \log B_{X_t}(j))$;
6: $Z_N^* = \arg\max(\delta_N)$;
7: for $t = N - 1 : 1$ do
8:     $Z_t^* = a_{t+1} Z_{t+1}^*$;
9: Return $Z_{1:N}^*$
```

c. Baum-Welch Algorithm: Baum-Welch Algorithm is a dynamic programming algorithm which uses Expectation Maximization algorithm to tune parameters for HMM. It is used to find optimized values of A, B & $\boldsymbol{\pi}$ such that the model data overlaps with the actual data.

Algorithm 4 Baum-Welch algorithm

```
1: Input: $X_{1:N}$, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$
2: for $t = 1 : N$ do
3:     $\gamma(:,t) = (\alpha(:,t) \odot \beta(:,t))./sum(\alpha(:,t) \odot \beta(:,t))$;
4:     $\xi(:,:,t) = ((\alpha(:,t) \odot A(t+1)) * (\beta(:,t+1) \odot B(X_{t+1}))^T)./sum(\alpha(:,t) \odot \beta(:,t))$;
5: $\hat{\pi} = \gamma(:,1)./sum(\gamma(:,1))$;
6: for $j = 1 : K$ do
7:     $\hat{A}(j,:) = sum(\xi(2:N,j,:),1)./sum(sum(\xi(2:N,j,:),1),2)$;
8:     $\hat{B}(j,:) = \left(X(:,j)^T \boldsymbol{\gamma}\right)./sum(\boldsymbol{\gamma},1)$;
9: Return $\hat{\pi}$, $\hat{A}$, $\hat{B}$
```

5. Limitation:

HMM model considers only the last state. If we want to consider the last 2 states the training time grows exponentially considering that we have to build a set of cartesian multiplication of all states. Hence, our time complexity will become N^2 for training. For this we can use RNN as it can use continuous states.

6. Dataset:

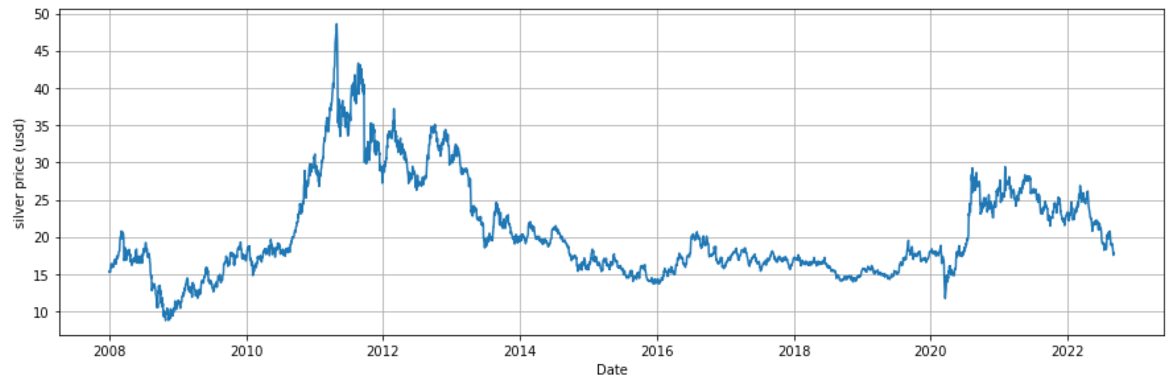
We will use the silver dataset from Kaggle. It has the data of daily silver price changes from January 2000 to September 2022.
Feature:
Date: Date of data instance
Open: Opening price of Silver per ounce in US Dollar
High: Highest price of Silver per ounce in US Dollar
Low: Lowest price of Silver per ounce in US Dollar
Close: Closing price of Silver per ounce in US Dollar
Volume: Ounces of Silver sold
Currency: Currency of transaction

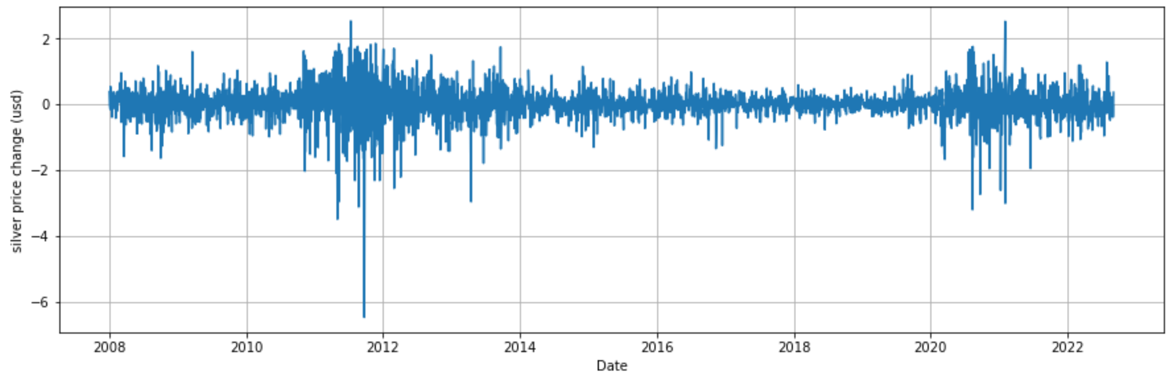
7. Experiment

We will analyze historical silver prices using hmmlearn, downloaded from: https://www.kaggle.com/datasets/psycon/daily-silver-price-historical-data.

We will import the necessary libraries and also import hmmlearn. We will calculate daily change in the silver prices as it is a better way of modelling stock like data which changes daily. We will consider data after 2008, ignoring the 2008 financial crisis.

We will plot the silver price and change in silver price from 2008.





Then, we will use a Gaussian model and fit the daily silver price change with 3 hidden states. We select 3 hidden states as we expect 3 different variety of data, namely high, medium low.

Fitting will result in three unique hidden states

```
print("Unique states:")
print(states)
```

```
Unique states:
[1 0 2]
```

We see that for this our data, the model starts mostly with state 1.

```
print("\nStart probabilities:")
print(model.startprob_)
```

```
Start probabilities:
[4.52687595e-27 1.00000000e+00 3.89759688e-60]
```

Next we look at our Gaussian Mean:

```
print("\nGaussian distribution means:")
print(model.means_)
```

```
Gaussian distribution means:
[[ 0.02767939]
 [-0.01342152]
 [-0.01316999]]
```

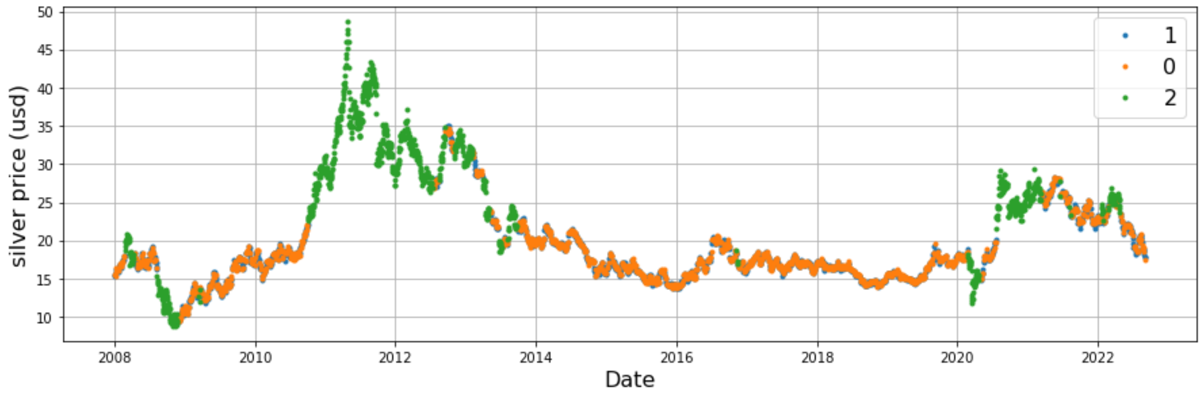
For state 1, the mean in 0.027 for state 2 our mean is -0.013, for state 3 our mean in -0.013. State 0 & 2 have similar mean; hence our model might not be good at representing the data.
Next we will look at the Gaussian covariance:

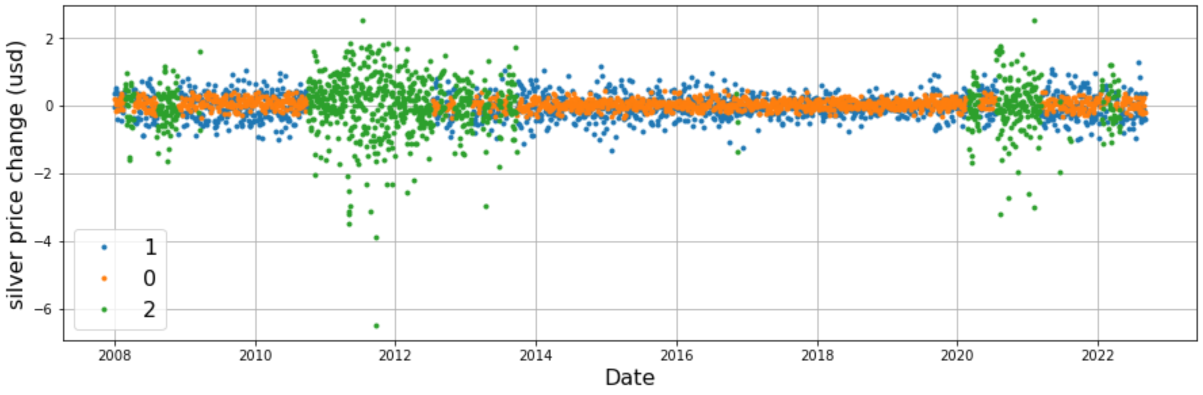
```
print("\nGaussian distribution covariances:")
print(model.covars_)
```

```
Gaussian distribution covariances:
[[[0.02602696]]

 [[0.1270092 ]]

 [[0.79569417]]]
```

As our data is one dimensional, our covariance matrix will be scalar(not vector).
For state 1 the covariance is 0.026, for state 0 the covariance is 0.127 and for state 2 the covariance 0.795. From this we can tell that for state 1 the volatility is low and for state 2 the volatility is high.

Next we will plot our models prediction :





We can tell that states 1,0,2 represent low, medium and high volatility.
From our plot we can see, the period of high volatility due to recession during 2011-2012 and the high volatility during 2020-2021 due to covid crisis.
We can also tell that the cost of silver increases during these highly volatile periods as investors purchase commodities instead of equity when the markets collapse.

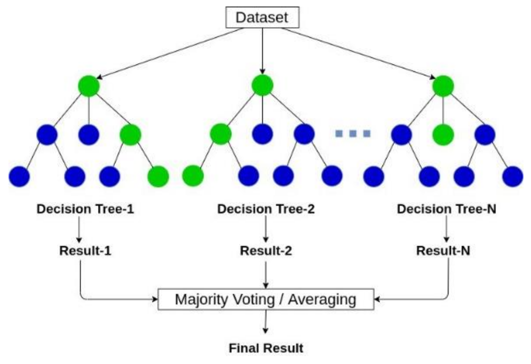
8. **References:**

https://www.kaggle.com/datasets/psycon/daily-silver-price-historical-data
https://scholar.harvard.edu/files/adegirmenci/files/hmm_adegirmenci_2014.pdf
https://medium.com/@natsunoyuki/hidden-markov-models-with-python-c026f778dfa7
https://medium.com/analytics-vidhya/baum-welch-algorithm-for-training-a-hidden-markov-model-part-2-of-the-hmm-series-d0e393b4fb86

Random Forest Classifier Models Experiment Report

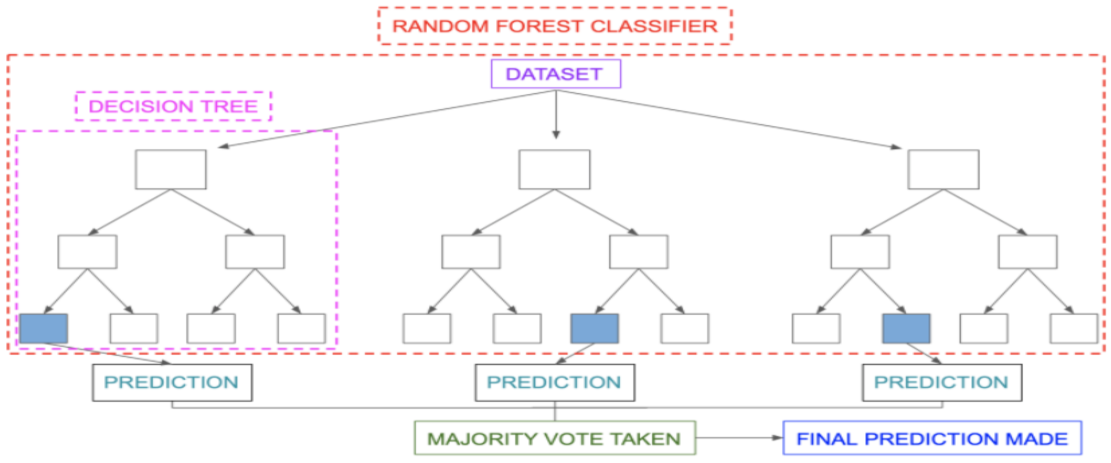
1. Introduction

Random Forest algorithm is a type of supervised machine learning algorithm. It has two variations: Classifier and Regressor.
Random forest Classifier uses ensemble learning i.e. many algorithms are run or one algorithm is run multiple times to create several decision trees from random subsets of the data. It uses the average from these decision tree predictions to predict the final class of the test object. This selection from several decision trees is known as voting.
As Random Forest algorithm uses several random decision trees, it is known as Random Forest algorithm.



2. Algorithm

Random Forest algorithm works in the following steps:
Starts by selecting random samples from the data
It constructions decision trees for all the samples and gets prediction results from all the samples.
It performs voting for each predicted results.
It ends by selecting the most voted prediction result.



If we use a high depth decision tree we will get low bias and high variance, which means that our model will be overfitted and predictions of unknown data will not be accurate. To solve this problem we use Random forest by combining many decision trees instead of one decision tree.

Gini Index:
While splitting the data into decision trees, the algorithm uses the Gini index to get a pure split. A pure split is a split in which one node has one type of categorical data and another has a different.

$$\boxed{\begin{aligned} Gini\ Index &= 1 - \sum_{i=1}^{n}\left(P_i\right)^2 \\ &= 1 - [(P_+)^2 + (P_-)^2] \end{aligned}}$$

Here, P+ is the probability of a positive class and
P_ is the probability of a negative class.

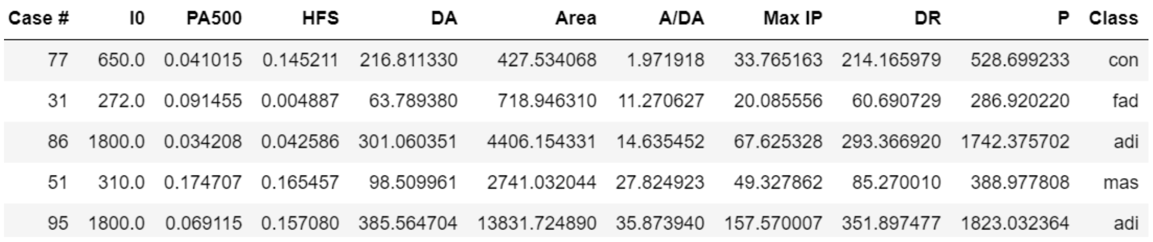
We, choose the decision tree feature to split which gives us the minimum Gini index(impurity) for the root.

Feature Selection: We can perform an initial run of the model to get the important features that we should use for the main model.

$$fi_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k \in all\ nodes} ni_k}$$

3. Experiment

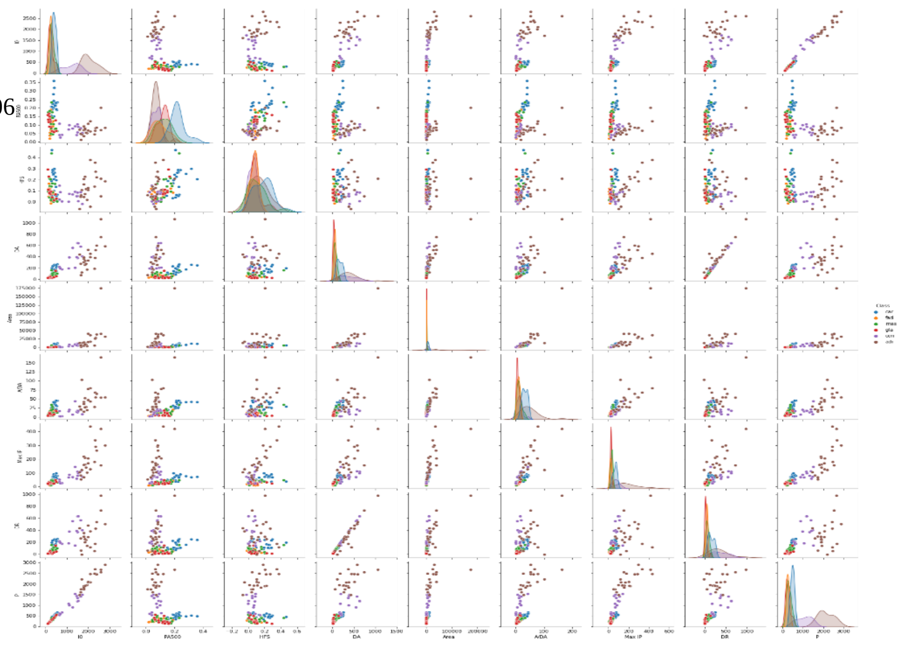
We will use the breast tissue dataset to classify the data into different classes using Random forest classifier. The target variable here is class.



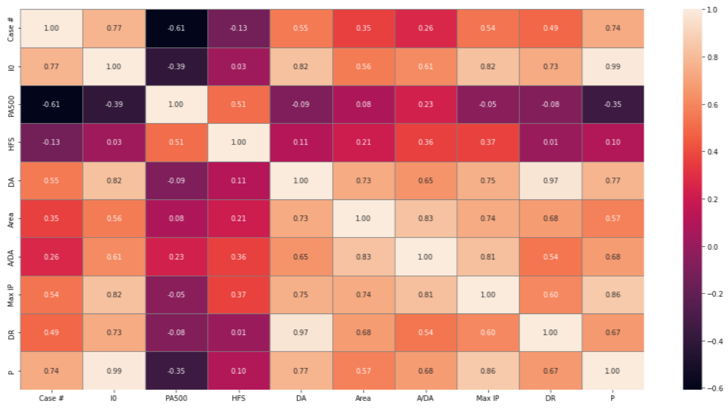
Case #	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	P	Class
77	650.0	0.041015	0.145211	216.811330	427.534068	1.971918	33.765163	214.165979	528.699233	con
31	272.0	0.091455	0.004887	63.789380	718.946310	11.270627	20.085556	60.690729	286.920220	fad
86	1800.0	0.034208	0.042586	301.060351	4406.154331	14.635452	67.625328	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.

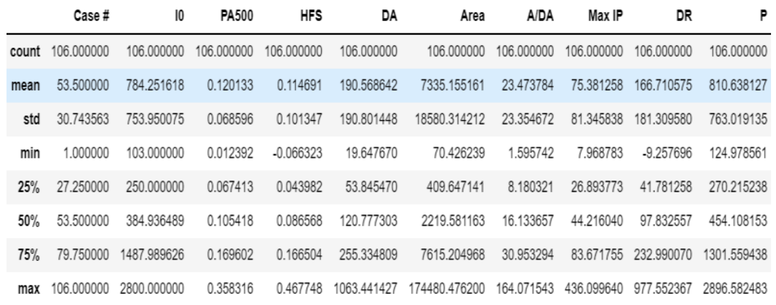
From the pairplot we can analyse That our target variable P & IO are In a direct linear relationship.



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
Case #	1.00	0.77	-0.61	-0.13	0.55	0.35	0.26	0.54	0.49	0.74
I0	0.77	1.00	-0.39	0.03	0.82	0.56	0.61	0.82	0.73	0.99
PA500	-0.61	-0.39	1.00	0.51	-0.09	0.08	0.23	-0.05	-0.08	-0.35
HFS	-0.13	0.03	0.51	1.00	0.11	0.21	0.36	0.37	0.01	0.10
DA	0.55	0.82	-0.09	0.11	1.00	0.73	0.65	0.75	0.97	0.77
Area	0.35	0.56	0.08	0.21	0.73	1.00	0.83	0.74	0.68	0.57
A/DA	0.26	0.61	0.23	0.36	0.65	0.83	1.00	0.81	0.54	0.68
Max IP	0.54	0.82	-0.05	0.37	0.75	0.74	0.81	1.00	0.60	0.86
DR	0.49	0.73	-0.08	0.01	0.97	0.68	0.54	0.60	1.00	0.67
P	0.74	0.99	-0.35	0.10	0.77	0.57	0.68	0.86	0.67	1.00



	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

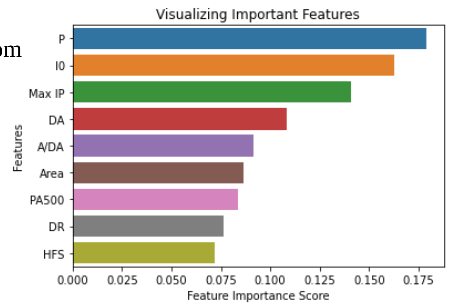
For more observations we can also use the describe function for data frame.

Then we shall remove the ‘case#’ variable split our data into test and training in the ration of 20% & 80% respectively, while also deleting the ‘class’ column, our target data, from the test data.

We will then perform Hyper-Parameter Optimization using GridSearchCV to find the best parameters to use in our model.

```
Best Parameters using grid search:
 {'bootstrap': True, 'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto'}
Time taken in grid search:  20.08
```

Next we will train our model with the training data and use feature_importances_ from sklearn to find the best features for our data and plot the importance of each feature.



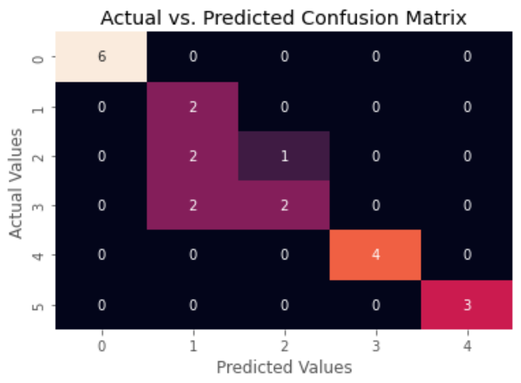
We can tell that the feature ‘P’ is the most important and the feature HFS is the least. We will remove ‘HFS’ from out dataset.

Using ‘HFS’ removed we will again train our model with the best parameters given by grid search.

```
print("Here is our accuracy on the test set:\n {0:.3f}"\
      .format(accuracy_rf))
```

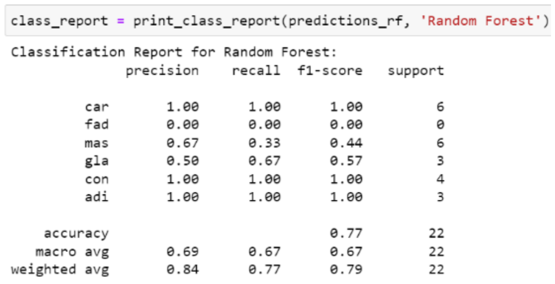
```
Here is our accuracy on the test set:
 0.773
```

We got an accuracy score of 0.773, this does not make much sense so we will plot the results in a confusion matrix.



Our confusion matrix looks good and acceptable, there are not many false positive and false negatives.

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



```
class_report = print_class_report(predictions_rf, 'Random Forest')
```

```
Classification Report for Random Forest:
              precision    recall  f1-score   support

         car       1.00      1.00      1.00         6
         fad       0.00      0.00      0.00         0
         mas       0.67      0.33      0.44         6
         gla       0.50      0.67      0.57         3
         con       1.00      1.00      1.00         4
         adi       1.00      1.00      1.00         3

    accuracy                           0.77        22
   macro avg       0.69      0.67      0.67        22
weighted avg       0.84      0.77      0.79        22
```

4. References

1. https://medium.com/analytics-vidhya/random-forest-classification-and-its-mathematical-implementation-1895a7bb743e

2. https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial

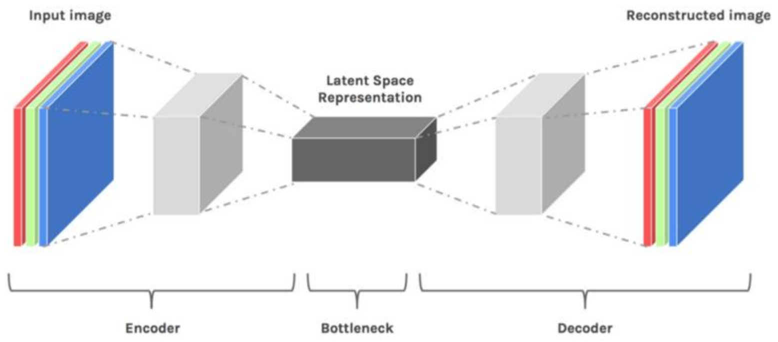
3. https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/

Autoencoder Experiment Report

1. Introduction

Autoencoder is a type of Artificial neural network which has the ability to learn the input representation. It can encode and compress data efficiently and then learn from the process how to reverse it and construct the input from the compressed data.

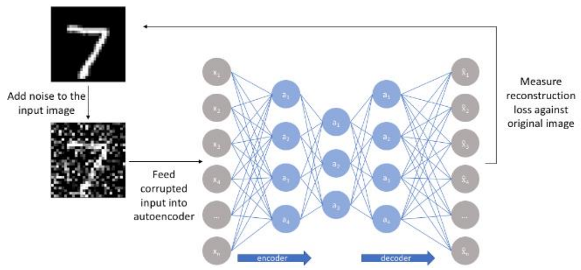
There are three main parts in an autoencoder: encoder, decoder and latent block(or bottle neck . The encoder extracts the best features, the latent space is used for storage of these features, and the decoder does the reverse of encoder and reconstructs an image from using the features.

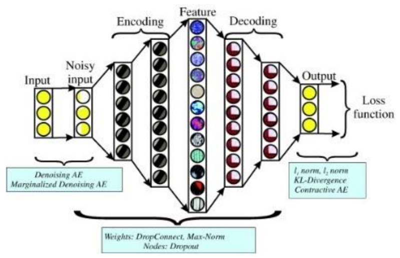


One application of Autoencoder is denoising data. Data can lose quality over transmission on the internet. Autoencoders can be used to get back the lost quality.

2. Types of Autoencoder:

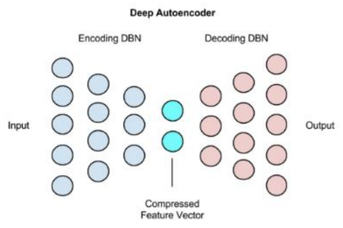
1. Denoising Autoencoder: in this encoder we create a copy of the input data with some noise added. This avoids the generalization of the input and makes the models learn features from the input data and train to recover the original data.
   It minimizes the loss function between the output and the noisy input. It is easy to setup. The only drawback is that his model does not memorize the training as the input and output keeps changing


Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder



2. Sparse Autoencoder: in this encoder, we have sparsity penalty in the training criterion. There are two ways to add sparsity: L1 regularization and KL-divergence. Adding sparsity helps to activate only fewer but insightful nodes, which helps the model learn from latent representations instead of redundant nodes. This also prevents overfitting.

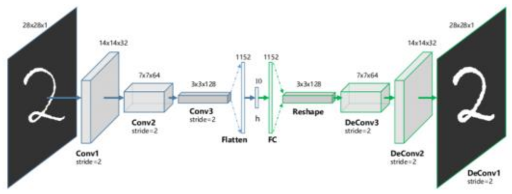
3. Deep Autoencoder: It works by creating two identical deep belief networks: one for encoding and one for decoding. Unsupervised layer by layer pretraining is used for this model. Each is a Restricted Boltzmann Machine which is the base of deep belief network, a binary transform is performed after each layer. Encoding is fast using Deep autoencoder and they are used in topic modeling across a set of documents.



4. Contractive Autoencoder: It is just like sparse autoencoder, but here the regularizer used is the calculated by the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input. This model is robust, and less sensitive to variations.

5. Undercomplete Autoencoder: They have a small number of hidden layers which helpt to capture the most important features in the data. They do not need any regularization as they maximize probability.

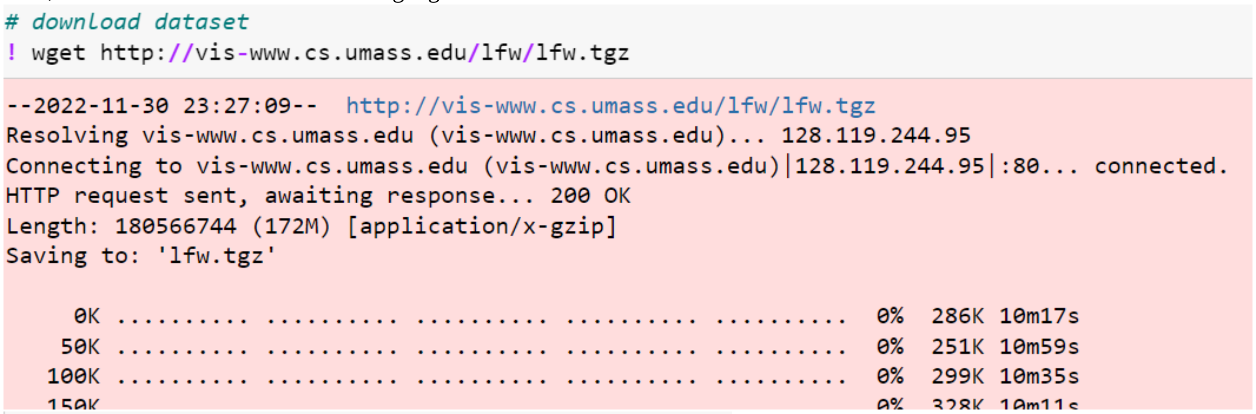
6. Convolutional Autoencoder: Convolutional Autoencoder uses a mix sum of signals to reconstruct the input data rather than taking them one at a time like other models. It uses convolution neural network, which is better at retaining connected information between the data.



3. Experiment

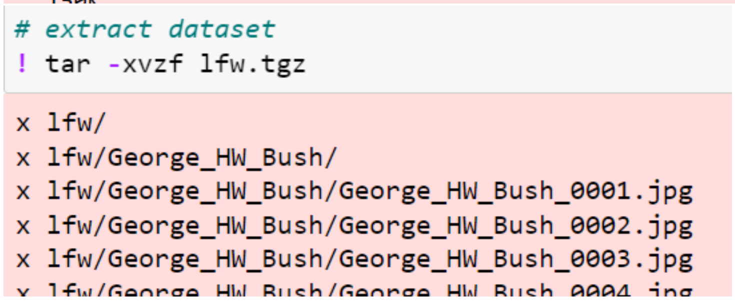
We will use the ‘Labeled Faces in the Wild’ dataset. It contains the photographs of faces used for face recognition. We will use this dataset to build a model for image resolution enhancement.

First, we will download the data set using wget and extract it.



```
# download dataset
! wget http://vis-www.cs.umass.edu/lfw/lfw.tgz
```

```
--2022-11-30 23:27:09--  http://vis-www.cs.umass.edu/lfw/lfw.tgz
Resolving vis-www.cs.umass.edu (vis-www.cs.umass.edu)... 128.119.244.95
Connecting to vis-www.cs.umass.edu (vis-www.cs.umass.edu)|128.119.244.95|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 180566744 (172M) [application/x-gzip]
Saving to: 'lfw.tgz'

     0K .......... .......... .......... .......... ..........  0%  286K 10m17s
    50K .......... .......... .......... .......... ..........  0%  251K 10m59s
   100K .......... .......... .......... .......... ..........  0%  299K 10m35s
   150K                                                         0%  328K 10m11s
```



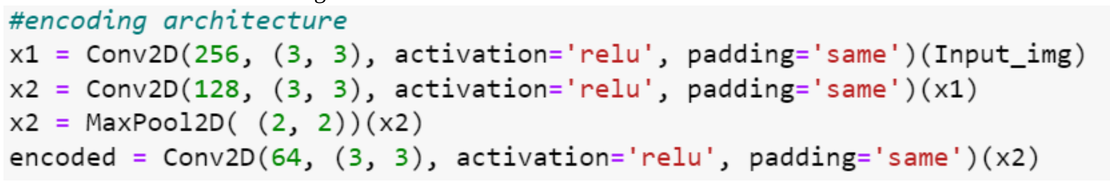
```
# extract dataset
! tar -xvzf lfw.tgz
```

```
x lfw/
x lfw/George_HW_Bush/
x lfw/George_HW_Bush/George_HW_Bush_0001.jpg
x lfw/George_HW_Bush/George_HW_Bush_0002.jpg
x lfw/George_HW_Bush/George_HW_Bush_0003.jpg
x lfw/George_HW_Bush/George_HW_Bush_0004.jpg
```

We will reduce the size of our images as per the limitations of our device and to reduce the training time. I have set it at 80x80.
Then we will split the date into training and test. We will keep aside the test dataset and used it for model evaluation.

We then reduce the resolution of the images to 40% so as to add noise in the training data, which we will later enhance using our model. We will use this low resolution image dataset as training data for our model.

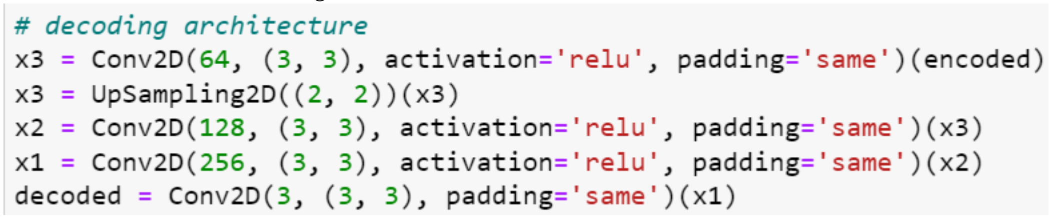
We will then define our encoding model.



```
#encoding architecture
x1 = Conv2D(256, (3, 3), activation='relu', padding='same')(Input_img)
x2 = Conv2D(128, (3, 3), activation='relu', padding='same')(x1)
x2 = MaxPool2D( (2, 2))(x2)
encoded = Conv2D(64, (3, 3), activation='relu', padding='same')(x2)
```

Our encoder consists of 2 Convolution neural networks followed by a max pool layer and then again a Convolution neural network. We have used the activation function as relu for all layers.

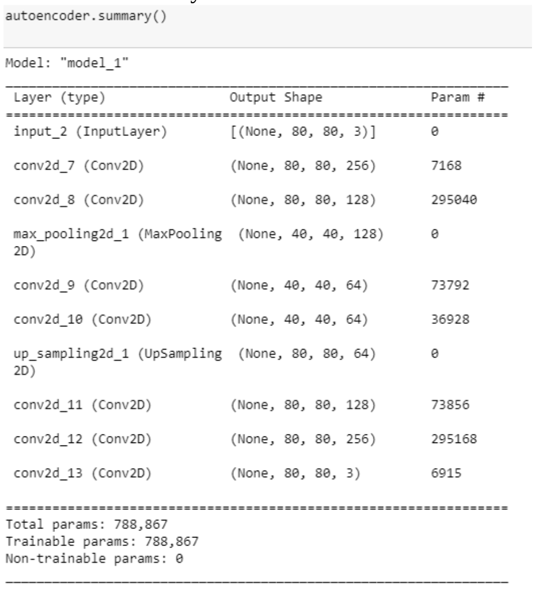
We will then define our decoding model.



```
# decoding architecture
x3 = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)
x3 = UpSampling2D((2, 2))(x3)
x2 = Conv2D(128, (3, 3), activation='relu', padding='same')(x3)
x1 = Conv2D(256, (3, 3), activation='relu', padding='same')(x2)
decoded = Conv2D(3, (3, 3), padding='same')(x1)
```

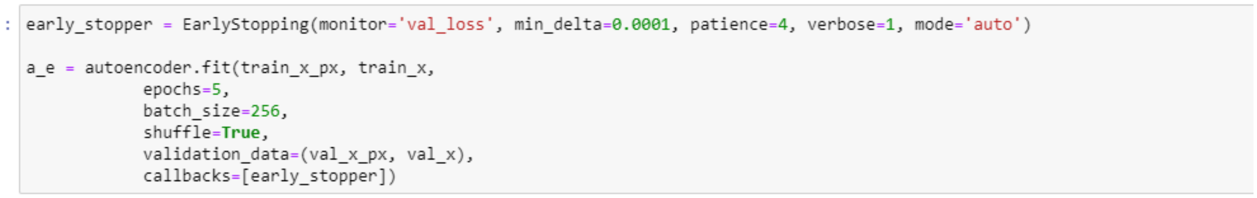
Our decoder consists of 1 Convolution neural network layer followed by an up-sampling layer and then 2 Convolution neural networks. We have used the activation function as relu for all layers.

Our model summary is:



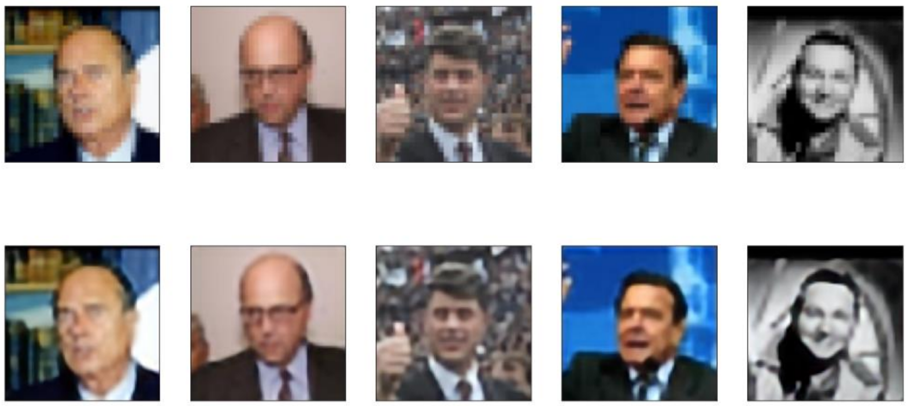
```
autoencoder.summary()

Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 80, 80, 3)]       0

 conv2d_7 (Conv2D)           (None, 80, 80, 256)       7168

 conv2d_8 (Conv2D)           (None, 80, 80, 128)       295040

 max_pooling2d_1 (MaxPooling  (None, 40, 40, 128)      0
 2D)

 conv2d_9 (Conv2D)           (None, 40, 40, 64)        73792

 conv2d_10 (Conv2D)          (None, 40, 40, 64)        36928

 up_sampling2d_1 (UpSampling  (None, 80, 80, 64)       0
 2D)

 conv2d_11 (Conv2D)          (None, 80, 80, 128)       73856

 conv2d_12 (Conv2D)          (None, 80, 80, 256)       295168

 conv2d_13 (Conv2D)          (None, 80, 80, 3)         6915

=================================================================
Total params: 788,867
Trainable params: 788,867
Non-trainable params: 0
_________________________________________________________________
```

I have set the model parameters as epochs = 5 and batch size = 256 depending on the limitations of my machine.



```
: early_stopper = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=4, verbose=1, mode='auto')

  a_e = autoencoder.fit(train_x_px, train_x,
                 epochs=5,
                 batch_size=256,
                 shuffle=True,
                 validation_data=(val_x_px, val_x),
                 callbacks=[early_stopper])
```

Now we can use the trained model on our test dataset:



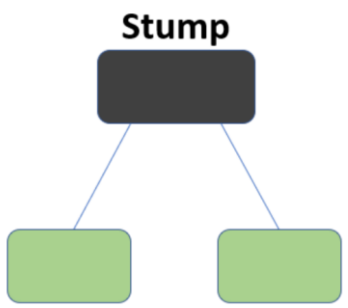
4. References

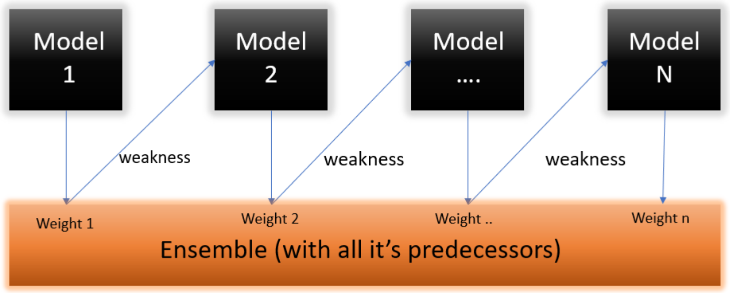
1. https://www.analyticsvidhya.com/blog/2020/02/what-is-autoencoder-enhance-image-resolution/
2. https://medium.com/ai%C2%B3-theory-practice-business/understanding-autoencoders-part-i-116ed2272d35
3. https://blog.francium.tech/enhance-images-with-autoencoders-58afa4fd638f
4. https://medium.com/@syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6
5. http://vis-www.cs.umass.edu/lfw/lfw.tgz

AdaBoost Experiment Report

1. Introduction

Adaptive Boosting is a machine learning algorithm used in ensemble method. It iteratively combines and learns from the mistakes of weak classifiers and promote them into strong ones. It can use any classifier but the most commonly used is Decision tree with one 1. This is called a decision stump.



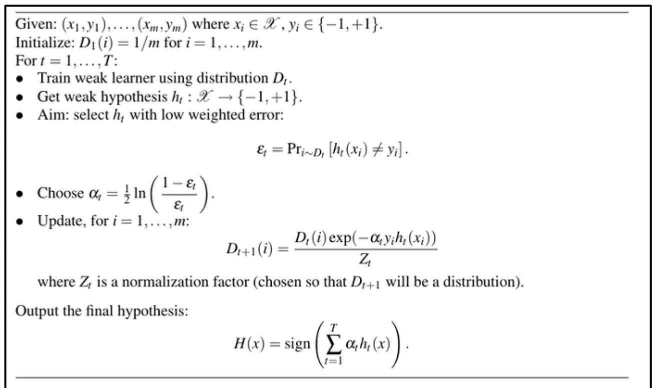


2. Algorithm

Steps for the algorithm:

1. A decision stump is created which is a weak classifier on the training data based on the weights and are given equal weights for the initial stump.
   Equation for assigning weights:
   $$w(x_i, y_i) = \frac{1}{N}, \ i = 1, 2, \ldots n$$
   N: number of datapoints
2. Decision stumps are created for each variable. Each decision stump is analyzed to see how well it classifies the data to the target class. This analysis is done by Gini index of each stump.
   $$\boxed{\begin{aligned} Gini\ Index &= 1 - \sum_{i=1}^{n} (P_i)^2 \\ &= 1 - [(P_+)^2 + (P_-)^2] \end{aligned}}$$
   Here, P+ is the probability of a positive class and P_ is the probability of a negative class.
3. We increase the weights of the stumps which classified incorrectly, this in turns make them classify correctly in the next decision stump. More accurate accuracy stumps are given more weights as well.
   $$Performance\ of\ the\ stump = \frac{1}{2}\log_e\left(\frac{1 - Total\ Error}{Total\ Error}\right)$$
   $$New\ sample\ weight = old\ weight * e^{\pm Amount\ of\ say\ (\alpha)}$$
4. We iterate step 2 and 3 until all the data has been correctly classified.

ADA Boosting steps with mathematics:



Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:
- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:
  $$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i].$$
- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:
  $$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Fig. 1 The boosting algorithm AdaBoost.

3. Experiment

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 using ADA bosting.

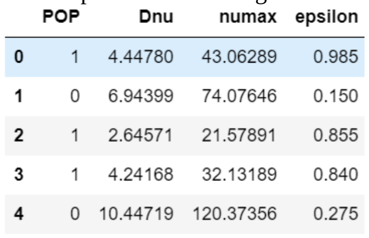
Feature:

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



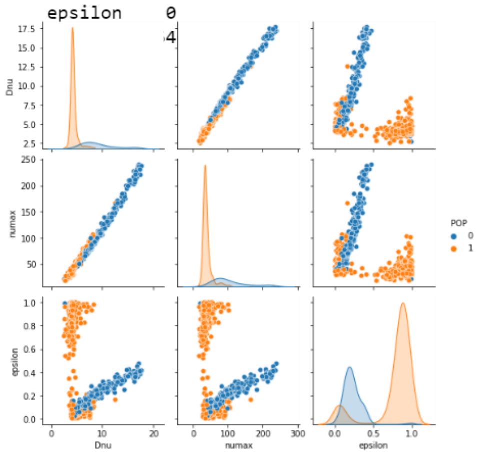
	POP	Dnu	numax	epsilon
0	1	4.44780	43.06289	0.985
1	0	6.94399	74.07646	0.150
2	1	2.64571	21.57891	0.855
3	1	4.24168	32.13189	0.840
4	0	10.44719	120.37356	0.275

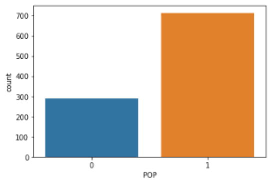
We will read the dataset from csv and store it into a data frame. Then we will perform our initial analysis of the data. We can see that all the features are integer or float so will not require any labeling or one hot encoding them.

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
```

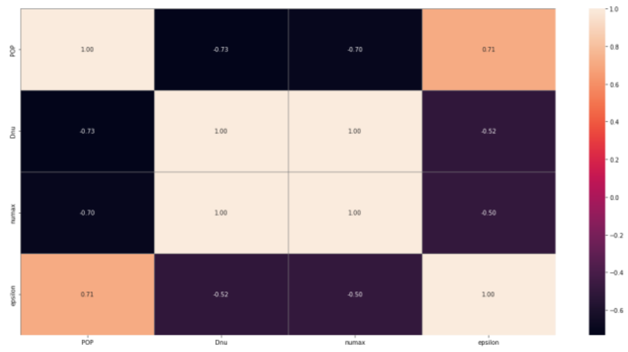
We will perform bivariate analysis using pair plot. Pairplot helps us plot pairwise bivariate distributions in a dataset which helps us summaries the data visually. The x & y axis have all the features plotting against the target variable POP and the diagonal has the distribution of each element against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



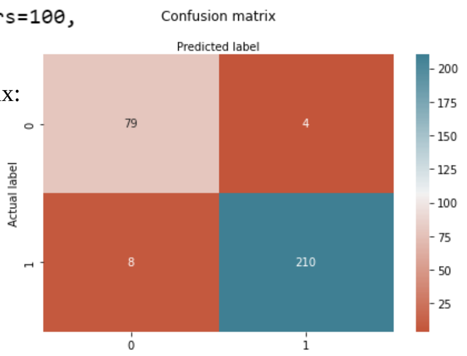
We will then split our dataset into test(30%) and training(70%)

We train the ADA boost model from our training set with n_estimators = 100

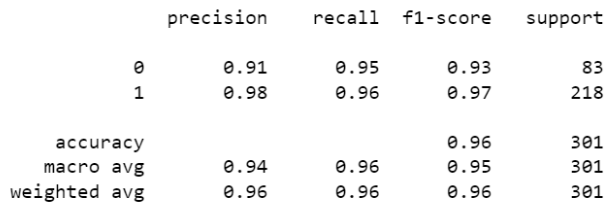
```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
ada = AdaBoostClassifier(DecisionTreeClassifier(),n_estimators=100, random_state=0)
ada.fit(X_train,y_train)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100,
                   random_state=0)
```

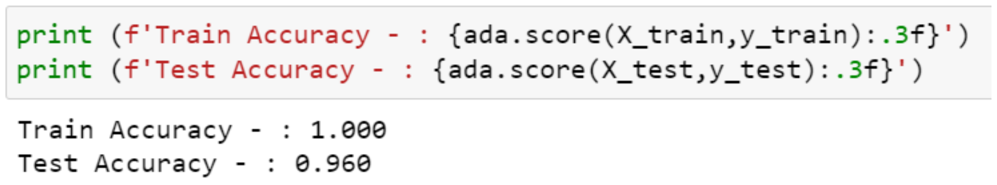
Now we will use our model to predict our test data and evaluate it using a confusion matrix:



And check our accuracy scores:



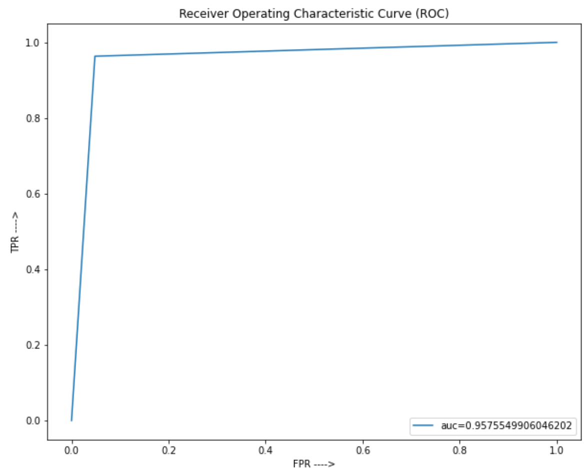
```
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        83
           1       0.98      0.96      0.97       218

    accuracy                           0.96       301
   macro avg       0.94      0.96      0.95       301
weighted avg       0.96      0.96      0.96       301
```



```
print (f'Train Accuracy - : {ada.score(X_train,y_train):.3f}')
print (f'Test Accuracy - : {ada.score(X_test,y_test):.3f}')
```

```
Train Accuracy - : 1.000
Test Accuracy - : 0.960
```

Our ROC/AUC curve fits as:



We can see that we have got a good accuracy score although it is a little overfit and our AUC is close to one so we can say that our model is good for use

4. References

1. https://blog.paperspace.com/adaboost-optimizer/
2. https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/
3. https://towardsdatascience.com/adaboost-for-dummies-breaking-down-the-math-and-its-equations-into-simple-terms-87f439757dcf
4. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5