Coding Assignment 3

Kaviyaa Vasudevan
Engineering Science (Data Science) MS
University at Buffalo, Buffalo, NY 14260
kaviyaav@buffalo.edu

Hidden Markov Model

1.1 Definition

The evolution of observable events that depend on internal, indirectly visible causes can be modeled statistically using a hidden Markov model (HMM). We refer to the observed event as a "symbol" and the unobservable component as the "state" that underlies the observation. The two stochastic processes that make up an HMM are a hidden process with hidden states and a visible process with observable symbols. Modeling biological sequences like DNA and protein sequences can potentially benefit from this method. A biological sequence typically comprises of smaller substructures with various purposes, and various functional sections frequently exhibit distinctive statistical characteristics.

1.2 Interpreting Hidden Markov Model

The observed symbol sequence is designated as x = x1x2...xL and the underlying state sequence is designated as y = y1y2...yL, where yn. represents the underlying state of the nth observation, xn. N and M stand for the number of different observations and the number of distinct states in the model, respectively. Each symbol xn takes on a finite number of possible values from the set of observations O = O1O2,...,ON and each state yn takes one of the values from the set of states S = 1,2,...,M. We presum that the concealed state sequence is a first-order Markov chain that is time-homogeneous. As a result, it follows that the likelihood of entering state j at the next time point depends only on the current state I and does not depend on any other factors.

P{yn+1 = j|yn = i, yn−1 = in−1,...,y1 = i1} = P{yn+1 = j|yn = i} = t(i, j)

for all states i, j $\in$ S and for all n $\geqslant$ 1. The fixed probability for making a transition from state i to state j is called the transition probability, and we denote it by t(i, j). For the initial state y1, we denote the initial state probability as π (i)= P {y1 = i} for all i $\in$ S. The probability that the n th observation will be xn = x depends only on the underlying state yn, hence

P{xn = x|yn = i, yn−1, xn−1, ... } = P{xn = x|yn = i} = e(x|i) – (2)

for all possible observations x $\in$ O, all state i $\in$ S, and all n $\geqslant$ 1. This is called the emission probability of x at state i, and we denote it by e(x | i). The three probability measures t(i, j), π (i), and e(x | i) completely specify an HMM. For convenience, we denote the set of these parameters as Θ.

Based on these parameters, we can now compute the probability that the HMM will generate the observation sequence x = x1 x2 ... xL with the underlying state sequence y = y1 y2 ... yL. This joint probability P {x,y | Θ} can be computed by

P{x, y |Θ} = P{x|y, Θ}P{y|Θ}, - (3)

where

P{x|y, Θ} = e(x1|y1)e(x2|y2)e(x3|y3) ... e(xL|yL) - (4)

P{y|Θ} = π(y1)t(y1, y2)t(y2, y3) ... t(yL−1, yL). - (5)

As we can see, computing the observation probability is straightforward when we know the underlying state sequence.

1.3 Gaussian Mixture Model

The most intricate model that is pre-built is this one. The Gaussian mixture emissions model makes the assumption that each hidden state's multivariate Gaussian distribution contributed a different mixture to the values in X. A multivariate mean and covariance matrix characterizes each multivariate Gaussian distribution in the mixture.

A function called a Gaussian Mixture is made up of several Gaussians, each of which is denoted by the string k 1,., K, where K is the number of clusters in our dataset. The following parameters are the components of each Gaussian k in the mixture:

- A mean μ that defines its centre.
- A covariance Σ that defines its width. This would be equivalent to the dimensions of an ellipsoid in a multivariate scenario.

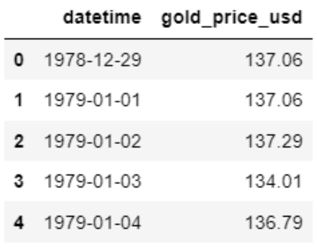
There is a mixing probability that determines the size of the Gaussian function. we can place certain constraints on the covariance matrices for the Gaussian mixture emission model. The covariance matrix for each multivariate Gaussian distribution in the mixture is diagonal for each hidden state, and these matrices may differ. The covariance matrix for each multivariate Gaussian distribution in the mixture is proportional to the identity matrix for each hidden state, and these matrices may differ.

- covariance type = "spherical"
- covariance type = "full" — the covariance matrices for all of the hidden states and mixtures are unrestricted.
- covariance type = "tied" — all mixture components utilize the same complete covariance matrix for each hidden state.

In contrast to the Gaussian emission model, these matrices may vary between the hidden states.

1.3 Data set

In addition to the spot price of gold, there is also the LBMA Gold Price and a number of local prices. The other regional gold prices are significant for local markets, while the LBMA Gold Price serves as a significant benchmark for the whole gold market. This data collection offers the price of gold for several time periods (daily, weekly, monthly, and annually) and in the main trading, producer, and consumer currencies going back to1978. The dataset has two attributes namely the datetime which specifies the data and time and the price of gold in US dollars on the particular data specified.



	datetime	gold_price_usd
0	1978-12-29	137.06
1	1979-01-01	137.06
2	1979-01-02	137.29
3	1979-01-03	134.01
4	1979-01-04	136.79

Figure 1.1: Dataset

2 Data Preprocessing

The goal of data preprocessing is to organize the original business data according to the new "business model," remove attributes that aren't relevant to the purpose of data mining, and provide clean, accurate, and simplified data to increase the effectiveness and quality of excavation carried out with the help of domain expertise. Data cleansing, integration, transformation, and reduction are the key outcomes of the data preprocessing. This allows for the correction of inaccurate, insufficient, and inconsistent data from the real world. Importing the data collection from which we build models is the main step in the data preprocessing process. The pandas library will be used to carry out this. We must import the relevant libraries first before importing the dataset. One of the trickiest issues in inductive ML is the removal of noise instances. Outliers—also known as highly diverging examples with an excessive number of null feature values—are typically found in the eliminated instances.

To achieve the purpose of clearing data, fill the null, smooth noise data, locate and eliminate isolated data, and address inconsistencies.

Data integration is the process of transferring information from many data sources, such as databases, data cubes, and plain old files, to a centralized data repository (such as a data warehouse).

Data conversion: Convert the data into a format that is appropriate for excavation, such as by proportionally zooming the attribution data to make it fall into a more narrowly defined area. Data compression: Although the dataset's compressed version is substantially smaller than its original version, the original data's integrity is not compromised. The reduced dataset will therefore be more affected by data mining, and the analytical outcome will be the same (or nearly the same). Data mining requires the preparation of data. According to statistics, data preprocessing takes up 60% of the time during the entire data mining process.

Another problem that is frequently addressed throughout the data preparation processes is missing data handling. Furthermore, when representing data from real-world sources, too many features are frequently used, even though only a small number of them may be relevant to the intended concept. When some traits are associated, there may be redundancy, making it unnecessary to model all of them; and interdependence, where two or more features work together to convey crucial information that would be hidden if any one of them were included alone. The act of detecting and eliminating as much unnecessary and redundant information as feasible is known as feature subset selection. This decreases the data's dimension and might make learning algorithms work more quickly and efficiently. One of the key steps in data preprocessing is splitting the data collection. When machine learning algorithms are used to make predictions on data that was not used to train the model, their performance is estimated using the train-test split technique. It is a quick and simple process to complete, and the outcomes will be useful in comparing the performance of machine learning algorithms for your predictive modeling problem.

2.1 Visualization

The next step is to visualize the results for which matplotlib library or seaborn library. To better reflect the situation of the market, we model daily changes in the gold price rather than the price of gold itself. We fitted a Gaussian emissions model with three hidden states to the daily variation in gold prices. We foresee at least three different regimes in the daily changes low, medium, and high volatility which is why we use three concealed states.

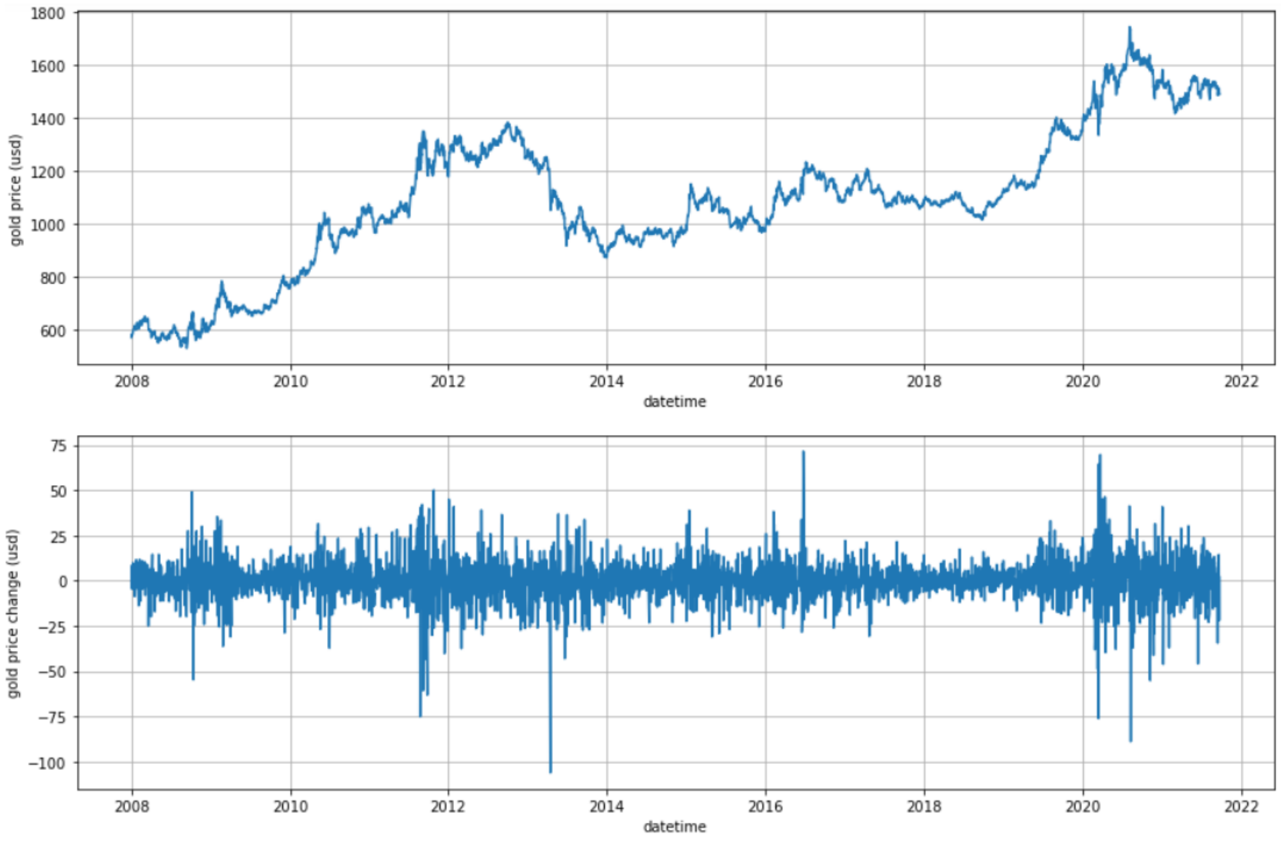
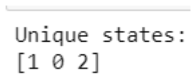


Figure 1.2 Changes in gold price

2.2 Fitting the Model and Predicting Results

After fitting the HMM model to our data we found that the model does indeed return 3 distinct hidden states, as we discover. These values are meaningless on their own; the model parameters must be examined to determine which state corresponds to which volatility regime.

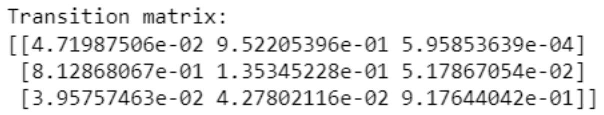


```
Unique states:
[1 0 2]
```

Figure 1.3: Examine model parameters

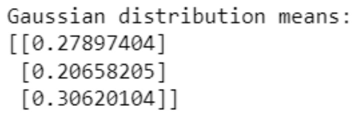
We discover that the model does, in fact, return three distinct hidden states. These statistics are meaningless on their own; it is necessary to check which state corresponds to which volatility regime by examining the model's parameters.

The diagonal elements are larger than the off-diagonal elements, according to the transition matrices for the three hidden states. This suggests that there is a low likelihood of a transition up or down; instead, the model likely to desire to stay in the state it is in.



```
Transition matrix:
[[4.71987506e-02 9.52205396e-01 5.95853639e-04]
 [8.12868067e-01 1.35345228e-01 5.17867054e-02]
 [3.95757463e-02 4.27802116e-02 9.17644042e-01]]
```

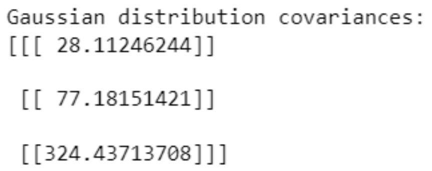
Figure 1.4: Transition matrix



```
Gaussian distribution means:
[[0.27897404]
 [0.20658205]
 [0.30620104]]
```

Figure 1.5 Gaussian distribution mean

we examine the Gaussian emission parameters. Every observable is taken from a multivariate Gaussian distribution, keep that in mind. The Gaussian mean is 0.27 for state 0, 0.20 for state 1, and 0.30 for state 2. Our current model might not be very good at truly reflecting the data given how comparable the average of states 0 and 2 are.



```
Gaussian distribution covariances:
[[[ 28.11246244]]

 [[ 77.18151421]]

 [[324.43713708]]]
```

Figure 1.6 Gaussian distribution covariance

The Gaussian covariances are also available. The covariance matrices are reduced to scalar values, one for each state, as a result of our data only having one dimension. The covariance is 28.1 for state 0, 77.8 for state 1, and 324.43 for state 2. This seems to support our initial hypothesis regarding the three different volatility regimes: low volatility should have a small covariance, whereas high volatility should have a very large covariance.

We observe that the states 0, 1, and 2 appear to correspond to low volatility, medium volatility, and high volatility when we plot the model's state predictions against the data.

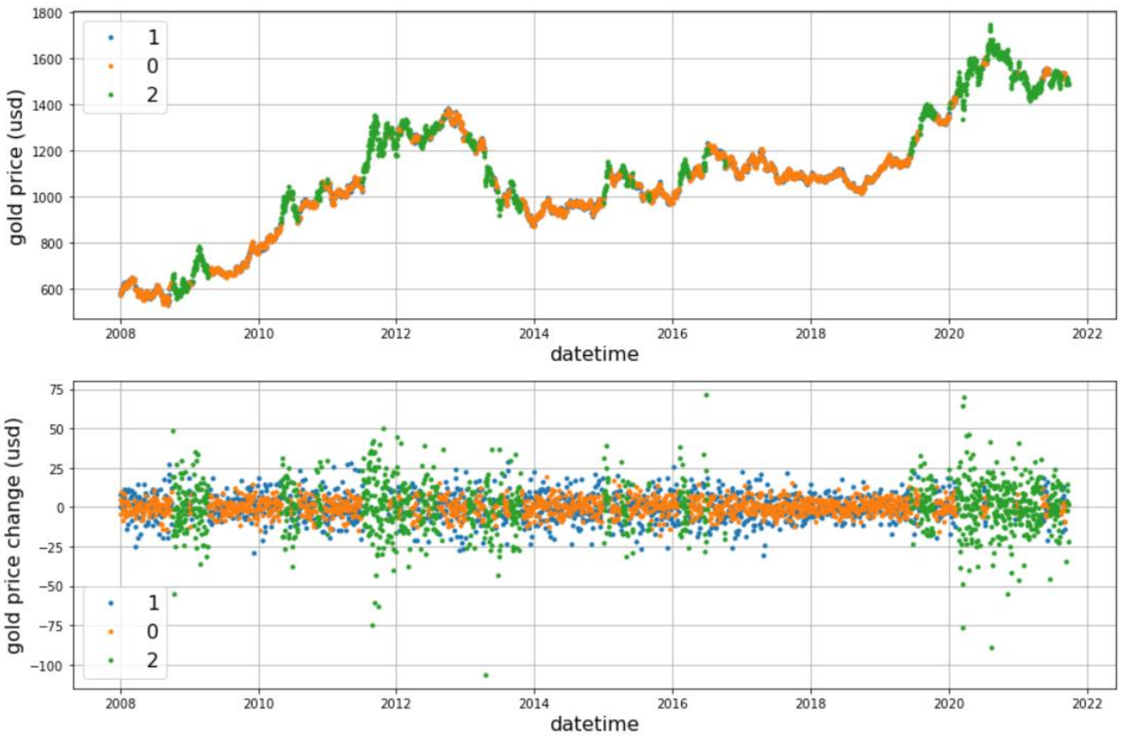


Figure 1.7 Visualizing the changes in gold prices

Coding Assignment 3

Kaviyaa Vasudevan
Engineering Science (Data Science) MS
University at Buffalo, Buffalo, NY 14260
kaviyaav@buffalo.edu

Random Forest

1.1 Definition

Random Forests are a combination of tree predictors where each tree depends on the values of a random vector sampled independently with the same distribution for all trees in the forest. The basic principle is that a group of "weak learners" can come together to form a "strong learner." A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree

The default values for the parameters controlling the size of the trees (e.g., max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those parameter values. The features are always randomly permuted at each split. Therefore, the best-found split may vary, even with the same training data, max_features=n_features and bootstrap=False, if the improvement of the criterion is identical for several splits enumerated during the search of the best split. To obtain a deterministic behaviour during fitting, random_state must be fixed.

The Random Forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.

1.2 Random Forest Analysis

1.2.1 Implementation in Scikit-learn

For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}$$

- ni sub(j)= the importance of node j
- w sub(j) = weighted number of samples reaching node j
- C sub(j)= the impurity value of node j
- left(j) = child node from left split on node j
- right(j) = child node from right split on node j

The importance for each feature on a decision tree is then calculated as:

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

- fi sub(i)= the importance of feature i
- ni sub(j)= the importance of node j

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

$$normfi_i = \frac{fi_i}{\sum_{j \in all\ features} fi_j}$$

The final feature importance, at the Random Forest level, is it's average over all the trees. The sum of the feature's importance value on each trees is calculated and divided by the total number of trees:

$$RFfi_i = \frac{\sum_{j \in all\ trees} normfi_{ij}}{T}$$

- RFfi sub(i)= the importance of feature i calculated from all trees in the Random Forest model
- normfi sub(ij)= the normalized feature importance for i in tree j
- T = total number of trees

1.3 Random Forest Model with default hyperparameters

Before understanding the working of the random forest, we must look into the ensemble technique. Ensemble uses two types of methods: Bagging and Boosting

1.3.1 Bagging– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest. . Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

1.3.2 Boosting– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST

1.3.3 Steps involved in random forest algorithm:

Step 1: In Random Forest n number of random records are taken from the data set having k number of records.
Step 2: Individual decision trees are constructed for each sample.
Step 3: Each decision tree will generate an output.
Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.

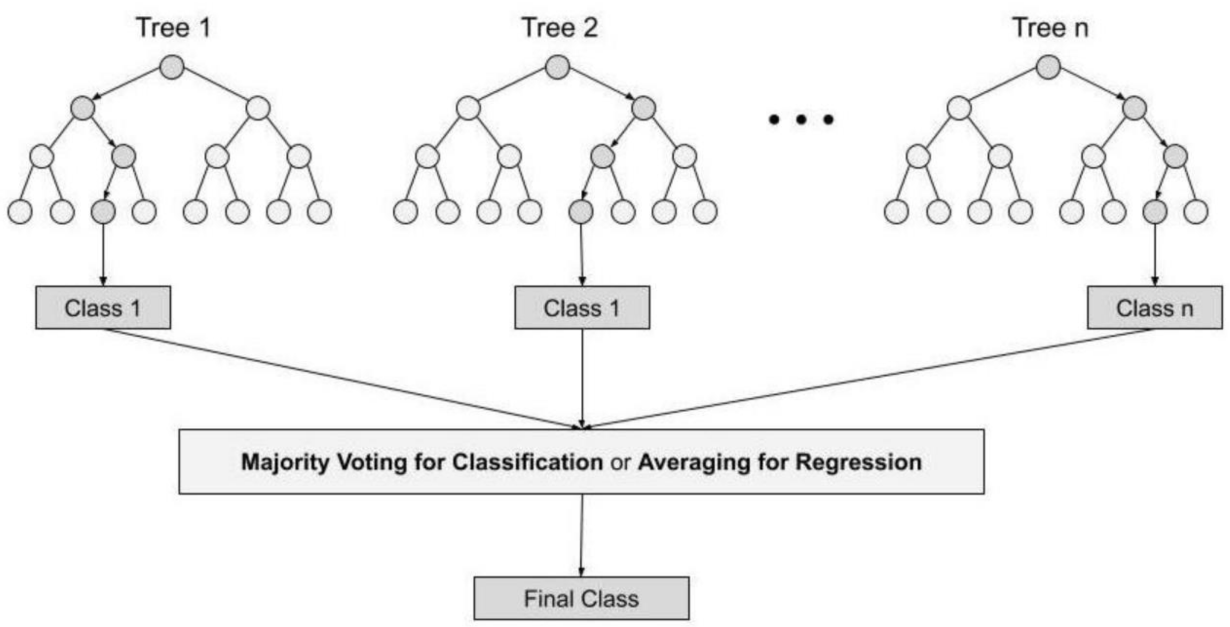

Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority Voting for Classification or Averaging for Regression
Final Class

Figure 1.1 Random Forest Algorithm

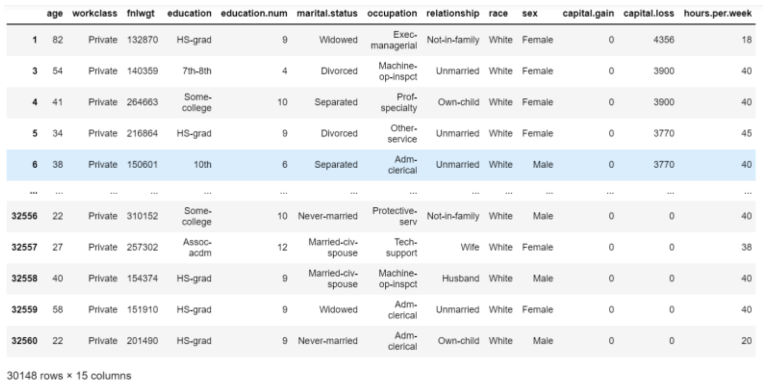
1.4 Interpretation of Random Forest:

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster. Following hyperparameters increases the predictive power:
1. n_estimators– number of trees the algorithm builds before averaging the predictions.
2. max_features– maximum number of features random forest considers splitting a node.
3. mini_sample_leaf– determines the minimum number of leaves required to split an internal node.
Following hyperparameters increases the speed:
1. n_jobs– it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

2. random_state– controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.
3. oob_score – OOB means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

1.5 Data set

The Adult UCI data set contains the questionnaire data of the Adult database (originally called the Census Income Database) formatted as a data frame. The Adult data set contains the data already prepared and coerced to transactions for use with a rules. Adult is an object of class transactions with 48842 transactions and 115 items. The Adult UCI data set contains a data frame with 48842 observations on the following 15 variables. Age a numeric vector, Workclass a factor with levels Federal-gov, Local-gov, Never-worked, Private, Self-emp-inc, Self-emp-not-inc, State-gov, and Without-pay. Education an ordered factor with levels Preschool < 1st-4th < 5th-6th < 7th-8th < 9th < 10th < 11th < 12th < HS-grad < Prof-school < Assoc-acdm < Assoc-voc < Some-college < Bachelors < Masters < Doctorate. Education-num a numeric vector. Marital-status a factor with levels Divorced, Married-AF-spouse, Married-civ-spouse, Married-spouse-absent, Never-married, Separated, and Widowed. Occupation a factor with levels Adm-clerical, Armed-Forces, Craft-repair, Exec-managerial, Farming-fishing, Handlers-cleaners, Machine-op-inspct, Other-service, Priv-house-serv, Prof-specialty, Protective-serv, Sales, Tech-support, and Transport-moving. Relationship a factor with levels Husband, Not-in-family, Other-relative, Own-child, Unmarried, and Wife. Race a factor with levels Amer-Indian-Eskimo, Asian-Pac-Islander, Black, Other, and White. Sex a factor with levels Female and Male. Capital-gain a numeric vector. Capital-loss a numeric vector. Fnlwgt a numeric vector. Hours-per-week a numeric vector. Native-country a factor with levels Cambodia, Canada, China, Columbia, Cuba, Dominican-Republic, Ecuador, El-Salvador, England, France, Germany, Greece, Guatemala, Haiti, Holand-Netherlands, Honduras, Hong, Hungary, India, Iran, Ireland, Italy, Jamaica, Japan, Laos, Mexico, Nicaragua, Outlying-US(Guam-USVI-etc), Peru, Philippines, Poland, Portugal, Puerto-Rico, Scotland, South, Taiwan, Thailand, Trinadad&Tobago, United-States, Vietnam, and Yugoslavia. Income an ordered factor with levels small < large.



	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	3770	45
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	3770	40
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20

30148 rows × 15 columns

Figure 1.2 Dataset

2 Data Preprocessing

The data preprocessing can often have a significant impact on generalization performance of a ML algorithm. The foremost process of data preprocessing is to import the data set by which we develop models. This will be performed using the pandas library. Prior to importing the dataset we are required to import the necessary libraries. The elimination of noise instances is one of the most difficult problems in inductive ML. Usually the removed instances have excessively deviating instances that have too many null feature values which are also referred to as outliers. Missing data handling is another issue often dealt with in the data preparation steps. Moreover, in real-world data, the representation of data often uses too many features, but only a few of them may be related to the target concept. There may be redundancy, where certain features are correlated so that is not necessary to include all of them in modeling; and interdependence, where two or more features between them convey important information that is obscure if any of them is included on its own. Feature subset selection is the process of identifying and removing as much irrelevant and redundant information as possible. This reduces the dimensionality of the data and may allow learning algorithms to operate faster and more effectively. Splitting the data set becomes one of the significant processes of data preprocessing. The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem.

The dataset contains missing values that are marked with a question mark character (?). There are a total of 48,842 rows of data, and 3,620 with missing values, leaving 45,222 complete rows. There are two class values '>50K' and '<=50K', meaning it is a binary classification task. The classes are imbalanced, with a skew toward the '<=50K' class label.

'>50K': majority class, approximately 25%.

'<=50K': minority class, approximately 75%.

The Special characters that are found in the dataset is removed in this step. Our dataset contains the special characters such as "?", " ." . The rows containing the special characters are removed as a part of the cleaning process.

2.1 Visualization

The next step is to visualize the results for which matplotlib library can be used to make scatter plots of our training set results and test set results to see how close our model has predicted the values. The Seaborn Pair plot allows us to plot pairwise relationships between variables within a dataset. This creates a nice visualization and helps us understand the data by summarizing a large amount of data in a single figure. Pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters.

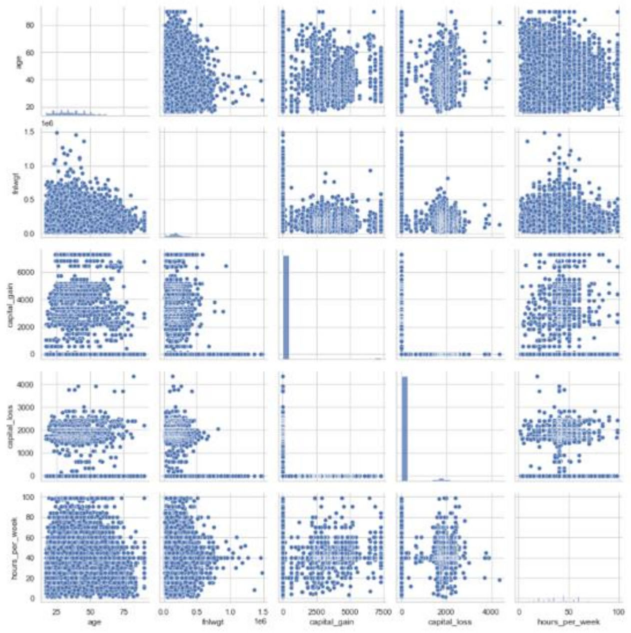


Figure 1.3 The above image shows the pair plot for all the columns.

The visualization results below for the income column specifies that there is a huge ratio of people earning below 50k compared to above 50k.

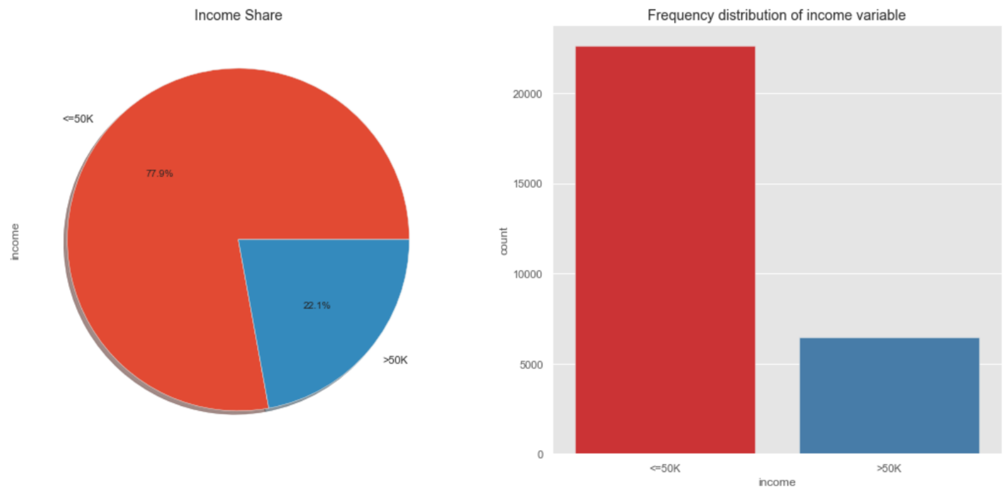


Figure 1.4 The above figure displays the visualization of income

2.1 Fitting the Model and Predicting Results

2.1.1 Random Forest with default hyperparameters

After splitting the test and train set, the random forest algorithm is applied to the dataset to check the accuracy and the prediction results. The accuracy score of the model came out to 82% which is a pretty good score.

Applying the random forest classifier to our prediction values is 81%.

```
The accuracy of the model is 81.89999999999999 %
```

Figure 1.5 Accuracy of the random forest model

2.1.2 AUC Random Forest

The roc_auc_score is defined as the area under the ROC curve, which is the curve having False Positive Rate on the x-axis and True Positive Rate on the y-axis at all classification thresholds. But it's impossible to calculate FPR and TPR for regression methods, so we cannot take this road. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance.
Applying the roc_auc_score to our prediction values is 87.3%.

```
The AUC Score  is 87.3 %
```

Figure 1.6 Accuracy of the random forest model using roc_auc_score

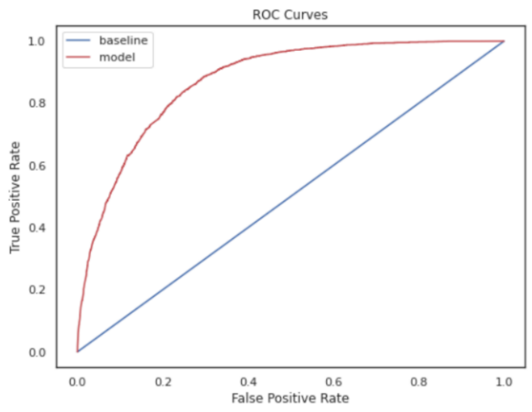


Figure 1.7 ROC curve

2.1.3 Confusion matrix

It gives direct comparisons of values like True Positives, False Positives, True Negatives and False Negatives. **TP** is 6807, **FP** is 639, **FN** is 1166 and **TN** is 1342.

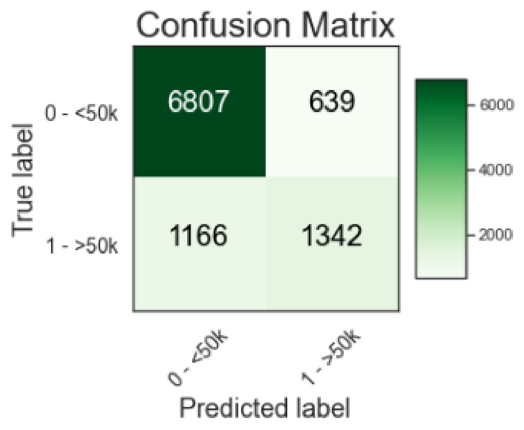


Figure 1.8 Confusion matrix of our Confusion matrix

2.2 Feature importance of Random Forest model

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.

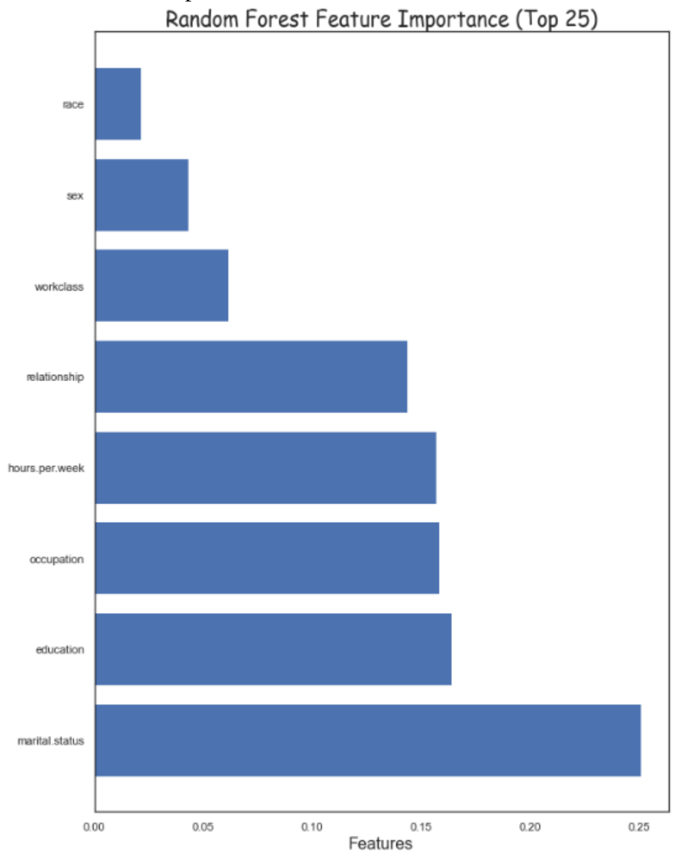


Figure 1.9 Feature importance of Random Forest model

Coding Assignment 3

Kaviyaa Vasudevan
Engineering Science (Data Science) MS
University at Buffalo, Buffalo, NY 14260
kaviyaav@buffalo.edu

Ada Boost Classifier

1.1 Definition

An AdaBoost classifier is a meta-estimator that first fits a classifier on the original dataset, then fits additional copies of the classifier on the same dataset with the weights of instances that were incorrectly classified being changed so that subsequent classifiers concentrate more on challenging cases. To improve classifier accuracy, it combines number of classifiers. An iterative ensemble algorithm is AdaBoost. AdaBoost classifier combines anumber of ineffective classifiers to create a strong classifier that has a high degree of accuracy. The fundamental idea underlying Adaboost is to train the data sample and set the classifier weights in each iteration in a way that provides accurate predictions of uncommon observations. Any machine learning method that accepts weights from the training set can be used as the basis classifier.

1.2 Adaboost Classifier Analysis

Adaboost must adhere to two requirements:

- On a variety of weighed training instances, the classifier should be trained interactively.
- It strives to minimize training error in order to offer the best match possible for these instances in each iteration.

1.3 Working of Adaboost Classifier

The process of algorithm is as follows:

1. Adaboost initially chooses a training subset at random.
2. By choosing the training set based on the precision of the previous training, it iteratively trains the AdaBoost machine learning model.
3. It gives incorrectly classified observations a larger weight so that they will have a higher chance of being correctly classified in the following round.
4. Additionally, according to the trained classifier's accuracy, weight is assigned to it in each iteration. The classifier that is more precise will be given more weight.
5. This process iterates until the entire training set fits perfectly, or until the specified maximum number of estimators has been reached.
6. Vote among all of the learning algorithms you created in order to categorize.

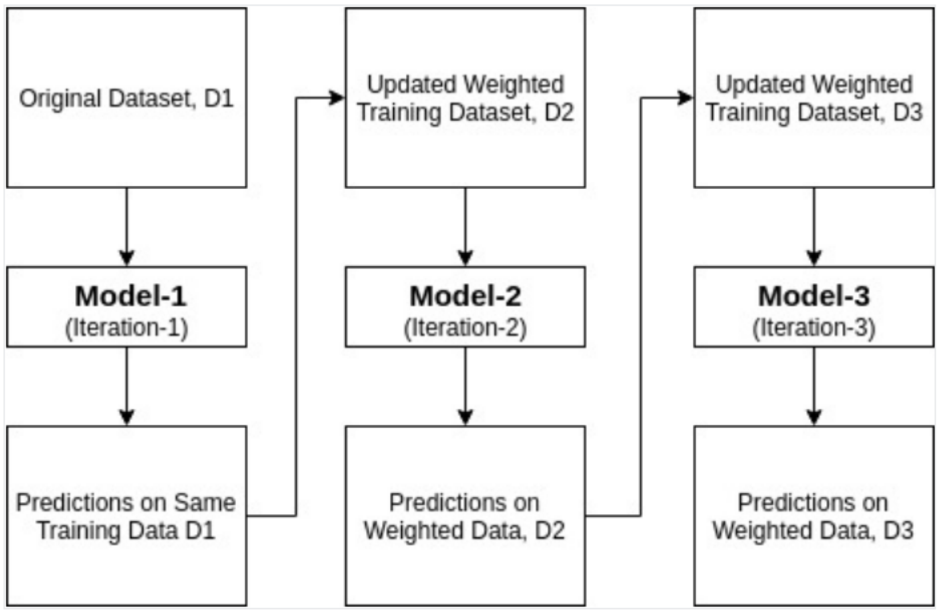

Original Dataset, D1
Updated Weighted Training Dataset, D2
Updated Weighted Training Dataset, D3
Model-1 (Iteration-1)
Model-2 (Iteration-2)
Model-3 (Iteration-3)
Predictions on Same Training Data D1
Predictions on Weighted Data, D2
Predictions on Weighted Data, D3

Figure 1.1 Working of Adaboost Classifier

1.4 Interpreting Adaboost Classifier Algorithm

Step 1: The First Base Learner: Creating

The algorithm takes the first feature, feature 1, and constructs the first stump, f1, to produce the first learner. It will produce an equal amount of stumps as features. As there are only 3 features in the dataset in the scenario below, it will produce 3 stumps. Three decision trees will be made from these stumps. The stumps-base learner model can be used to describe this procedure. Only one of these three models is chosen by the algorithm. When choosing a base learner, Gini and Entropy are taken into account. The same formula used for decision trees must be used to calculate Gini or Entropy. The least valuable stump will be first base learner. the beginner learner. Three attributes can be used to create each of the three stumps in the picture below. The correctly and wrongly classified records are indicated by the numbers beneath the leaves. The Gini or Entropy index is computed using these records. The base learner will be the stump with the lowest Gini or Entropy. Assume that stump 1 has the lowest entropy index. So let's use feature 1, or stump 1, as our first learner.

Here, feature (f1) accurately identified two records while wrongly classifying one. In the figure, the row that is highlighted in red is misclassified. We shall compute the overall error for this. across each of the learning algorithms you created.

1.4.1 Gini Index

Gini index or Gini impurity measures the degree /probability of a particular variable being wrongly classified when it is randomly chosen. The degree of the Gini index varies between 0 and 1, where 0 denotes that all elements belong to a certain class or if there exists only one class, and 1 denotes that the elements are randomly distributed across various classes. A Gini index of 0.5 denotes equally distributed elements into some classes.

The formula for Gini Index

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

where *pi* is the probability of an object being classified into a particular class.

Step 2: Calculating the Total Error (TE)

The total error is the sum of all the errors in the classified record for sample weights. In our case, there is only 1 error, so Total Error (TE) = 1/5.

Step 3: Calculating Performance of the Stump

In our case, TE is 1/5. By substituting the value of total error in the above formula and solving it, we get the value for the performance of the stump as 0.693. Why is it necessary to calculate the TE and performance of a stump? The answer is, we must update the sample weight before proceeding to the next model or stage because if the same weight is applied, the output received will be from the first model. In boosting, only the wrong records/incorrectly classified records would get more preference than the correctly classified records. Thus, only the wrong records from the decision tree/stump are passed on to another stump. Whereas, in AdaBoost, both records were allowed to pass and the wrong records are repeated more than the correct ones. We must increase the weight for the wrongly classified records and decrease the weight for the correctly classified records. In the next step, we will be updating the weights based on the performance of the stump.

Step 4 Updating Weights

For incorrectly classified records, the formula for updating weights is:
New Sample Weight = Sample Weight * e^(Performance)
In our case Sample weight = 1/5 so, 1/5 * e^ (0.693) = 0.399

For correctly classified records, we use the same formula with the performance value being negative. This leads the weight for correctly classified records to be reduced as compared to the incorrectly classified ones. The formula is:
New Sample Weight = Sample Weight * e^- (Performance)
Putting the values, 1/5 * e^-(0.693) = 0.100

The updated weight for all the records can be seen in the figure. As is known, the total sum of all the weights should be 1. In this case, it is seen that the total updated weight of all the records is not 1, it's 0.799. To bring the sum to 1, every updated weight must be divided by the total sum of updated weight. For example, if our updated weight is 0.399 and we divide this by 0.799, i.e. 0.399/0.799=0.50.

0.50 can be known as the normalized weight. In the below figure, we can see all the normalized weight and their sum is approximately 1.

Step 5 Creating a New Dataset

Now, it's time to create a new dataset from our previous one. In the new dataset, the frequency of incorrectly classified records will be more than the correct ones. The new dataset has to be created using and considering the normalized weights. It will probably select the wrong records for training purposes. That will be the second decision tree/stump. To make a new dataset based on normalized weight, the algorithm will divide it into buckets. So, our first bucket is from 0 – 0.13, second will be from 0.13 – 0.63(0.13+0.50), third will be from 0.63 – 0.76(0.63+0.13), and so on. After this the algorithm will run 5 iterations to select different records from the older dataset. Suppose in the 1st iteration, the algorithm will take a random value 0.46 to see which bucket that value falls into and select that record in the new dataset. It will again select a random value, see which bucket it is in and select that record for the new dataset. The same process is repeated 5 times.

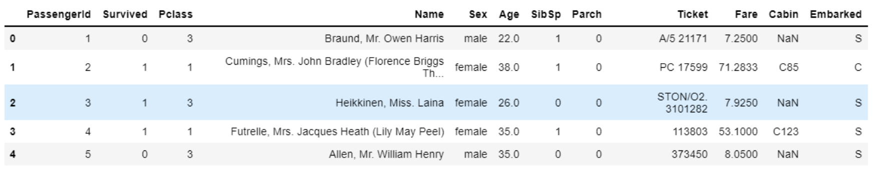
There is a high probability for wrong records to get selected several times. This will form the new dataset. It can be seen in the image below that row number 2 has been selected multiple times from the older dataset as that row is incorrectly classified in the previous one.

Based on this new dataset, the algorithm will create a new decision tree/stump and it will repeat the same process from step 1 till it sequentially passes through all stumps and finds that there is less error as compared to normalized weight that we had in the initial stage.
language processing.

1.6 Data set

The titanic data frame describes the survival status of individual passengers on the Titanic. The titanic data frame does not contain information from the crew, but it does contain actual ages of half of the passengers. The variables on our extracted dataset are pclass, survived, name, age, embarked, home.dest, room, ticket, boat, and sex. pclass refers to passenger class (1st, 2nd, 3rd), and is a proxy for socio-economic class. Age is in years, and some infants had fractional values. The titanic2 data frame has no missing data and includes records for the crew, but age is dichotomized at adult vs. child. The variable descriptions includes
Pclass - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
survival - Survival (0 = No; 1 = Yes)
name - Name
sex - Sex
age - Age
sibsp - Number of Siblings/Spouses Aboard
parch - Number of Parents/Children Aboard
ticket - Ticket Number
fare - Passenger Fare (British pound)
cabin - Cabin

embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
boat - Lifeboat
body - Body Identification Number
home.dest - Home/Destination



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 1.2 Dataset

2 Data Preprocessing

The effectiveness of an ML algorithm's generalization can frequently be significantly impacted by the preprocessing of the data. Importing the data collection from which we build models is the main step in the data preprocessing process. The pandas library will be used to carry out this. We must import the relevant libraries first before importing the dataset. One of the trickiest issues in inductive ML is the removal of noise instances. Outliers—also known as highly diverging examples with an excessive number of null feature values—are typically found in the eliminated instances. Another problem that is frequently addressed throughout the data preparation processes is missing data handling. Furthermore, when representing data from real-world sources, too many features are frequently used, even though only a small number of them may be relevant to the intended concept. There may be interdependence, where two or more features together transmit critical information that is obscure if any of them are included alone, and redundancy, where certain features may be associated such that, it is not required to include all of them in modeling. The act of detecting and eliminating as much unnecessary and redundant information as feasible is known as feature subset selection. This decreases the data's dimension and might make learning algorithms work more quickly and efficiently. One of the key steps in data preprocessing is splitting the data collection. When machine learning algorithms are used to make predictions on data that was not used to train the model, their performance is estimated using the train-test split technique. It is a quick and simple process to carry out, and the outcomes let you compare the effectiveness of machine learning algorithms for your predictive modeling issue.

2.2 Visualization

The next step is to visualize the dataset using the seaborn or matplotlib library. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. Here the variable embarked which represents the port of embarkation is plotted by its count. Southampton station is seemed to the highest number of people boarded.

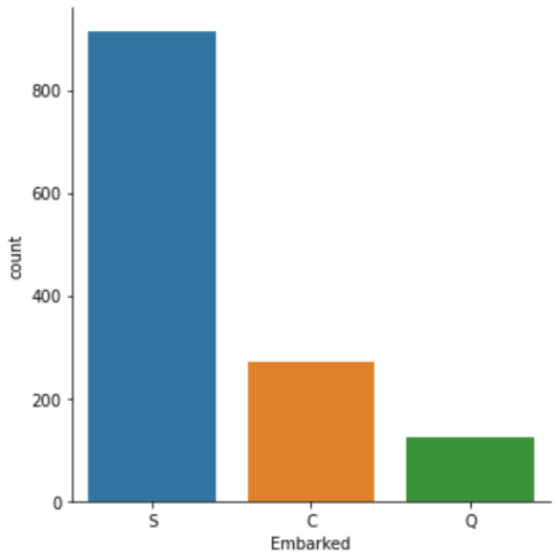
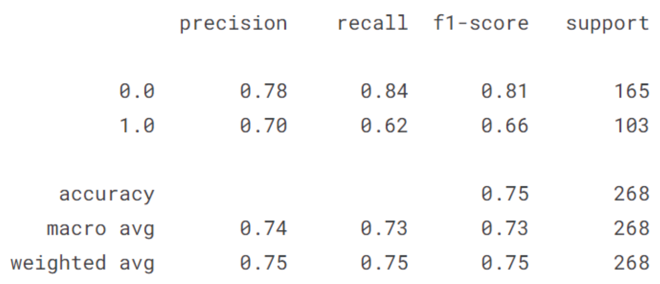


Figure 1.3 Visualizing embarked

2.2 Fitting the Model and Predicting Results

After dropping all the unwanted variables like name, ticket which is least correlated with the target variable, we start to fit the dataset into the adaboost classifier model. A dummy variable also known as an indicator variable is used here to convert all the variables to categorical variables. This would help in the better accuracy of the model. The next step is to split the entire dataset into test and train set using the sklearn library. Then we use the adaboost classifier to predict the results of test set. The classification report which we obtained after modelling is as follows:



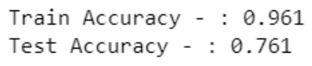
```
              precision    recall  f1-score   support

         0.0       0.78      0.84      0.81       165
         1.0       0.70      0.62      0.66       103

    accuracy                           0.75       268
   macro avg       0.74      0.73      0.73       268
weighted avg       0.75      0.75      0.75       268
```

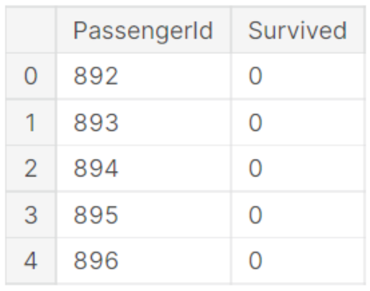
Figure 1.4 Classification report

After analyzing the values of precision, recall, f1-score, the model seems to perform pretty good on the dataset. The accuracy of the test and train set is calculated which is given below:



```
Train Accuracy - : 0.961
Test Accuracy - : 0.761
```

Figure 1.5 Accuracy results



	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

Figure 1.6 Predicted results

The accuracy of the system is really high meaning that the predicted results would be accurate to an extent. The predicted results of the test set is :

Coding Assignment 3

Kaviyaa Vasudevan
Engineering Science (Data Science) MS
University at Buffalo, Buffalo, NY 14260
kaviyaav@buffalo.edu

Autoencoder

1.1 Definition

Self-supervised machine learning models called autoencoders are employed to recreate input data in order to reduce its size. These models are referred to as self-supervised models since they are taught as supervised machine learning models and then operate as unsupervised models during inference.
A combination of NN (Neural Networks) layers are used in an autoencoder's encoder and decoder, which assists by replicating the input image and so reducing its size. These layers are CNN layers (Convolutional, Max Pool, Flattening, etc.) in the case of CNN Autoencoder, whilst RNN/LSTM uses their corresponding layers.

1.2 Autoencoder types

Numerous autoencoder kinds are available, and a few of them are included here with a brief description.

Convolutional Autoencoders (CAE) learn to encode the input in a sequence of straightforward signals and then reconstruct the input from those signals. Additionally, by using CAE, we can alter the geometry or produce the image's reflectance. Convolution layers and deconvolution layers are terms used to describe the encoder and decoder layers, respectively, in this form of autoencoder.

Variational Autoencoders: Similar to GANs, this kind of autoencoder can produce new images. Strong assumptions about the distribution of latent variables are frequently made by variational autoencoder models. For latent representation learning, they employ a variational approach, which yields a further loss component and a particular estimator for the training procedure termed the Stochastic Gradient Variational Bayes estimator. A variational autoencoder often matches the training data more closely than a normal autoencoder in terms.

Denoising autoencoders: These algorithms add some noise to the input image and then figure out how to get rid of it. Consequently, it is prevented from copying the input to the output without discovering aspects of the data. While training to recover the original, undistorted input, these autoencoders use a partially corrupted input. In order to remove the extra noise, the model learns a vector field for mapping the input data towards a lower-dimensional manifold that describes the natural data.

Deep autoencoders: Two symmetrical deep belief networks with four to five shallow layers make up a deep autoencoder. The encoding portion of the network is represented by one of the networks, and the decoding portion by the second network. They can learn more complex features since they have more layers than a straightforward autoencoder. The underlying units of deep belief networks, limited Boltzmann machines, make up the layers.

1.3 Autoencoder Analysis

The design of autoencoders

A three-part autoencoder consists of the following:

Encoder: An encoder is a feedforward, fully connected neural network that encodes an input image as a compressed representation in a smaller dimension after compressing the input into a latent space representation. The original image has been warped in the compressed form.

Code: The input that is supplied into the decoder is represented more briefly in this area of the network.

Decoder: The decoder is likewise a feedforward network with a topology resembling that of the encoder. This network is in charge of translating the input from the code back to its original dimensions.

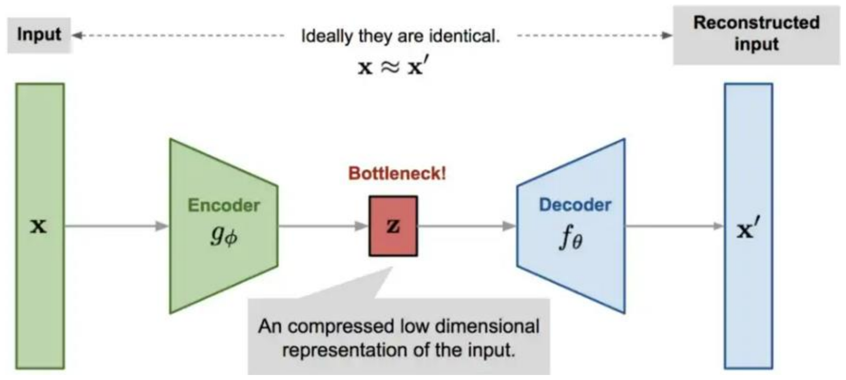

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
x
Encoder
g_ϕ
Bottleneck!
z
Decoder
f_θ
x'
An compressed low dimensional representation of the input.

Figure 1.1 Architecture of Autoencoder

Prior to the decoder decompressing the original input from the code, the input first passes via the encoder where it is compressed and stored in the layer called Code. The autoencoder's primary goal is to produce an output that matches the input exactly.

1.4 Expectation-Maximization (EM) Algorithm

The decoder block tries to rebuild the input image from the compressed representation after the encoder block converts an input vector of images into a compressed vector 'z' and sends it to the bottleneck. As an example, suppose we have an input image that is 28 × 28 pixels in size. Conventionally, this image must be flattened before being fed into a neural network. This image's flattened representation is (784), which is provided to the encoder (the first block). The bottleneck or latent space is then fed the encoder's output, which should be a condensed form; for example, if the latent space has 8, 16, or any other number of nodes, it just implies that there are less nodes there.

We were able to reduce an image with a size of 784 to 8 nodes, 16 nodes, or any other number. The decoder network then attempts to reconstruct the original (28 x 28) input image from the bottleneck's compressed state. This is how it operates.

As soon as the image is reconstructed, you compare the reconstructed image with the original image, compute the difference, and calculate the loss which can then be minimized.

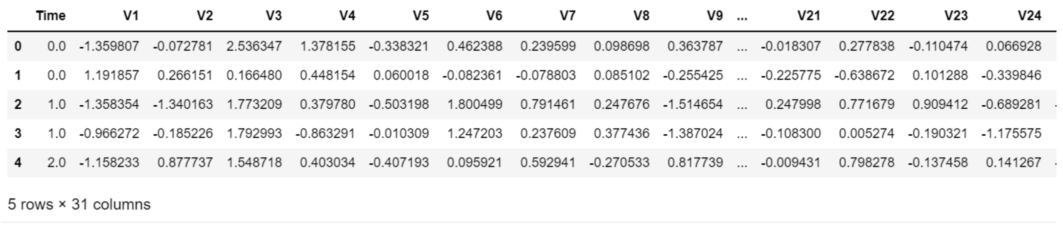
The Loss is calculated by:

$$L(\theta, \varphi) = \frac{1}{n}\sum_{i=1}^{n} (x^i - f_\theta(g_\varphi(x^i)))^2$$

As seen above, the loss function depends on 'theta' and 'phi' which are the parameters that define the encoder and the decoder. As explained earlier and from the autoencoder image above, the encoder is represented by Gφ, while the decoder is represented by Fθ and they simply mean the weights and bias of the neural network. So in the equation, we are summing up the difference between the original image, x`, and the reconstructed image Fθ(gφ(x`)).

1.4 Data set

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 31 columns

Figure 1.2: Dataset

2 Data Preprocessing

The task of data preprocessing is to organize the original business data with the new "business model" , clear those attributes irrelevant to the aim of data mining, supply clean, accurate, simplified data so as to improve the quality and efficiency of excavation under the guidance of domain knowledge. The data preprocessing mainly concludes data cleaning, integration, transformation and reduction. In this way, the dirty, incomplete and inconsistent data in real world can be corrected. The foremost process of data preprocessing is to import the data set by which we develop models. This will be performed using the pandas library. Prior to importing the dataset we are required to import the necessary libraries. The elimination of noise instances is one of the most difficult problems in inductive ML. Usually the removed instances have excessively deviating instances that have too many null feature values which are also referred to as outliers.

Data cleaning: By filling the null, smoothing noise data, identify and delete the isolated data, and solve inconsistency to attain the goal of clearing data.

Data integration: Save the data belonging to several data sources to a consistent data storage (such as data warehouse), these data sources may include several databases, data cubes or ordinary files.

Data conversion: Convert the data into one form that is suitable for excavation, for instance, zoom the attributive data in proportion, make it falls into a comparatively smaller specified zone.

Data reduction: The compressed data used to acquire the dataset is much smaller than the original data, but it still keeps its integrity. Thus, the data mining will have more effect on the condensed dataset, and produce the same analysis result. Data preprocessing is indispensable to data mining. Statistics suggests that data preprocessing takes up 60 percent of the time in a complete process of data mining.

Missing data handling is another issue often dealt with in the data preparation steps. Moreover, in real-world data, the representation of data often uses too many features, but only a few of them may be related to the target concept. There may be redundancy, where certain features are correlated so that is not necessary to include all of them in modeling; and interdependence, where two or more features between them convey important information that is obscure if any of them is included on its own. Feature subset selection is the process of identifying and removing as much irrelevant and redundant information as possible. This reduces the dimensionality of the data and may allow learning algorithms to operate faster and more effectively. Splitting the data set becomes one of the significant processes of data preprocessing. The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem.

2.1 Variational encoder for the dataset:

The "Variational" in the name refers to the extra restriction that the latent representation learn the probability distribution parameters of the input rather than an arbitrary encoding of the input. Variational Autoencoders likewise attempt to reconstruct the input. First, by precisely describing the mechanisms for generating both data and noise, VAE may learn to distinguish between the two, strengthening it. Two, applying disentanglement restrictions makes the latent space easier to interpret. Three, by sampling latent vectors and passing them through the decoder, you can create fresh samples.

2.2 Visualization:

After visualizing the preprocessed transaction features, the sample 1000 non-fraud transactions from the training set and plot with all fraudulent transactions in the training set. The T-SNE plot shows that after Time and Amount preprocessing, fraud transactions (in red) seem to be adequately separated from non-fraud transactions (in green) in this particular projection.

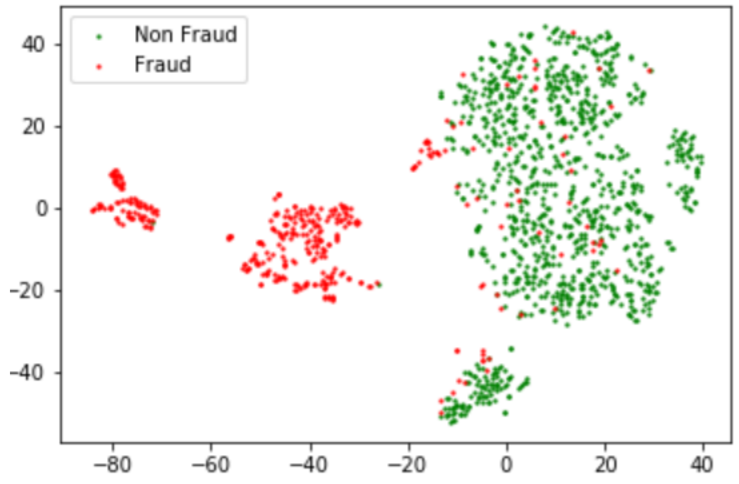


Figure 1.3 Visualizing the transaction features

2.3 Fitting the Model and Predicting Results

The training of a variational auto encoder resulted asa in a random unit multivariate normal vector of the latent dimension was chosen as the prior for the latent variables. The latent dimension is set to 2. The latent distribution parameters, which are the encoder's output, were purposefully chosen to be multivariate normal with non-zero covariance because I noticed it had an impact on how well I could distinguish between legitimate and fraudulent transactions. This suggests that the covariance of fraudulent transactions may exhibit patterns. The lower triangle of the 2-buy-2 covariance matrix has 3 covariance values, making a total of 5 distribution parameters that need to be learned. The data distribution parameters produced by the decoder follow feature-independent normal distributions. This decision is crucial. The majority of instances I could find was using binary images, as those in the MNIST collection, where the results would typically follow independent bernoulli distributions. It only makes sense to represent the output with normal distributions or something similar since the data are real-valued and often follow normal distributions. The fact that it will provide the necessary log probability loss during training is another significant consequence of having the correct distribution.

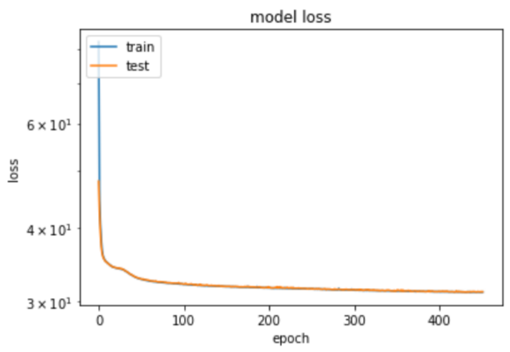


Figure 1.4 Log probability loss

The training stops when the validation losses fail to decrease for 20 consecutive epochs. Defining a function that would perform Monte Carlo on inputs to compute the reconstruction probability would be appropriate.

Visualizing the Latent Representations, the T-SNE plots of latent distribution parameters and samples. There is a clear separation between the fraud and non-fraud transactions around the origin [0,0] mean vector, a sign that the VAE is learning something meaningful.

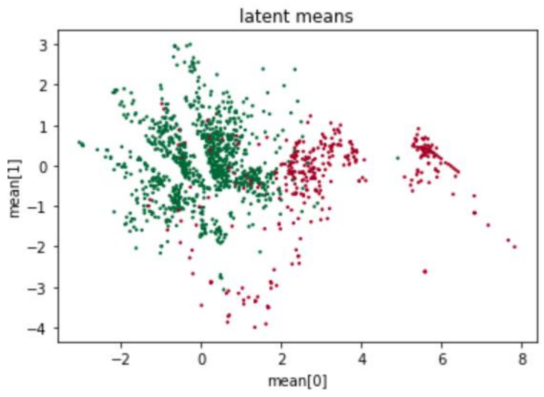


Figure 1.5 Visualizing the latent representation

The negative reconstruction log probability, and visualize a ROC curve across the range to see how it would perform if we were to build a threshold-based fraud detector on it.

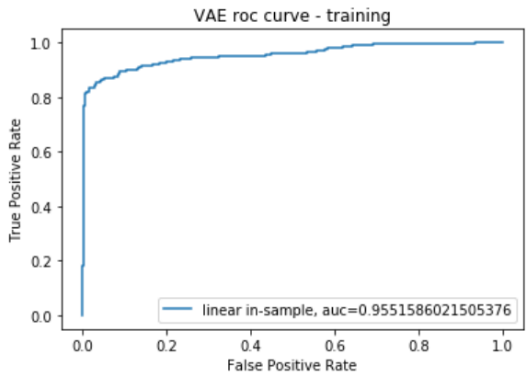


Figure 1.6 Visualizing the ROC curve

Above 0.96 in training set, pretty good number.

References

1. https://www.mygreatlearning.com/blog/adaboost-algorithm/
2. https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95
3. https://www.sciencedirect.com/topics/medicine-and-dentistry/hidden-markov-model#:~:text=The%20Hidden%20Markov%20model%20(HMM,then%20used%20for%20further%20analysis.
4. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/
5. https://www.analyticsvidhya.com/blog/2021/06/complete-guide-on-how-to-use-autoencoders-in-python/
6. https://www.statology.org/sklearn-classification-report/