Forest Cover Type Prediction

|  |  |
| --- | --- |
| Tirumala Sree Akhil Nandyala  *School of Electrical Engineering and Computer Science University of*Ottawa,  Ottawa, Canada  [tnand032@uottawa.ca](mailto:tnand032@uottawa.ca) | Hariprasad Ramakrishnan  *School of Electrical Engineering and Computer Science University of Ottawa,*  Ottawa, Canada  [hrama039@uottawa.ca](mailto:hrama039@uottawa.ca) |

*Abstract*— The Forest Cover Type Prediction is a supervised multi-class classification problem addressing the prediction of the forest cover type using cartographic features. The main goal of the project is to construct predictive models and then compare and evaluate their respective classification accuracies. The following are the tasks and goal associated with this project. The primary task would be to ensure the quality of the data via data cleansing. Post cleansing, dimensionality reduction (if required), feature Engineering, and correlation analysis using Pandas, sklearn packages would be used to build the model. K-fold approach would be used to train and test the model for different values of K and the metrics for each case would be recorded. The data set will be used to build classification models using various feasible algorithms like Naive Bayes, SVM, Random forests, K nearest neighbors, Decision trees and many more. The performance and evaluation of each algorithm will be recorded and visualized using matplotlib to graphically illustrate the qualitative and quantitative analyses performed as part of the project.

Keywords— model, classification, learning, dataset, feature

# Introduction

Data plays a highly significant role in today’s world and the buzz words that we often hear in the technological discussions have a lot to do with data. Millions of businesses thrive on the decisions that are made based on the predictions that have been arrived after quality data analysis. Various other critical fields like Governance, Armed Forces, Telecommunication, Entertainment industry, Education, E-commerce businesses also rely on data analysis results as a guidance for carrying out various operations which include critical decision making. Therefore, it is not surprising if someone infers that data science is one of the most ‘happening’ fields in today’s world.

Machine learning, as we know, plays a vital role in data science as it is used by data-scientists to build components (machine learning models) which help then to achieve a variety of task(s). As creation of these models require a large data for it to learn from, it is very important that we know every detail about the data that we are using and how it would impact the machine learning model which we select to use on the data.

In this paper, we will discuss about a problem which involves a data set about Forest Cover. This dataset includes cartographic information on tree type, shadow coverage, distance to nearby landmarks (roads etcetera), soil type, and local topography. The task is to predict the cover type based on the various cartographic features (no remote sensing). This dataset is part of the UCI Machine Learning Repository, and the original source can be found [in](https://archive.ics.uci.edu/ml/datasets/Covertype) their website. The original dataset owners are Jock A. Blackard, Dr. Denis J. Dean and Dr. Charles W. Anderson of the Remote Sensing and GIS Program at Colorado State University [1]. The task of identifying the forest cover type using the cartographic data can be done by developing a machine learning model that uses an algorithm that learns from a set of data (supervised learning) and performs multi-class classification. The challenge is to identify the algorithm and training method that provides the optimum result. (which is, in our case, performance measures like accuracy, precision and recall of the model). We will be using various data cleansing and pre-processing techniques along with evaluation techniques like test-train paradigm, k-fold validation methodology to create the optimum possible machine learning model.

# Problem domain description

As mentioned earlier, we have found the Forest Cover type dataset in the UCI Machine Learning Repository that takes forestry data from four wilderness areas in Roosevelt National Forest in northern Colorado. The observations are taken from 30m by 30m patches of forest that are classified as one of seven cover types:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

The forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type [2].

These areas taken for experimental analysis represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices. . Each instance or data consists of 12 measures that are broken into 54 distinct input variables (54 features). Of these, 10 are quantitative measures while the remaining 44 are boolean values indicating soil conditions and area. The target is label or dependant feature is the one of the above mentioned seven forest cover types. A short description of the features of the data is given below.

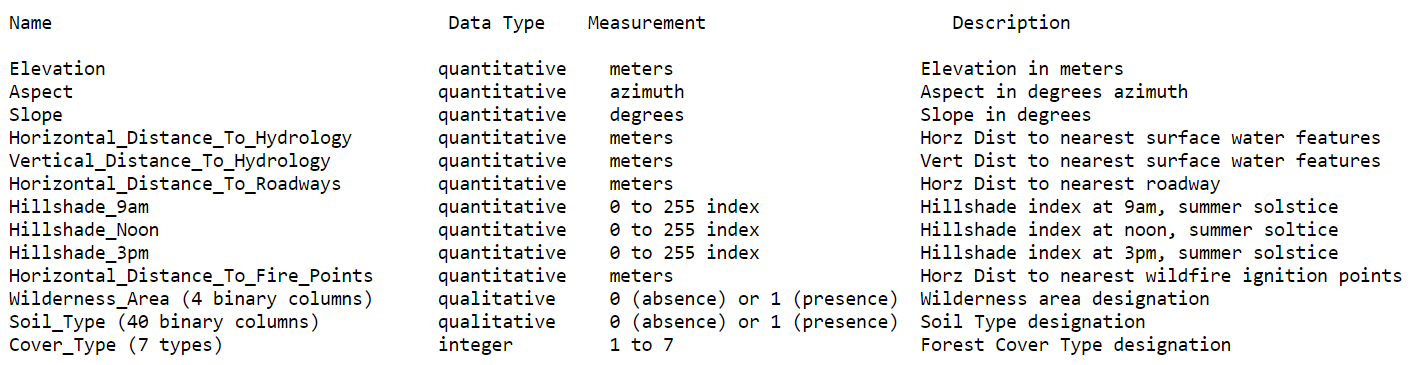


Fig. 1: Feature details [1]

# data analysis

The primary step and important step would understand the data that we are to work with. This will provide a high-level idea about the data set and how it can be used. We will need to clear with the information like the number and types of features, the type of the target feature, how the data is distributed within each feature and many more. This will help us to understand what machine learning model or algorithms and training techniques can be employed to solve our problem. We performed the below activities as part of the initial data analysis to get a better insight of the data.

## Scan for missing values

Missing values are sometimes the nightmares of data scientists. There are various strategies to treat missing values. Some of them are ignore the missing values, exclude any records containing missing values, replace missing values with the mean, or infer missing values from existing values, ignore the records with missing values and many more. Each technique has its own consequence and pros. On analyzing our data set, we could see that we are lucky to get a clean data set with ‘no missing values’. Even the ‘readme’ file in the data set source says so. Hence, there does not exist a need to worry about how they need to be handled.

## Understanding data dimensions and Class distribution

We analyzed the data and used visualizations to understand the distribution of the data across the dataset. A good understanding of how the data is distributed in both the features and labels will provide us with idea of any existing issues in the data which might potentially have a high impact on the learning of the model.

On analyzing the target label distribution, we could see that the data set as an issue of class imbalance.

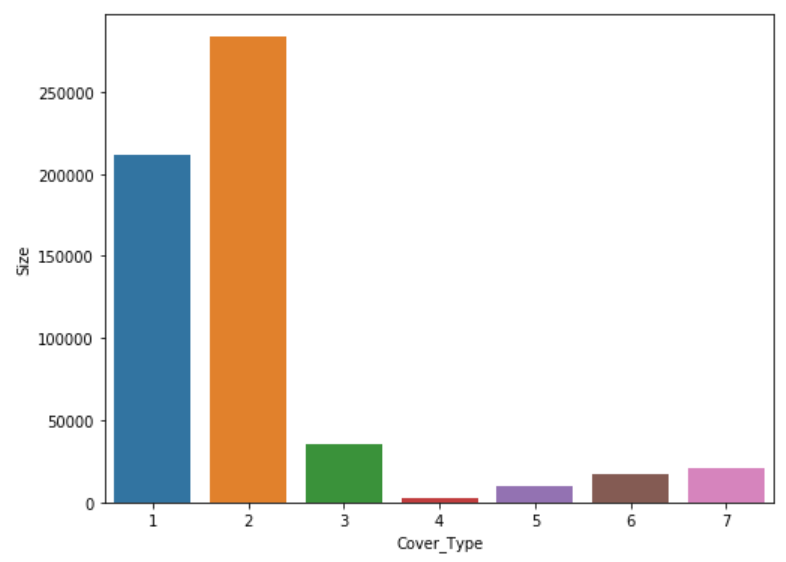


Fig. 2 Class or label distribution – bar graph

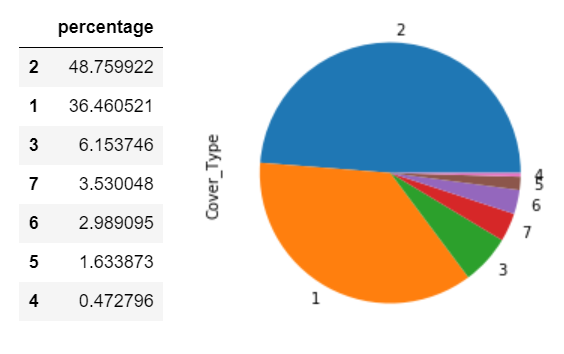


Fig. 3 Class or label distribution as percentage/pie-chart

From the below graphs we could clearly see hat we have an issue of class imbalance. We have 7 classes out of which 2 classes form 85% of the data. There are 2 classes with less than 2% representation. This might lead to the minor classes being predicted poorly even though the accuracy. Therefore, this needs to be handled as part the data pre-processing.

## Finding and visualising skew.

Analogous to how we checked for class imbalance issue in the target variable, we should be checking for any skewed data in the features. Skew will impact the performance of the model. Based on the skew value, we could calculate if our feature is left skewed (negative skew value) or right skewed (positive skew value). The ideal data set would have no skew and the entire graph will be having 0 for all the y-coordinates.

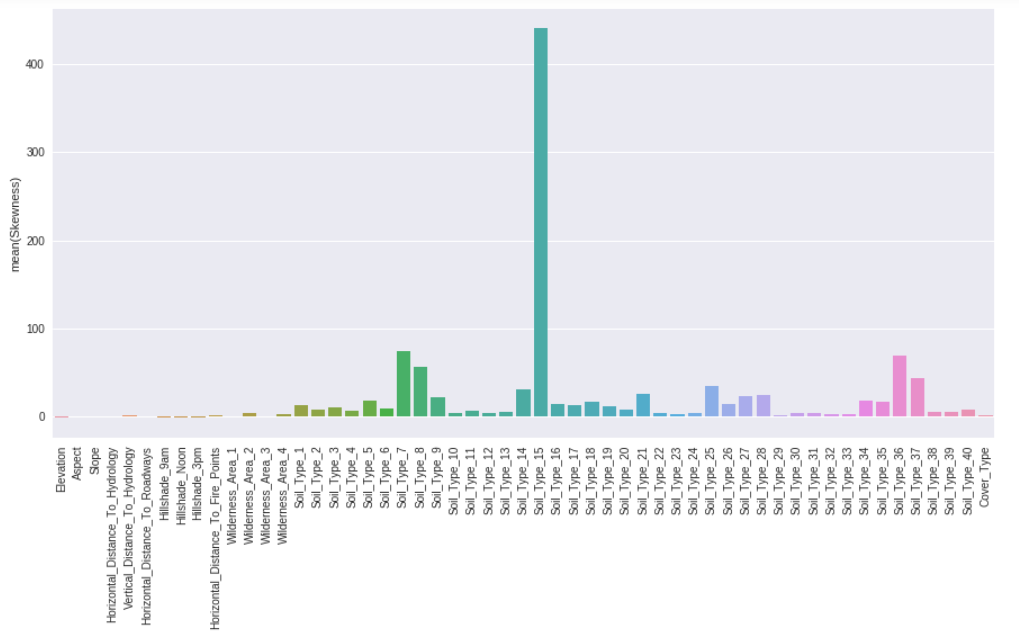


Fig. 4 Skew of all features

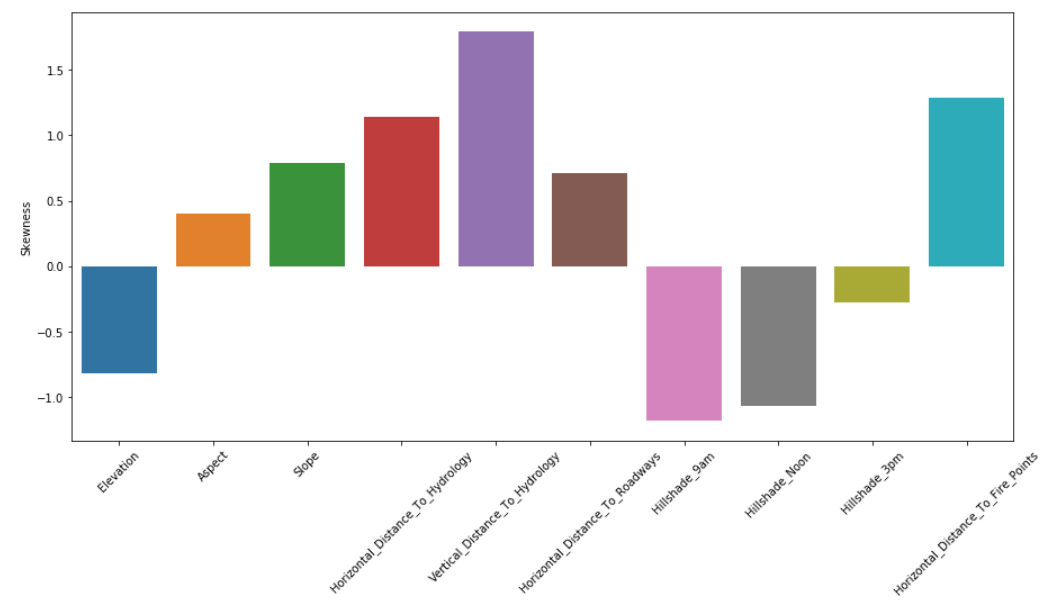


Fig. 5 Skew of features ( having continuous values)

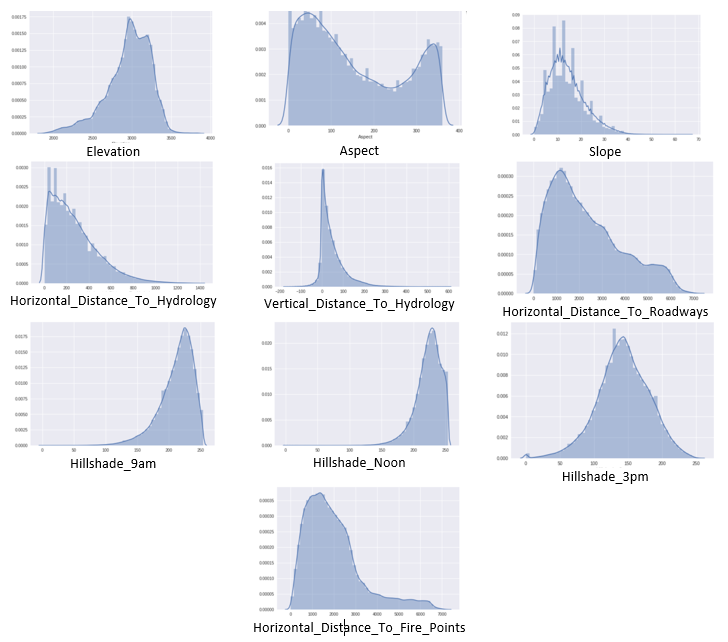


Fig 6 Valued of each feature visualized as graph

Fig. 5 shows the skew of the features that are of non-boolean type and we can infer that they are not very highly skewed. Fig 6 shows the individual visualization of the value distribution of each feature. We have ignored the boolean type features because it makes more sense to not calculate skew on boolean data. We can see how the various data processing techniques affects this existing skew. This will be discussed when we encounter the process of over sampling and under sampling in the preprocessing section.

## Correlation analysis

A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other [3]. Correlation analysis helps to identify the affinity between the features and how much a feature can help in deciding or predicting the class. If the correlation is positive, then the features are directly proportional to each other. If one increases, the other also increases. If the correlation value is negative, then they are inversely proportional to each other. Two features are mutually exclusive and have no impact on each other when their correlation value is zero. Therefore, correlation is a way to understand the relationship between multiple variables and attributes in your dataset. Using Correlation, you can get some insights such as [6]:

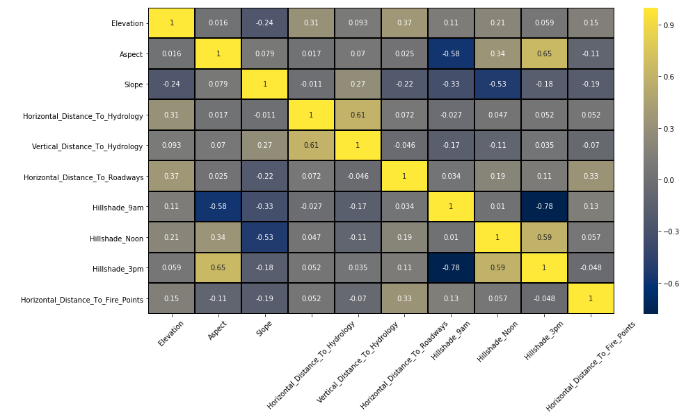
* One or multiple attributes depend on another attribute or a cause for another attribute.
* One or multiple attributes are associated with other attributes.

Fig. 7 Correlation matrix of the continuous features

Two very highly correlated data is not very useful. They do not help the model that much as both give the same information when it comes to building or training the model. Not removing will not cause any harm to the model in majority of the cases, but on the same hand, does not prove to be beneficial too. Hence, one of them could be removed. As a matter of fact, their impact varies from algorithm to algorithm.

Luckily, decision trees and boosted trees algorithms are immune to multicollinearity by nature. Multicollinearity happens when one predictor variable in a multiple regression model can be linearly predicted from the others with a high degree of accuracy. This can lead to skewed or misleading results. When they decide to split, the tree will choose only one of the perfectly correlated features. However, other algorithms like Logistic Regression or Linear Regression are not immune to that problem and you should fix it before training the model [6].

From Fig. 7, we could see how the features are correlated to one another. As part of pre-processing, based on the correlation matrix, we will decide what actions are needed to be performed over the data set.

# data pre-processing

Data pre-processing is the most important part of any data related project. The various pre-processing techniques helps to transform the raw data set into a much better format which can be better processed and understood by the machine learning algorithm. Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model [7].

## One hot encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction [8]. Categorical data are sometimes difficult for the machine learning model to learn and understand. Hence, instead of providing a single feature of categorical type with ‘n’ number of categories, we could convert this n features of binary data. This process of one hot encoding provides the categorical data in a better way for the machine learning model to understand and learn.

In out data set, there are two categorical features namely Wilderness Area and Soil Type. Wilderness area has four categories and Soil type has forty categories. It is seen that the data set already comes in a one hot encoded format where these two categorical features have been converted to four binary features representing wilderness area and forty binary features representing each one representing one of categories of the soil types feature.

However, as part of pre-processing and analysis, we performed a ‘reverse one hot encoding’ just as a temporary step to visualize the how the data is distributed in these two features and how much influence each category has on the target ‘variable forest cover type’.

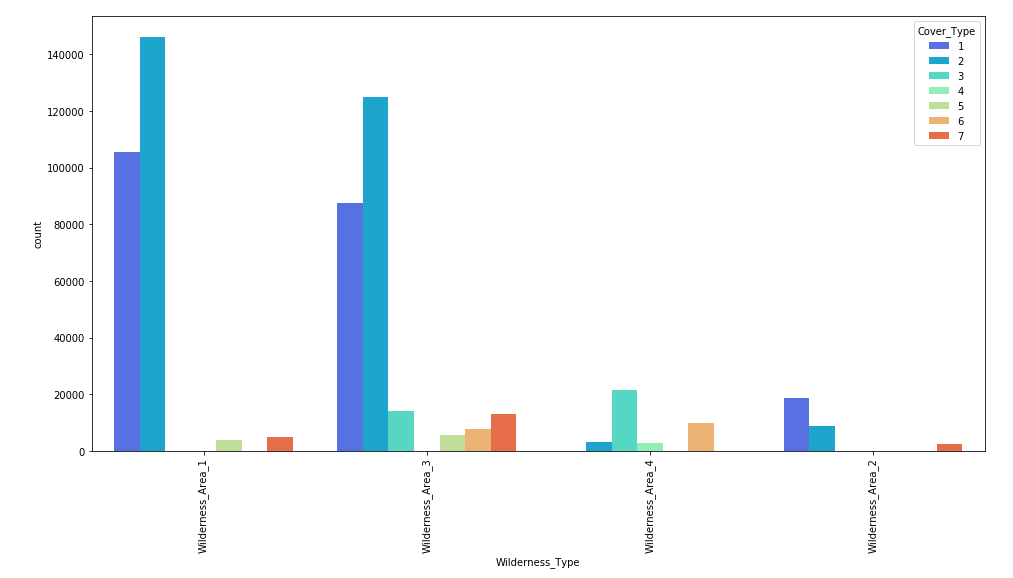


Fig. 8 Visualization of reverse one-hot encoded wilderness area and forest cover type distribution over each category.

The above graph provides an idea or insight of how the forest covers are distributed based on the wilderness area. It is evident that more instances have the wilderness area 1 and wilderness area 3 and wilderness area 2 is very sparsely found in the data set. We can also see the different cover types which match with each of the wilderness area.

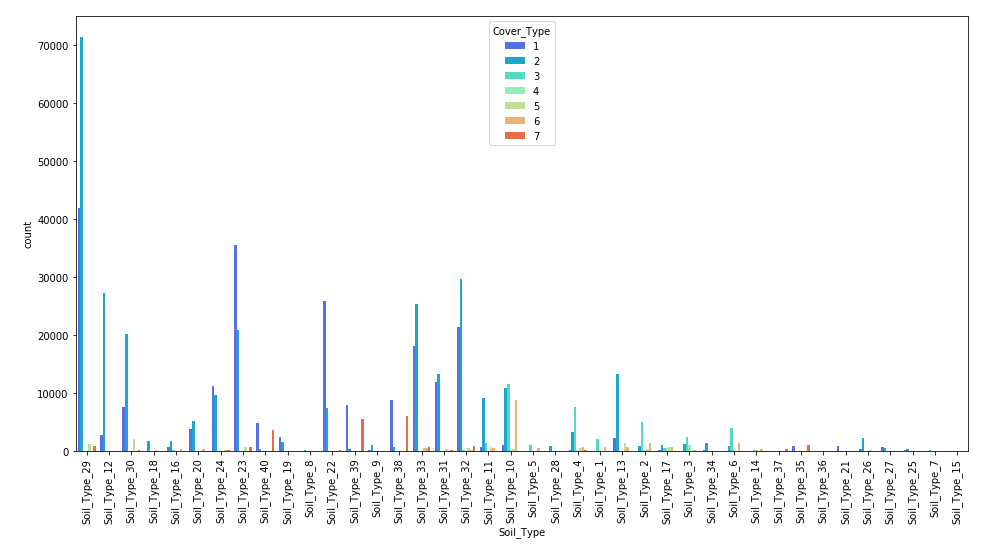


Fig. 9 Visualization of reverse one-hot encoded soil type and forest cover type distribution over each category

This graph provides the same insight which we saw earlier but this time, with soil type data. Here, we can infer from the visualization that the data is not evenly distributed amongst the soil types. The soil types towards the right side of the graph have comparatively lesser number of instances. Again, it also shows how the forest cover type is distributed amongst the various soil types. These inferences will provide us a better understanding of the dataset. Though reverse one hot encoding helped us to get a better insight of the two features, the one hot encoded format will be better for the machine learning model to learn from.

## Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges [9].

For example, consider a data set containing two features, age(x1), and income(x2). Where age ranges from 0–100, while income ranges from 0–20,000 and higher. Income is about 1,000 times larger than age and ranges from 20,000–500,000. So, these two features are in very different ranges. The main aim of normalization is to bring the features of the data set into a common scale so that it will be easier and better for the machine learning algorithm to learn from the data. Below is a snapshot of the data set before normalization. It can be noted that the feature values are in various range.

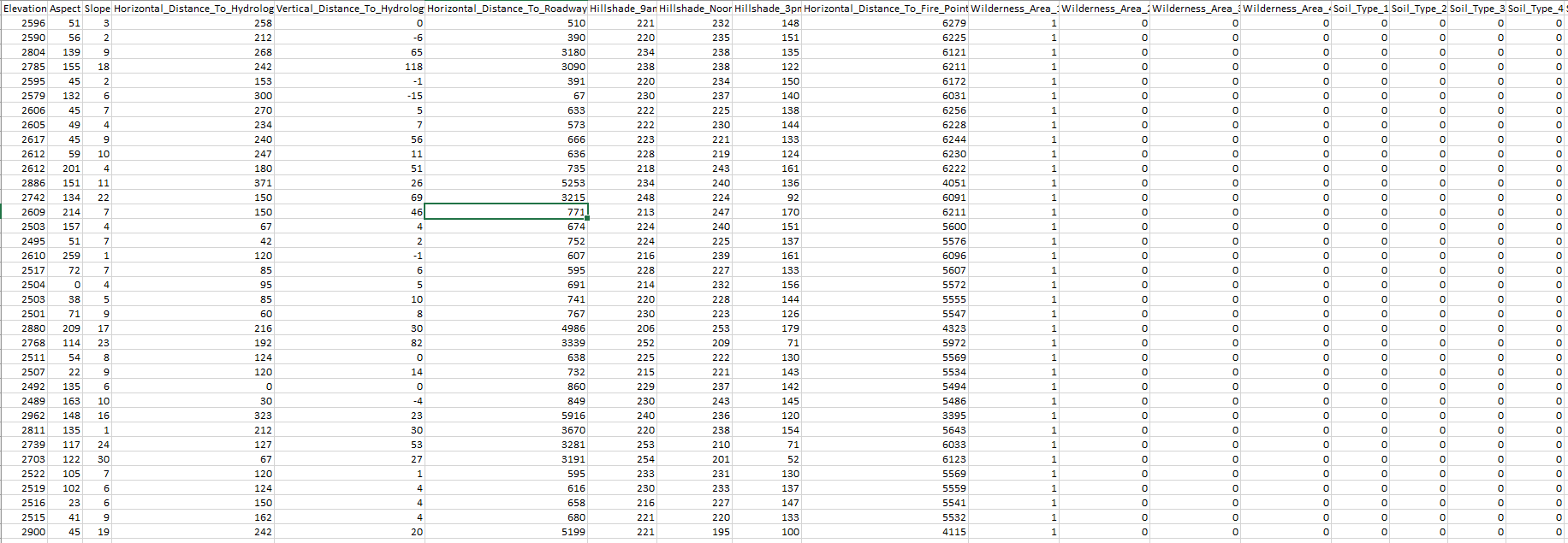
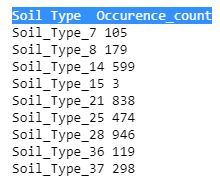


Fig. 10 Snap shot of the data set before normalization

There are so many normalization techniques are there namely Min-Max normalization, Z-score normalization and Decimal scaling normalization. We chose to perform min-max normalization and convert all the features to a range of 0 to 1. The formula for Min-Max normalization is as follows.

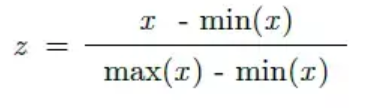


Fig. 11 Formula for min-max normalization

Where min and max are the chosen range of the feature, x is the feature value and z is the normalized feature value. Normally, we can set the range of a feature. But for normalizing our data set, we have used MinMaxScaler() function of Scikit learn. This automatically takes the min and max valued for the values of the feature in the data set and preforms the normalisation.

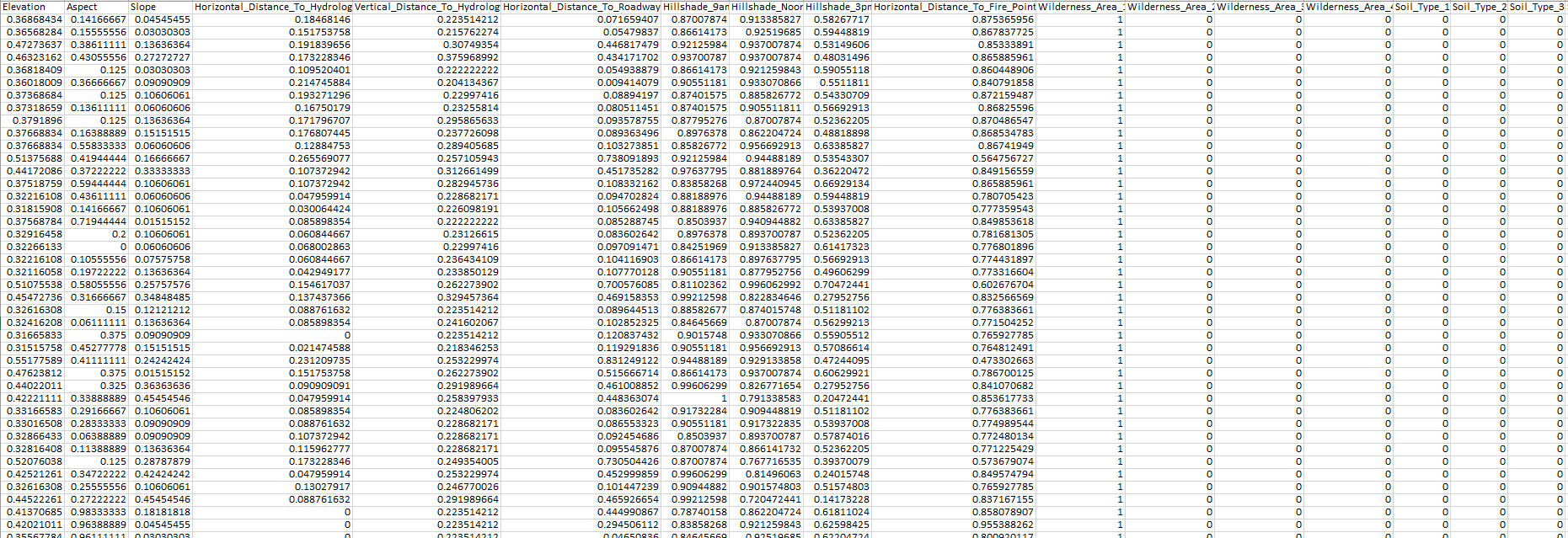


Fig. 12 Snap shot of the normalized data set

## Dimensionality reduction

In a feature set, not all the features will have the same degree of usefulness. The features that do not contribute much to solve the problem actually make the model less efficient as they increase the processing time and storage space required for training the model. This is called as the curse of dimensionality. Hence, we need to consider the value added by the additional dimension before including it into the feature space. This is a famous situation in machine learning or statistics known as ‘The Curse of Dimensionality’.

There are various ways by which we can perform dimensionality reduction. Lets us see them case by case.

1. Based on correlation analysis: In Fig 7, we see the correlation matrix. Based on this, when two features, have very high correlation, we can remove one of them. In the matrix, hillshade\_3pm and hillshade\_9am have the highest correlation of (-0.78) and hence we could remove one of features.
2. Based on count: Let us take the soil types feature. After one hot encoding, we can see that the data is sparse with 0 as the majority value of the soil type feature. Here, if a particular soil type has only less number of positive instances, we can remove that feature. For our experiment, we have set 1000 (which is less than 2% of the size of our data set) as the threshold. For feature have less than 1000 instances, it is better to remove them so that dimensionality will get reduced. Therefore, we could remove the below features from the feature set.

Fig. 13 Features with less than 1000 positive instances

## Sampling

Class imbalance is a major issue that exists in most of datasets. These imbalances datasets can be fixed to a certain extent by the process of sampling. The definition of imbalanced data is straightforward. A dataset is imbalanced if at least one of the classes constitutes only a very small minority. The issue of class imbalance can result in a serious bias towards the majority class, reducing the classification performance and increasing the number of false negatives. How can we alleviate the issue? The most commonly used techniques are data resampling either under-sampling the majority class, or over-sampling the minority class, or a mix of both [10].

As we noticed during the data analysis phase, our target feature has an issue of class imbalance. This can be resolved to an extent by doing oversampling or under sampling.

### Oversampling: When one class of data is the underrepresented minority class in the data sample, over sampling techniques maybe used to duplicate these results for a more balanced amount of positive results in training. Over sampling is used when the amount of data collected is insufficient. A popular over sampling technique is SMOTE (Synthetic Minority Over-sampling Technique), which creates synthetic samples by randomly sampling the characteristics from occurrences in the minority class[12]. The initial size of the dataset was 581012. After running SMOTE, the oversampled data set size was observed to increase to 861566.

### Undersampling : Conversely, if a class of data is the overrepresented majority class, under sampling may be used to balance it with the minority class [12]. Under sampling is used when the amount of collected data is sufficient. We use ‘RandomUnderSampler()’ of imblearn to perform the undersampling. After running SMOTE, the undersampled data set size was observed to decrease to 19229.

Also, sampling also impacts the skew of the categorical features which we noticed in Fig 5. Please notice that sampling has reduced the skew of certain features. It can also be noted that both undersampling and oversampling have impacted the skew almost similarly and hence the graphs in the Fig 15 and Fig 16 look identical.

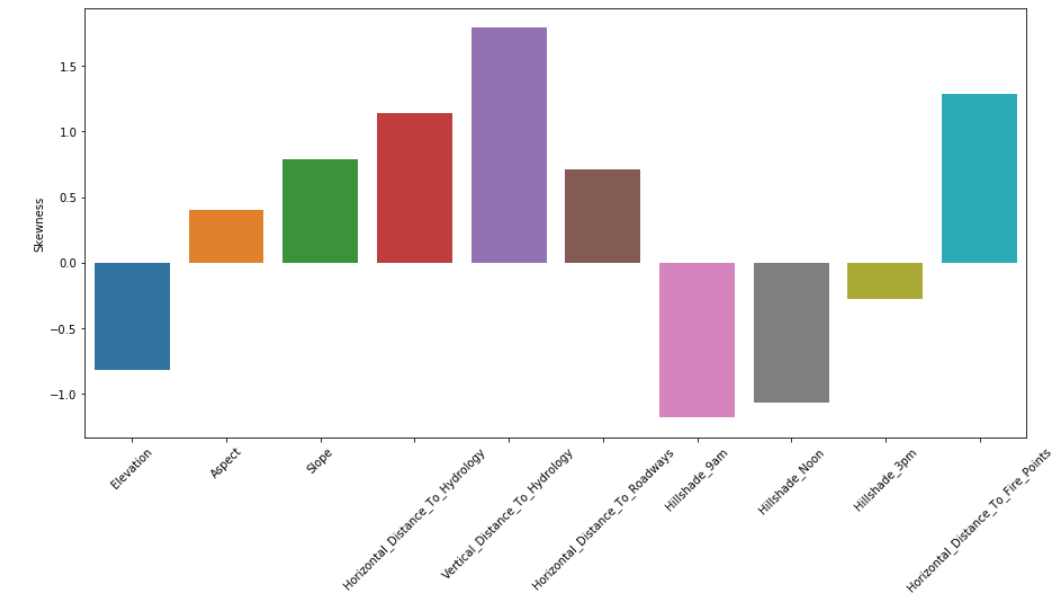


Fig. 14 Skew of features (having continuous values) -same as Fig 5

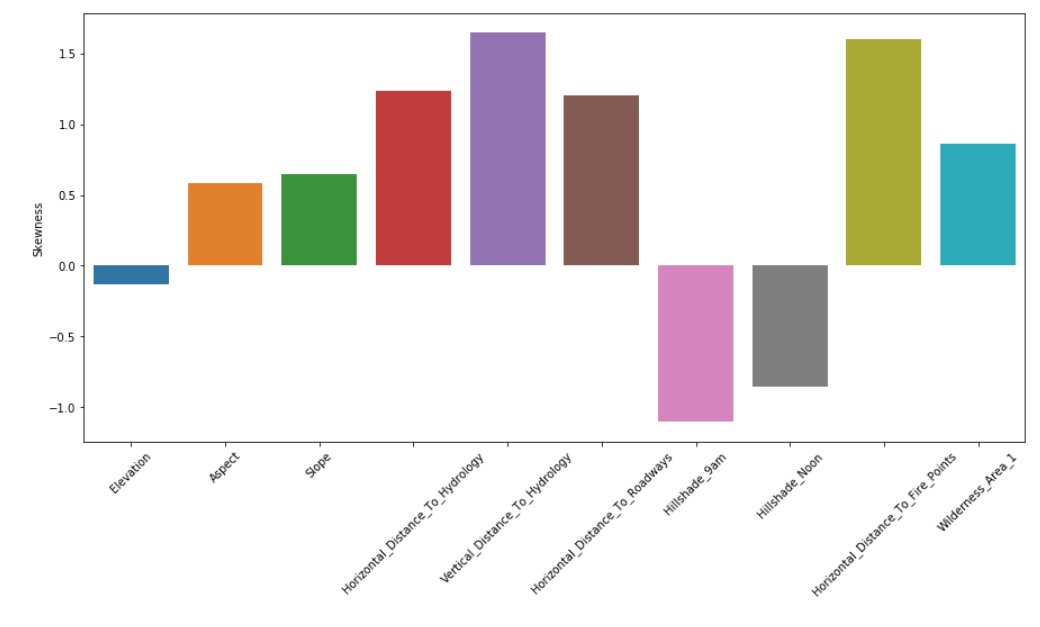


Fig. 15 Skew of features after oversampling

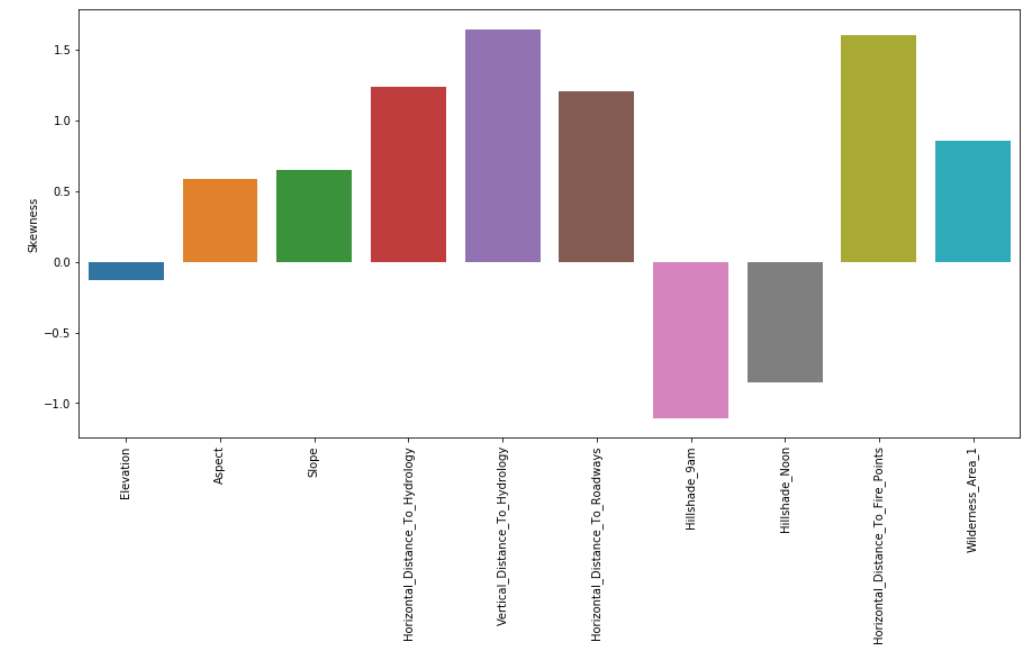


Fig. 16 Skew of features after undersampling

# Model Construction

Once the pre-processing of the data is completed, the next

step is to construct the machine learning model. We will choose the algorithm and the training technique based on the dataset and the problem that needs to be solved. Just to re-affirm, the problem at hand is a multi-class classification problem where our independent variables are continuous in nature and dependent variable is in categorical form. As a part of this project, we ran the pre-processed dataset against models belonging to each type and obtained the results as below. (Detailed explanation of each algorithm follows the image)

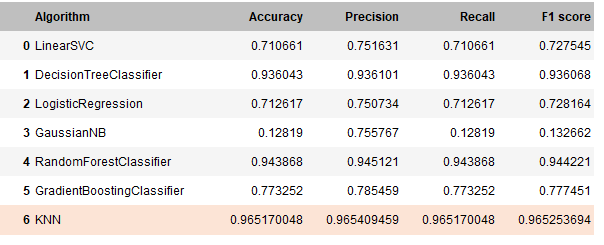


Fig. 17 Performance metrics of each algorithm against the pre-processed data

As per the above results obtained, we found that K-Nearest Neighbors (KNN) to have the best results for a standalone algorithm. The later part uses various techniques like K-fold and Ensembles using the above algorithm(s) which potentially might provide better results. At this moment, we identified KNN to have the best performance metrics for the forest cover data set closely followed by Random Forest Classifier and Decision Tree Classifier.

## Linear Models:

In machine learning linear classifiers are any model in which there is a single hypothesis function which maps between model inputs and predicted outputs.At the core of any linear model is a dot product between the input example and the parameter / weight vector. In the case of linear regression this is the entire hypothesis function. Where as logistic regression feeds the dot product through a sigmoid function such that the output is between 0 and 1 and hence is suitable for binary classification problems [13]. Now, any model which makes sure that its learning a linear transform on data to get output is linear model. We have chosen the below liner models for training the data.

* 1. Linear SVC (Support Vector Classifier)

Linear SVC is a more efficient flavor of SVM for classification problems. The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. We chose to use this flavor of SVM because Liner SVC uses ‘one vs rest’ classifier training technique. Below is the confusion matrix obtained on running the data on the Linear SVM classifier.

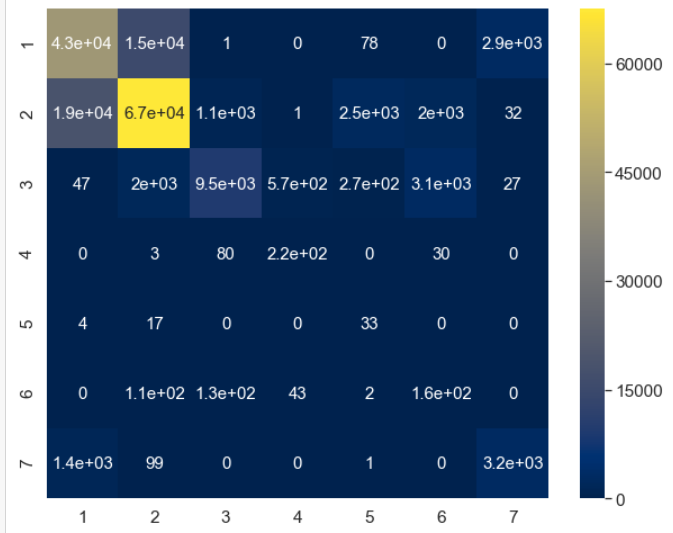


Fig. 18 Linear SVC Confusion Matrix

As we can see in the performance metrics Linear SVC provided only mediocre results (Accuracy=0.71) for our dataset.

* 1. Logistic Regression

Logistic Regression uses a logistic function or sigmoid function that helps to handle the outliers in the classification problems effectively. Logistic regression was chosen over liner regression since linear regression will not work when the target variable is categorical. Logistic regression is also known to be very effective for two class classification problems. However, we wanted to check how it deals with multi class classification. Unsurprisingly, the results were not very impressive as it could provide an accuracy of just 0.712.

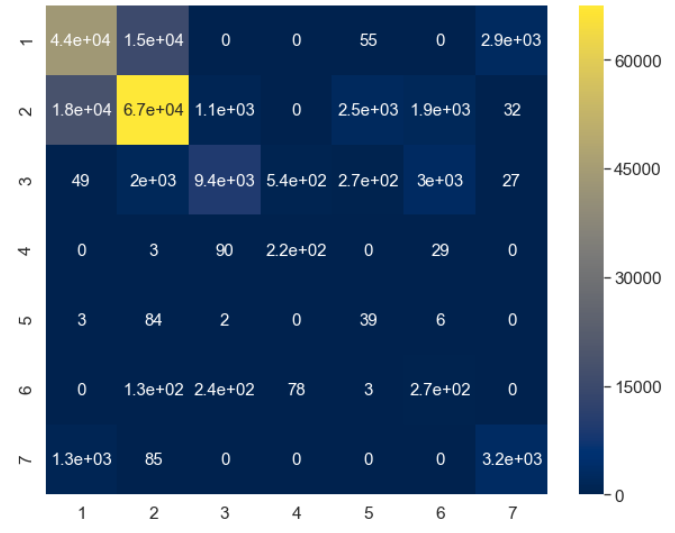


Fig. 19 Logistic regression Confusion Matrix

* 1. Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It’s specifically used when the features have continuous values. It’s also assumed that all the features are following a gaussian distribution (normal distribution). Bayes’ theorem is based on conditional probability. The conditional probability helps us calculating the probability that something will happen, given that something else has already happened. The naive Bayes classifier assumes all the features are independent to each other. Even if the features depend on each other or upon the existence of the other features. [16].

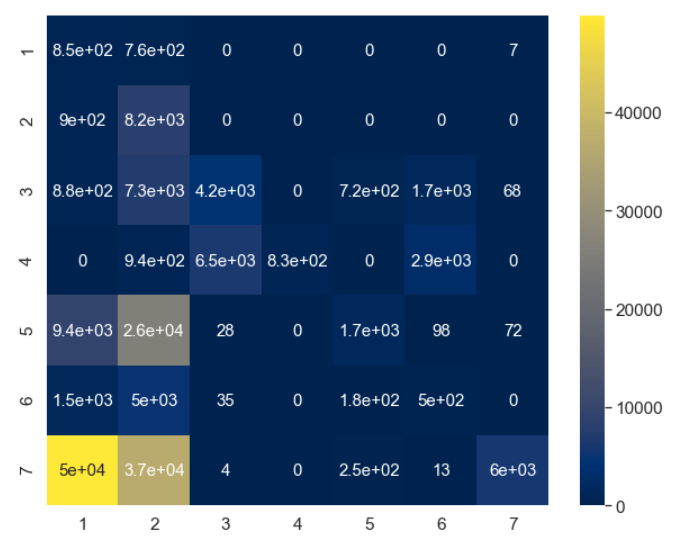


Fig. 20 Gaussian NB Confusion Matrix

The Gaussian NB classifier turned out to provide very poor accuracy of 0.12, which is not even up to the standards of random guessing. This might be since our data is skewed and is not exactly a normal distribution. Also, our features are correlated. Even from the confusion matrix we can see that majority of the data does not align to the diagonal which is justifies the poor accuracy. Hence Naive Bayes algorithm is clearly not suitable for our problem.

### Tree based Models:

1. Decision Tree

A decision tree is used to classify future observations given a body of already labeled observations. The tree begins with a root (where we still have all our observations) then comes a series of branches whose intersections are called nodes and ends are called leaves, each corresponding to one of the classes to predict. The depth of the tree refers to the maximum number of nodes before reaching a leaf. Each node of the tree represents a rule. To browse the tree is to check a series of rules. The tree is constructed in such a way that each node corresponds to the rule that best divides the set of initial observations (variable and threshold) [17]. We know that decision trees work well for classification problems involving either numeric or categorical data. Since our data set falls under the same category, we decided to build a Decision tree model and as expected, the model turned out to give a good accuracy of 0.93. The confusion matrix obtained for the Decision tree model is given below.

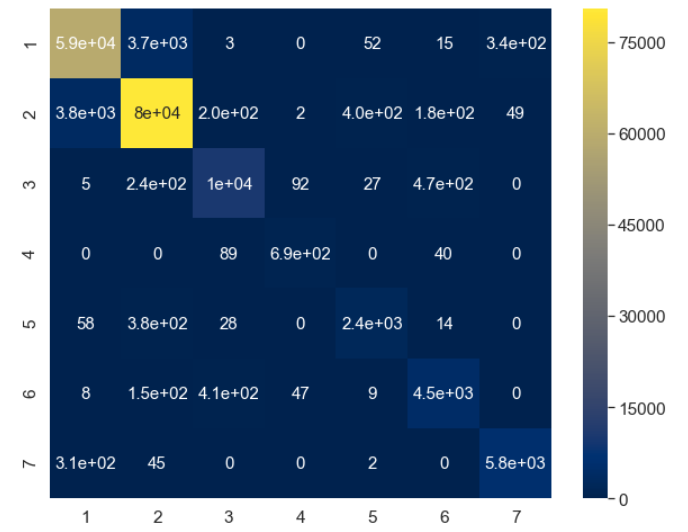


Fig. 21 Decision Tree Confusion Matrix

1. Random Forest

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it’s simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it. As the name might suggest, the random forest algorithm is based on a multitude of decision trees [18].

Here are the main steps: 1. We take a number X of observations from the starting dataset. 2. We take a number K of the M variables available (features). 3. We create a decision tree on this dataset. 4. Steps 1 to 4 are repeated N times so as to obtain N trees. To obtain the class of a new observation we go down the N trees. Each tree will predict a different class. Random forests can be used in ensemble technique. We will be discussing more on this in the ensembles part [17].

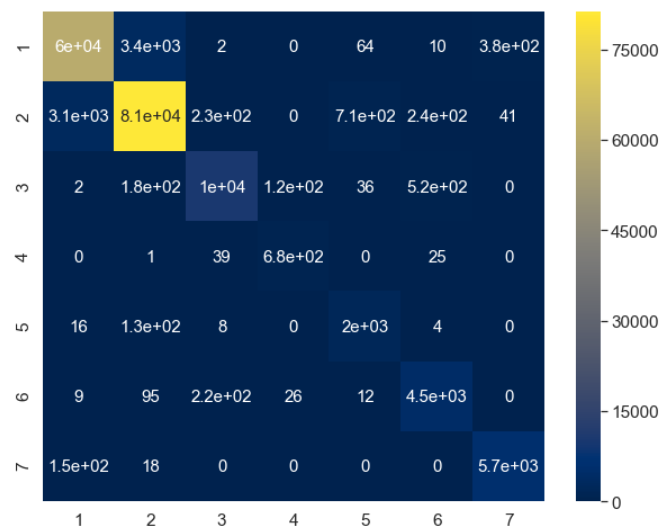


Fig. 22 Random Forest Confusion Matrix

However, from Fig. 17, we can see that Random Forests provide an accuracy of 0.94 which is good when compares to that of linear models. From the confusion matrix, we can see that diagonal population is high; which is good for the accuracy of the model.

1. *Distance-Based Model:*

Distance based methods use the same parametric model for the substitutions and deduce from these rates `evolutionary' distances between units. The distance matrix is then analyzed by hierarchical clustering type methods such as neighbor-joining (single linkage clustering) or unweighted pair-group with arithmetic mean (average clustering). Distance-based methods can be seen as intermediary containing both parametric and non-parametric components [19].

* 1. KNN (K- Nearest Neighbors)

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. To select the ‘k’ value, we could use the ‘Grid search’ technique which is used to find the optimum hyper-parameter valued of an algorithm for the data set under consideration. This will be discussed in detail in the evaluation techniques section. We can also simply try training the model for various valued of k and check their performance metrics directly to make the decision. The below graph shows the visualization of the various accuracy value obtained for various valued of k. From this it is clear that k=5 has the best accuracies for the train and test scenarios. Even though the values for k=1 and k=3 are higher, there is a huge gap between their test and train accuracies which is not good. Hence, we could say that k=5 is the better hyper parameter values.

When compared to the other algorithms, we go the best performance metrics for the distance-based model. We were able to achieve an accuracy of 0.965. The precision and recall values also were observed to be healthy.

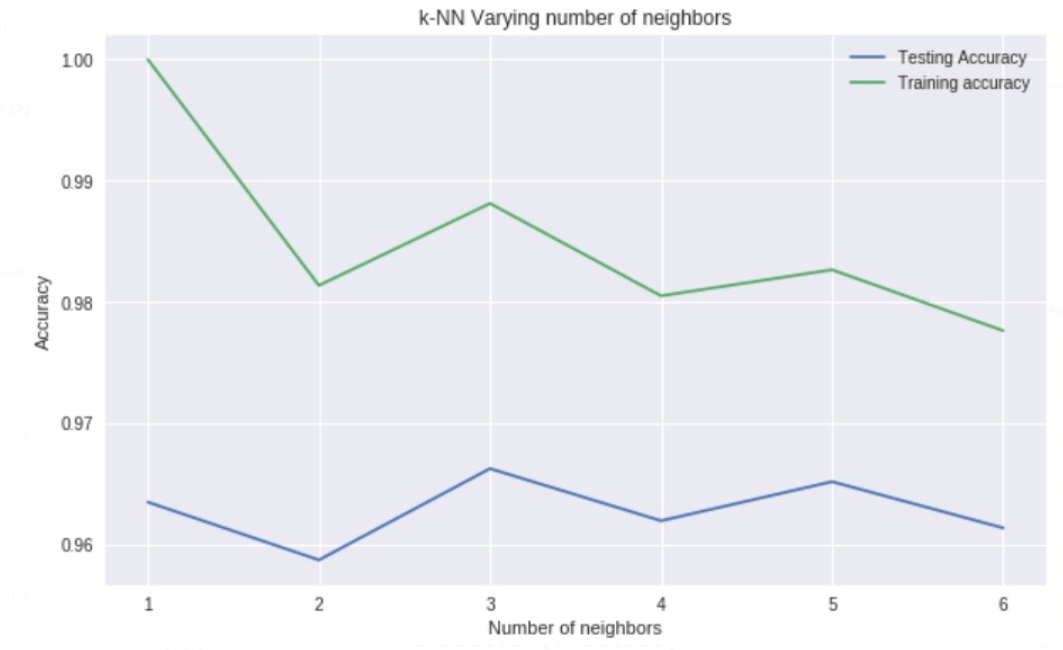
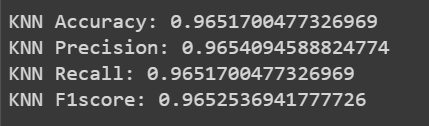


Fig. 23 Comparing accuracy for various K values



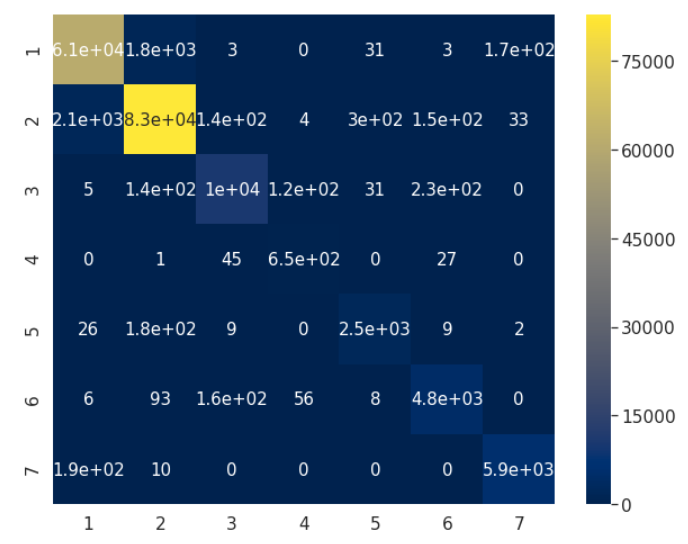
Fig. 24 KNN Performance metrics

Fig. 25 Confusion Matrix for KNN

The above confusion matrix shows that diagonal has higher numbers which explains the high accuracy. We re-ran the KNN model using the oversampled data (data was oversampled using SMOTE) and the observation was that the performance of the model slightly decreased. Below are the performance metrics and confusion matrix. Similarly, a model trained with under-sampled dataset seems to have much less performance. Performance metrics and confusion matrix are given below.

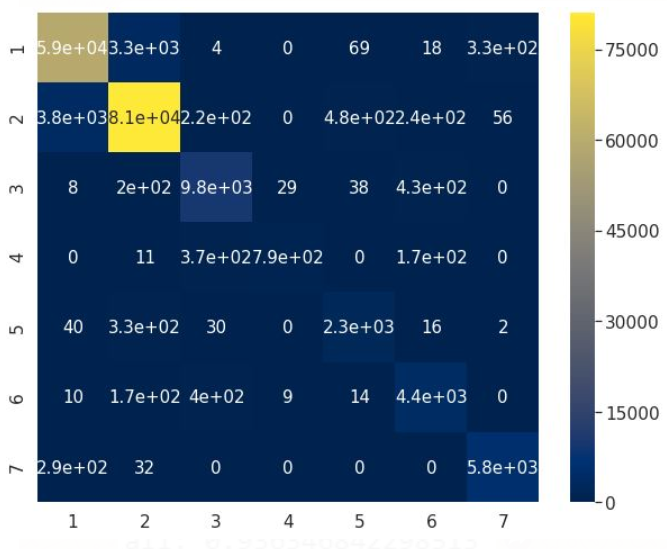


Fig. 26 KNN Confusion Matrix after SMOTE

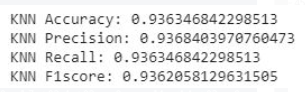


Fig. 27 KNN Performance Metrics after SMOTE

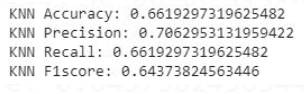


Fig. 28 KNN Performance Metrics after Undersampling

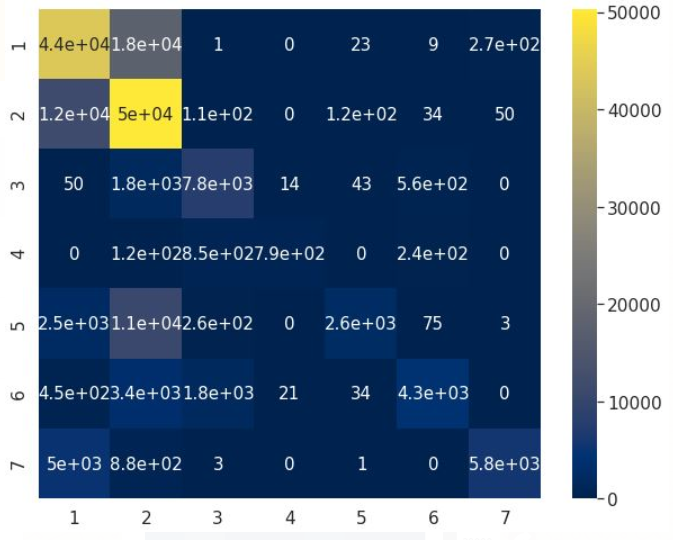


Fig. 29 KNN Confusion Matrix after Undersampling

1. *Rule based model:*
   1. Apriori Algorithm

We found a code in github which is a Python Implementation of Apriori Algorithm for finding Frequent sets and Association Rules. The advantage is that it does not require much manipulation of the data and can create association rules from a .csv file directly. The code can be found in the link given in the reference [20]. The disadvantage of using this code implementation is that the rule are ambiguous for continuous numeric data especially because the output does not give any information about the feature while displaying the rules. This is okay for categorical features but not useful for numeric continuous features. A future work on this would be to improve the implementation to present the rule with metadata which helps to understand the rule in a clearer manner.

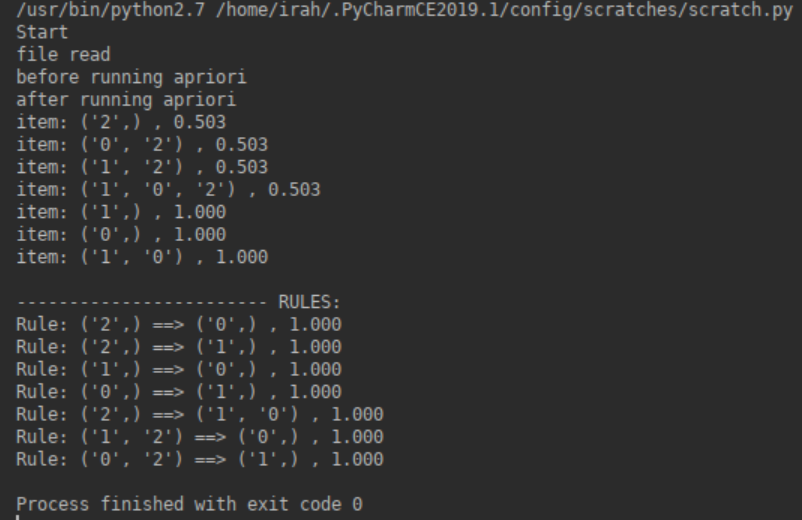


Fig. 30 Output of feeding the data set to Apriori implementation

* 1. Dummy Classifier

The dummy classifier gives you a measure of “baseline” performance — i.e. the success rate one should expect to achieve even if simply guessing. Therefore, we can say tat our models should at least have an accuracy of the dummy classifier as it is nothing, but the accuracy achieved by random guessing. A major motivation for Dummy Classifier is F-score, when the positive class is in minority (i.e. imbalanced classes). This classifier is used for sanity test of actual classifier. Dummy classifier completely ignores the input data. In case of 'most frequent' method, it checks the occurrence of most frequent label [21].

Dummy Classifier Accuracy: 0.487

1. *Ensembles:*

Ensembles or otherwise known as model ensembles is nothing but a combination of models. It is based on a simple concept that two models together can perform better than a single model. Bagging, Voting and Boosting are some of the ensembling techniques. Boosting technique is proven to reduce the bias and mostly used liner models whereas, Bagging technique is proven to reduce variance and hence mostly applied on tree- based models.

* 1. Voting

Voting classifier model combines multiple different models (i.e., sub-estimators) into a single model, which is (ideally) stronger than any of the individual models alone.

We specify the Voting Classifier as a list of (name, sub-estimator) tuples. Fitting the Voting Classifier on the data fits each of the sub-estimators in turn. We set then jobs argument to be -1, which instructs sklearn to use all available cores In hard voting, the voting classifier counts the number of each class-instance and then assigns to a test-instance a class that was voted by majority of the classifiers. In soft computing, there is a probability term coming that takes the average of probabilities for each class and then uses it to classify the test instance [22].

In our case, we have built a ensemble that uses voting technique and it uses KNN and Random forest models. We have selected these two models because they had the best performance amongst the tested algorithms or models. The accuracy of this voting ensemble was found to be 0.9471.

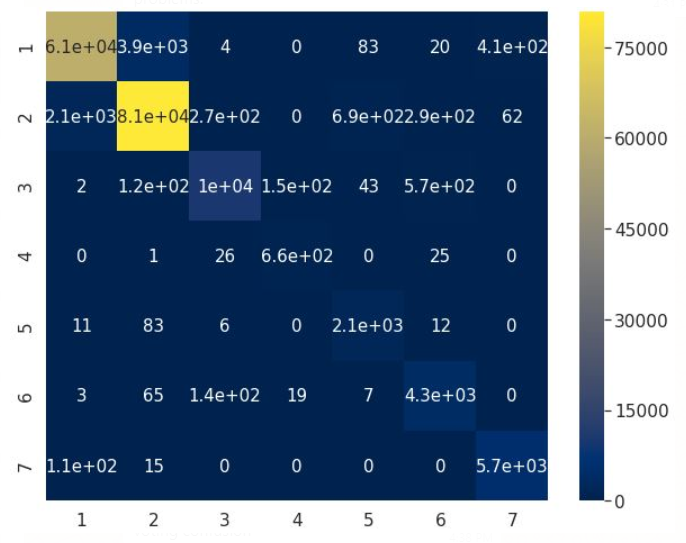


Fig. 31 Confusion Matrix for gradient boosting

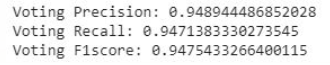


Fig. 32 Performance metrics for Voting

* 1. Bagging

Bagging is shorthand for the combination of bootstrapping and aggregating. Bootstrapping is a method to help decrease the variance of the classifier and reduce overfitting, by resampling data from the training set with the same cardinality as the original set. The model created should be less overfitted than a single individual model. Bagging is an effective method when you have limited data, and by using samples you’re able to get an estimate by aggregating the scores over many samples. The simplest approach with bagging is to use a couple of small subsamples and bag them [23].

We all know that bagging is a variance reduction technique. Since tree-based models have high variance, we have used decision tree to construct this ensemble. The bagging ensemble gave an accuracy of 0.9590 which is greater than the accuracy we obtained for the normal decision tree (accuracy =0.9354). Hence, ensembles has increased the performance of the classifier.

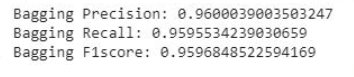


Fig. 33 Performance metrics for Voting

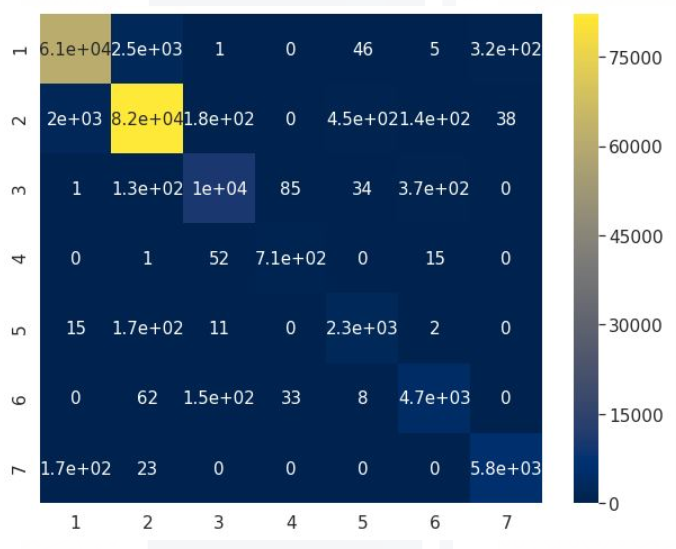


Fig.34 Confusion Matrix for gradient Bagging

* 1. Gradient Boosting algorithm

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. The XGBoost library implements the gradient boosting decision tree algorithm. This approach supports both regression and classification predictive modeling problems. The accuracy obtained is 0.77 and is not very effective.

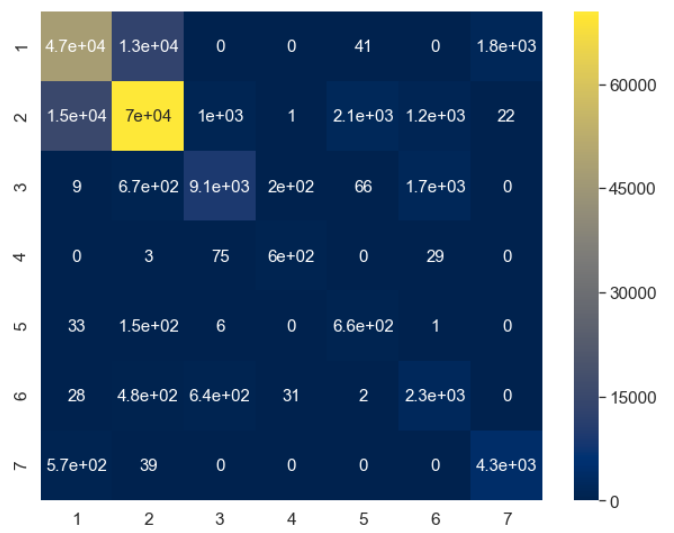


Fig. 35 Confusion Matrix for gradient boosting

# Evaluation Techniques

1. *Test-train split*

The concept of test train split is of the techniques that is used in machine learning process to partition the data set into training and testing data. This can easily be done by passing the pandas data frame to the train\_test\_split() function or library of scikit learn. We would need to provide the ratio in which the data set needs to be split. As a thumb rule or most commonly followed practice, we have used 70:30 as the ratio for splitting the dataset into train and test datasets. Similarly, we can try building the models for other ratios and observe the performance.

The test train split also provides an option to specify something called ‘random state’, which is nothing but a mere integer. This helps to ensure that that every time the dataset is split, we do not get different sets of train and test data. Below is the count of the test and train datasets after splitting.

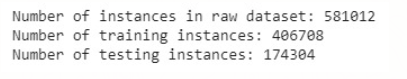


Fig.36 Instance count in test and training datasets

1. *K-fold cross validation*

K-fold cross validation is a famous evaluation technique used in machine learning. Based on the value of k, the dataset will be divided into k different partitions. K-1 parts will be used for training and the one partition left out will be used as a test data set. This is repeated for k times and the test dataset will be different for every kth time. It is mainly used to split the dataset into k combinations of train and test data and build a model k times with the obtained k different data. The average of k different models will be taken and finally used to get the average accuracy which will be the accuracy of the final model.

We can compare the average accuracy obtained using the k-fold process with the accuracy obtained in test train split. If the value is more less equal, then it means that we have used the correct value for train test split. Additionally, we can also use various value of k and re-run the test train split for a better and deeper analysis.

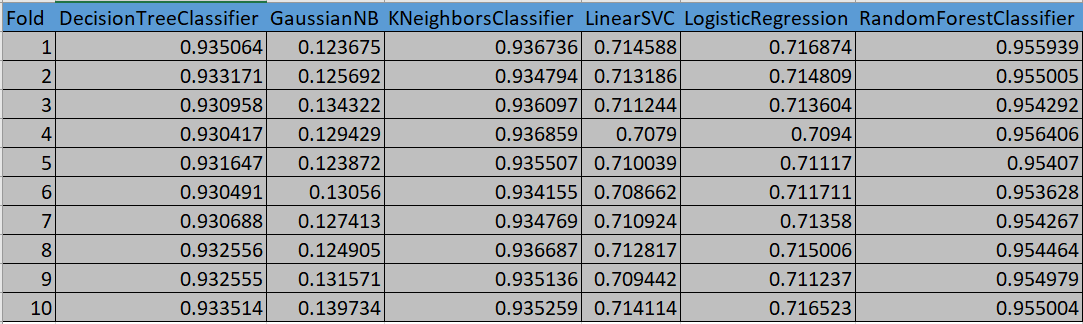


Fig. 37 10- Fold Values obtained for various algorithms

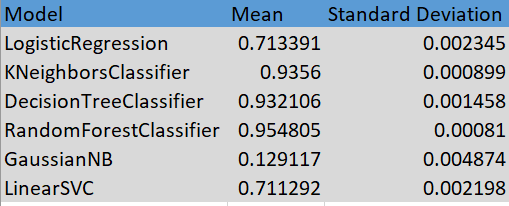


Fig. 38 Average and standard deviation from K-fold process

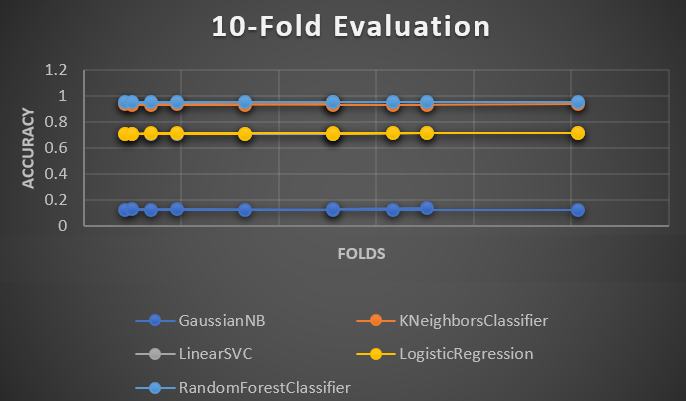


Fig. 39 Visualization 10-Fold values of algorithms

1. *Grid Search*

Grid search is the technique which is used to find out the best values for the hyper parameters of the various models used. Normally, hyper parameters are manually set by the user during the model construction. This might be sometimes error prone depending on the expertise of the individual. By using the grid search, we have tried to identify the best hyper parameter value of K in K-Nearest neighbor and N ( tree sample size or n\_estimators) value in Random Forest. We found that K=5 and N=3 are the best hyperparameter estimates for the respective algorithms for the forest cover type. A limitation of this technique is that it requires lot of processing power and time.

1. *Performance Measures*

For all the models or machine learning algorithms we have calculated and recorded various performance metrics like Accuracy, Recall, Precision, F1 Score. This would greatly help us to compare various models based the numbers obtained. We have seen that the KNN model has the best accuracy for a stand-alone model when trained with the forest cover type dataset. However, since the dataset has the issue of class imbalance, we also need to check the other parameters like Precision and Recall. This was also considered as one of the factors to finalize that KNN is the best stand-alone model.

We also calculated the Confusion Matrix or Contigency Table for all the models. This helps to understand the no. of instances that are correctly classified and the number of ones that are not. If a model has a heavy leading diagonal, then it means that the accuracy is high. Apparently, the confusion matrix of KNN model seems to the best of all. These are the various performance measures that we used to evaluate the best model for our machine learning problem.

1. *T-TEST EVALUATION*

We performed the t-test evaluation for KNN and Random Forest models (since they had the two highest accuracies). The accuracies obtained from the k-fold for the two algorithms were provided as input the scikit learn t-test function and below is the obtained ‘p’ and ‘t’ values. From the below we can understand that for alpha=0.05, there is a statistical significance.

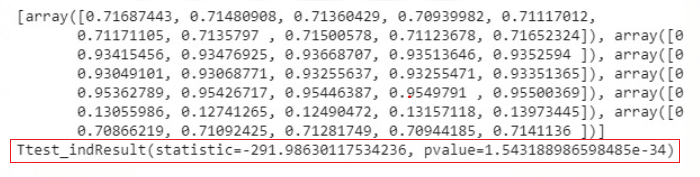
**

Fig. 40 T-test evaluation results

# Observations, Insights and Lessons learnt

1. Sampling does not always increase the accuracy

Since our dataset had the problem of class imbalance, we decided to perform sampling techniques to handle it. But it was observed that the performance of the models did not increase after the sampling was done. Below is the performance metrics of the models for the unsampled data.

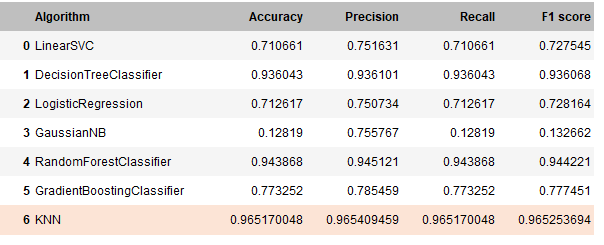


Fig. 41 Performance metrics for unsampled data

After oversampling, we observed that the accuracy and other performance metrics slightly decreased for all the models. Below is the performance metrics of models trained with over-sampled data (SMOTE).

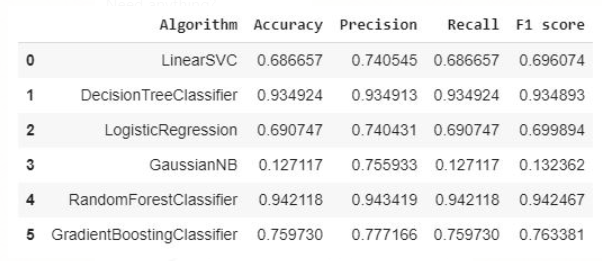


Fig. 42 Performance metrics after SMOTE

After undersampling, we observed that the accuracy and other performance metrics significantly decreased for all the models. Below is the performance metrics of models trained with under-sampled data.

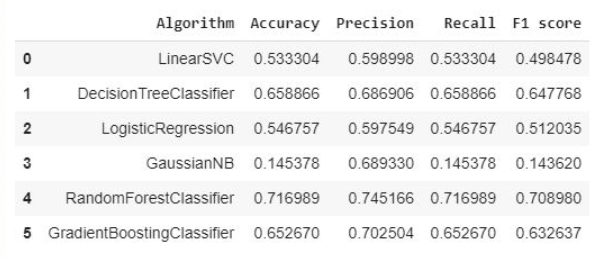


Fig. 43 Performance metrics after undersampling

Hence, we observe that sampling always doesn’t improve the accuracy of our model.

1. Gaussian Naïve Bases Classifier works terribly with our data set.

We could observe in Fig. 40 that the accuracy of Gaussian NB is poor (0.12). This is worse than random guessing (0.47 for our data set as per Dummy classifier results). This might be due the fact that Gaussian NB needs normalized and un-correlated feature data. Since our dataset is not normalized and have highly correlated features, we can assume it as the reason for the poor performance of the model.

1. Voting Ensemble model’s accuracy is less than stand-alone KNN model

We performed voting ensembling using KNN and Random Forest (the models which gave best results when constructed as stand-alone) by expecting that ensembling would give higher accuracy. But, on contrary to our expectations, the ensemble’s accuracy (0.9471) was lesser than that of KNN (0.9651). This was a very interesting observation.

1. Bagging (Decision Tree) ensemble model’s accuracy was higher than that of stand-alone Decision Tree model

The accuracy which we obtained for bagging ensemble (0.9354) was slightly higher than that of the stand-alone Decision Tree model (0.9340). Therefore, ensembling does increase the accuracy of the model in this case.

1. Choosing KNN model’s hyper-parameter

We could see form Fig. 23 that the train set accuracies of the KNN classifier for k=3 and k=1 are higher than that of k=5. But, we observe that their test set accuracies differ much from the trains set accuracies. Whereas, for k=5, they are both closer and there is not much difference. Hence should be chose k=5 as the hyper parameter.

# Conclusion and Future Works

Finally, after all the analyses, experimental evaluation and statistical testing, we can say that K-Nearest Neighbors provided the best performance amongst the chosen set of Machine learning algorithms for the forest cover type dataset.

As a future part of this analyses, we could try Deep learning algorithms or models and check if they provide better performance when compared to KNN.

# References

[1] UCI Machine Learning Repository: Data Set", Archive.ics.uci.edu, 2019.[Online].Available: https://archive.ics.uci.edu/ml/datasets/covertypev. [Accessed: 10- Apr- 2019]

[2] T. Raja, "Forest Cover Type Classification Study", Rstudio-pubs-static.s3.amazonaws.com, 2019. [Online]. Available: https://rstudio-pubs-static.s3.amazonaws.com/160297\_f7bcb8d140b74bd19b758eb328344908.html. [Accessed: 10- Apr- 2019]

[3] Cs.waikato.ac.nz, 2019. [Online]. Available: https://www.cs.waikato.ac.nz/~mhall/thesis.pdf. [Accessed: 10- Apr- 2019]

[4] S. Shekhar, M. Grogan and R. Noise, "Why is skewed data not preferred for modelling?", Cross Validated, 2019. [Online]. Available: https://stats.stackexchange.com/questions/267078/why-is-skewed-data-not-preferred-for-modelling. [Accessed: 12- Apr- 2019]

[5] In supervised learning, A. Tavory, V. Calomme and D. Jarratt, "In supervised learning, why is it bad to have correlated features?", Data Science Stack Exchange, 2019. [Online]. Available: https://datascience.stackexchange.com/questions/24452/in-supervised-learning-why-is-it-bad-to-have-correlated-features. [Accessed: 12- Apr- 2019]

[6] "Why Feature Correlation Matters …. A Lot!", Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/why-feature-correlation-matters-a-lot-847e8ba439c4. [Accessed: 12- Apr- 2019]

[7] "Introduction to Data Preprocessing in Machine Learning", Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d. [Accessed: 13- Apr- 2019]

[8] "What is One Hot Encoding? Why and When do you have to use it?", Hacker Noon, 2019. [Online]. Available: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f. [Accessed: 13- Apr- 2019]

[9] "Why Data Normalization is necessary for Machine Learning models", Medium, 2019. [Online]. Available: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029. [Accessed: 13- Apr- 2019]

[10] "SMOTE explained for noobs - Synthetic Minority Over-sampling TEchnique line by line · Rich Data", Rikunert.com, 2019. [Online]. Available: http://rikunert.com/SMOTE\_explained. [Accessed: 14- Apr- 2019]

[11] "4. Combination of over- and under-sampling — imbalanced-learn 0.4.3 documentation", Imbalanced-learn.readthedocs.io, 2019. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/combine.html. [Accessed: 14- Apr- 2019]

[12] "What is over sampling and under sampling? - Definition from WhatIs.com", WhatIs.com, 2019. [Online]. Available: https://whatis.techtarget.com/definition/over-sampling-and-under-sampling. [Accessed: 14- Apr- 2019]

[13] "A Practical Guide to Interpreting and Visualising Support Vector Machines", Towards Data Science, 2019. [Online]. Available:https://towardsdatascience.com/a-practical-guide-to-interpreting-and-visualising-support-vector-machines-97d2a5b0564e. [Accessed: 14- Apr- 2019]

[14] "Pyhon Programming Tutorials", Pythonprogramming.net, 2019. [Online] Available: https://pythonprogramming.net/linear-svc-example-sikit-learn-svm-python/. [Accessed: 14- Apr- 2019]

[15] J. Brownlee, "Logistic Regression for Machine Learning", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/logistic-regression-for-machine-learning/. [Accessed: 15- Apr- 2019]

[16] "Gaussian Naive Bayes Classifier implementation in Python", Dataaspirant, 2019. [Online]. Available: http://dataaspirant.com/2017/02/20/gaussian-naive-bayes-classifier-implementation-python/. [Accessed: 15- Apr- 2019]

[17] "8 Machine Learning Algorithms explained in Human language – Datakeen", Datakeen.co, 2019. [Online]. Available: https://www.datakeen.co/en/8-machine-learning-algorithms-explained-in-human-language/. [Accessed: 15- Apr- 2019]

[18] "The Random Forest Algorithm", Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd. [Accessed: 15- Apr- 2019]

[19] "Distance Based Methods", Statweb.stanford.edu, 2019. [Online]. Available: http://statweb.stanford.edu/~susan/phylo/index/node39.html. [Accessed: 15- Apr- 2019]

[20] "Jupyter Notebook Viewer", Nbviewer.jupyter.org, 2019. [Online]. Available: https://nbviewer.jupyter.org/github/datascienceguide/datascienceguide.github.io/blob/master/tutorials/Association-Rule-Mining.ipynb. [Accessed: 16- Apr- 2019]

[21] "What is the scikit-learn dummy classifier?", Medium, 2019. [Online]. Available: https://medium.com/@mamonu/what-is-the-scikit-learn-dummy-classifier-95549d9cd44. [Accessed: 16- Apr- 2019]

[22] "Use Voting Classifiers — dask-ml 0.12.1 documentation", Ml.dask.org, 2019. [Online]. Available: https://ml.dask.org/examples/voting-classifier.html. [Accessed: 18- Apr- 2019]

[23] "Boosting, Bagging, and Stacking — Ensemble Methods with sklearn and mlens", Medium, 2019. [Online]. Available: https://medium.com/@rrfd/boosting-bagging-and-stacking-ensemble-methods-with-sklearn-and-mlens-a455c0c982de. [Accessed: 18- Apr- 2019]