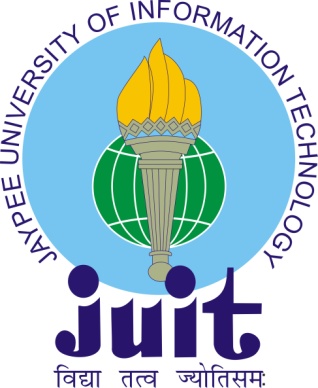
MACHINE **LEARNING** & ITS APPLICATIONS

**A REPORT**

***Submitted by***

**Harshit Gupta (161B094)**

**Under the guidance of: Mr. Piyush Jain**



**June – July 2019**

***Submitted in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Department of Computer Science & Engineering**

**JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY**

**AB ROAD, RAGHOGARH, DT. GUNA-473226 MP, INDIA**

**Declaration by the Student**

I hereby declare that the work reported in the summer industrial training report entitled as

“**MACHINE LEARNING & ITS APPLICATIONS**”, in partial fulfillment for the award of degree of BACHELOR OF TECHNOLOGY submitted at Jaypee University of

Engineering and Technology, Guna, as per best of my knowledge and belief there is no

infringement of intellectual property right and copyright. In case of any violation I will solely be responsible.

HARSHIT GUPTA (161B094)

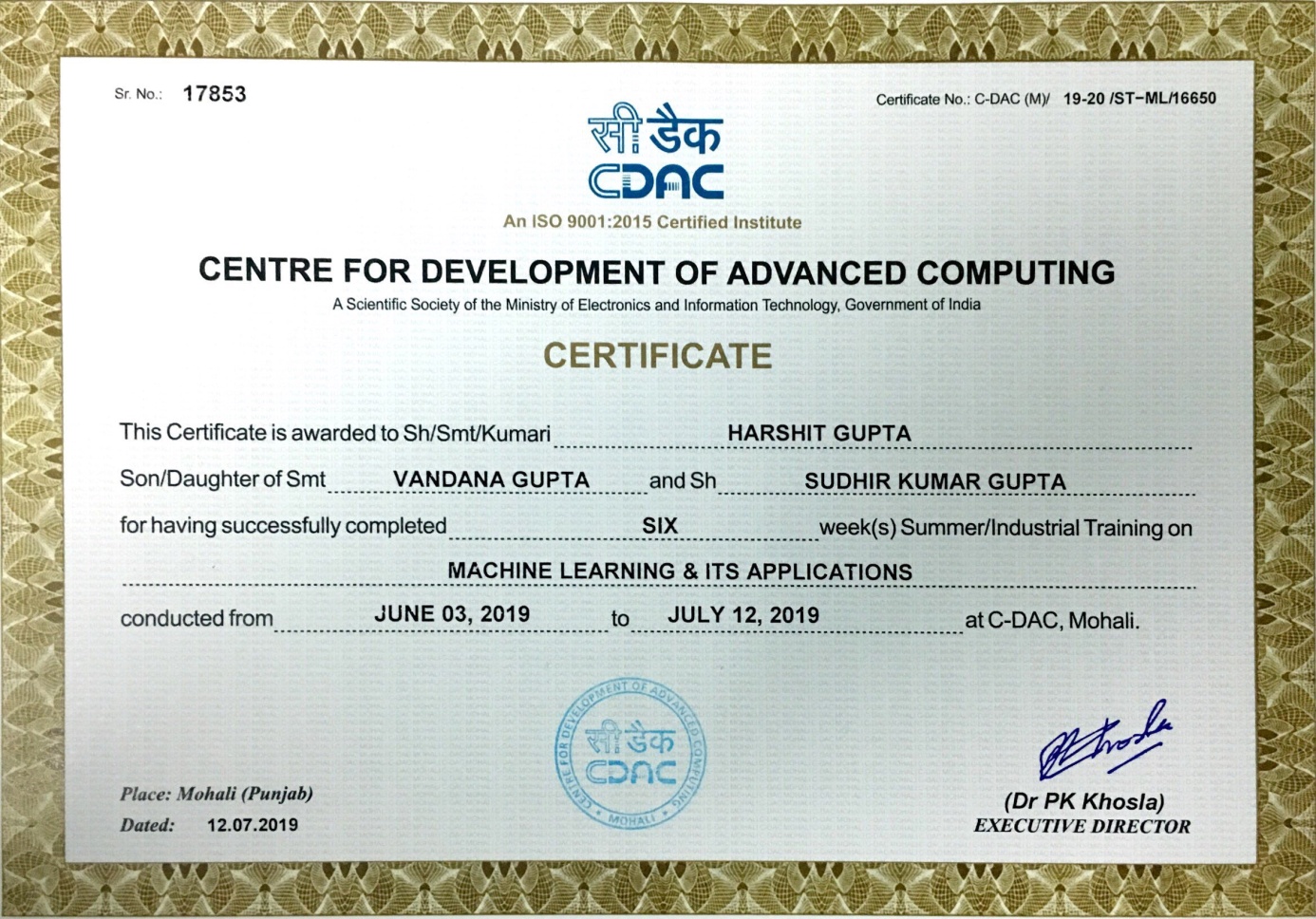
Department of Computer Science and Engineering

Jaypee University of Engineering and Technology

Guna, M.P., India

Date: 12/08/2019

**CERTIFICATE FROM THE SUPERVISOR**

****

**ACKNOWLEDGEMENT**

The training opportunity I had with Centre for Development of Advanced Computing(C-DAC), Mohali was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this training period.

Bearing in mind previous I am using this opportunity to express my deepest gratitude and special thanks to Mr. Piyush Jain who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out my project at their esteemed organization and extending during the training.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

Sincerely,

HARSHIT GUPTA (161B094)

Place: GUNA

Date: 12/08/2019

**EXECUTIVE SUMMARY**

The training opportunity I had with Centre for Development of Advanced Computing(C-DAC), Mohali was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this training period.

I was taught the following during my training:

* **Basics of Machine Learning** – Introduction, Supervised Learning, Unsupervised

Learning, Classification, Regression and Clustering

* **Linear Algebra** – Matrices and Vectors, Addition and Scalar Multiplication, Matrix Vector Multiplication, Matrix Multiplication, Multiplication Properties, Inverse and Transpose
* **Introduction to Python Programming** – Overview, Features, Installation, Python Basics: Data types, Strings, Operators & Expressions etc., Control Flow Instructions, Data Structures: Lists, Dictionaries, Tuples etc., Functions & Modules
* **Python Continue:** Python Standard Library & External Packages, Object Oriented concepts: Classes, Methods, Inheritance, Overriding etc., File Handling, Exception Handling
* **Regression:** Linear Regression Introduction, Model Representation, Cost Function, Gradient Descent for Linear Regression, Vectorized Implementation, Polynomial Regression Introduction, Multiple Features, Gradient Descent For Multiple Variables, Feature Scaling, Learning Rate, A case study in regression
* **Classification:** Logistic Regression Introduction, Model Representation, Decision Boundaries, Cost Function, Gradient Descent, Optimization Objectives, Training and evaluating a classifier, False positives, false negatives, and confusion matrices, Practical on Classification with logistic regression
* **Performance measures in Machine Learning:** Evaluating a Hypothesis, Model Selection and Train/Validation/Test Sets, Splitting the data into training and test sets, cross validation, Diagnosing Bias vs. Variance, Learning Curves, Error Analysis, Lab practical on linear regression
* **Classification - SVM:** Introduction, Optimization Objective, Large Margin, Kernels, applications of SVM
* **Clustering:** Introduction to Unsupervised Learning, Basic issues in clustering, types of clustering, K-Means clustering algorithm, Optimization Objective, Optimizing k value, Lab Practical on Clustering

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure Number** | **Figure Name** | **Page Number** |
| Figure 1 | Employee Turnover | Page 8 |
| Figure 2 | SVM Example | Page 22 |
| Figure 3 | SVM Hyperplane | Page 23 |
| Figure 4 | Confusion Matrix | Page 24 |
| Figure 5 | Specificity Vs. Sensitivity | Page 24 |
| Figure 6 | ROC Curve | Page 25 |
| Figure 7 | Class Count | Page 30 |
| Figure 8 | PCA | Page 32 |
| Figure 9 | Random Forest ROC Curve | Page 36 |
| Figure 10 | Gradient Boosting ROC Curve | Page 38 |
| Figure 11 | KNN ROC Curve | Page 39 |
| Figure 12 | Logistic Regression ROC Curve | Page 41 |
| Figure 13 | SVM ROC Curve | Page 42 |
| Figure 14 | Comparing all ROC Curves | Page 43 |
| Figure 15 | Feature importance graph | Page 44 |

**TABLE OF CONTENTS**

|  |
| --- |
| 1. **Introduction** Page 8 |
| 1. **Review/Background Material** Page 9 |
| * 1. What is Employee Turnover? Page 9 |
| * 1. Types of Employee Turnover Page 9 |
| 1. **Previous Work** Page 11 |
| 1. **Contributional Work** Page 12 |
| 1. **Basic Concepts** Page 13 |
| * 1. Data Preprocessing Page 13 |
| * 1. Data Transformation Page 14 |
| * 1. Data Reduction Page 14 |
| 1. **Managing Imbalanced Data** Page 15 |
| * 1. Undersampling Page 17 |
| * 1. Oversampling Page 17 |
| * 1. Synthetic Data Generation Page 18 |
| 1. **Types of Machine Learning Algorithms** Page 19 |
| * 1. Supervised Algorithms Page 19 |
| * 1. Unsupervised Algorithms Page 19 |
| * 1. Reinforcement Algorithms Page 19 |
| * 1. Some common Machine Learning Algorithms Page 19 |
| 1. **Algorithms Used** Page 20 |
| * 1. Random Forest Page 20 |
| * 1. Gradient Boosting Page 20 |
| * 1. K-Nearest Neighbours Page 20 |
| * 1. Logistic Regression Page 21 |
| * 1. Support Vector Machine(SVM) Page 22 |
| 1. **Evaluating Classification Models** Page 24 |
| * 1. Area under the AUC-ROC Curves Page 24 |
| * 1. Advantages of using the ROC curves Page 26 |

**Introduction**



Figure : Employee Turnover

**Employee turnover** refers to the percentage of workers who leave an organization and are replaced by new employees. It is very costly for organizations, where costs include but not limited to: separation, vacancy, recruitment, training and replacement. On average, organizations invest between four weeks and three months training new employees. This investment would be a loss for the company if the new employee decided to leave the first year. Furthermore, organizations such as consulting firms would suffer from deterioration in customer satisfaction due to regular changes in Account Repsand/or Consultants that would lead to loss of businesses with clients.

**Review/Background Material**

**What is Employee Turnover?**

In [human resources](https://en.wikipedia.org/wiki/Human_resources) context, **turnover** is the act of replacing an [employee](https://en.wikipedia.org/wiki/Employee) with a new employee. Partings between organizations and employees may consist of [termination](https://en.wikipedia.org/wiki/Termination_of_employment), [retirement](https://en.wikipedia.org/wiki/Retirement), death, interagency transfers, and [resignations](https://en.wikipedia.org/wiki/Resignation). An organization’s turnover is measured as a percentage rate, which is referred to as its turnover rate. Turnover rate is the percentage of employees in a workforce that leave during a certain period of time. Organizations and industries as a whole measure their turnover rate during a fiscal or calendar year.

If an [employer](https://en.wikipedia.org/wiki/Employer) is said to have a high turnover rate relative to its [competitors](https://en.wikipedia.org/wiki/Competitor), it means that employees of that company have a shorter average tenure than those of other companies in the same [industry](https://en.wikipedia.org/wiki/Industry). High turnover may be harmful to a company's productivity if skilled workers are often leaving and the [worker](https://en.wikipedia.org/wiki/Worker) population contains a high percentage of novices. Companies will often track turnover internally across departments, divisions, or other [demographic groups](https://en.wikipedia.org/wiki/Demographic_groups), such as turnover of women versus men. Most companies allow [managers](https://en.wikipedia.org/wiki/Manager) to terminate employees at any time, for any reason, or for no reason at all, even if the employee is in good standing. Additionally, companies track voluntary turnover more accurately by presenting parting employees with surveys, thus identifying specific reasons as to why they may be choosing to resign. Many organizations have discovered that turnover is reduced significantly when issues affecting employees are addressed immediately and professionally. Companies try to reduce employee turnover rates by offering benefits such as [paid sick days](https://en.wikipedia.org/wiki/Paid_sick_days), [paid holidays](https://en.wikipedia.org/wiki/Paid_holidays) and [flexible schedules](https://en.wikipedia.org/wiki/Flextime). In the United States, the average total of non-farm seasonally adjusted monthly turnover was 3.3% for the period from December, 2000 to November, 2008. However, rates vary widely when compared over different periods of time and with different job sectors. For example, during the 2001-2006 period, the annual turnover rate for all industry sectors averaged 39.6% prior to seasonal adjustments, while the Leisure and Hospitality sector experienced an average annual rate of 74.6% during this same period.

**Types of Employee Turnover**

There are four types of turnovers: **Voluntary** is the first type of turnover, which occurs when an employee voluntarily chooses to resign from the organization. Voluntary turnover could be the result of a more appealing job offer, staff conflict, or lack of advancement opportunities.[[1]](https://en.wikipedia.org/wiki/Turnover_(employment)#cite_note-ok.gov-1)

The second type of turnover is **involuntary**, which occurs when the employer makes the decision to discharge an employee and the employee unwillingly leaves his or her position. Involuntary turnover could be a result of poor performance, staff conflict, the at-will employment clause, etc.

The third type of turnover is **functional**, which occurs when a low-performing employee leaves the organization. Functional turnover reduces the amount of paperwork that a company must file in order to rid itself of a low-performing employee. Rather than having to go through the potentially difficult process of proving that an employee is inadequate, the company simply respects his or her own decision to leave.

The fourth type of turnover is **dysfunctional**, which occurs when a high-performing employee leaves the organization. Dysfunctional turnover can be potentially costly to an organization, and could be the result of a more appealing job offer or lack of opportunities in career advancement. Too much turnover is not only costly, but it can also give an organization a bad reputation. However, there is also good turnover, which occurs when an organization finds a better fit with a new employee in a certain position. Good turnover can also transpire when an employee has outgrown opportunities within a certain organization and must move forward with his or her career in a new organization.

Turnover as with the non resolution of an employee asset issue (E.g. Cyber : Opportunity & risk) and their course of work in the digital era.

**Previous Work**

Over the years there have been thousands of research articles exploring the various aspects of turnover, and in due course several models of employee turnover have been promulgated. The first model, and by far the one attaining most attention from researchers, was put forward in 1958 by March & Simon. After this model there have been several efforts to extend the concept. Since 1958 the following models of employee turnover have been published.

* March and Simon (1958) Process Model of Turnover
* Porter & Steers (1973) Met Expectations Model
* Price (1977) Causal Model of Turnover
* Mobley (1977) Intermediate Linkages Model
* Hom and Griffeth (1991) Alternative Linkages Model of Turnover
* Whitmore (1979) Inverse Gaussian Model for Labour Turnover
* Steers and Mowday (1981) Turnover Model
* Sheridan & Abelson (1983) Cusp Catastrophe Model of Employee Turnover
* Jackofsky (1984) Integrated Process Model
* Lee et al. (1991) Unfolding Model of Voluntary Employee Turnover[[25]](https://en.wikipedia.org/wiki/Turnover_(employment)#cite_note-25)
* Aquino et al. (1997) Referent Cognitions Model
* Mitchell & Lee (2001) Job Embeddedness Model

**Contributional Work**

**Employee turnover** refers to the percentage of workers who leave an organization and are replaced by new employees. It is very costly for organizations, where costs include but not limited to: separation, vacancy, recruitment, training and replacement. On average, organizations invest between four weeks and three months training new employees. This investment would be a loss for the company if the new employee decided to leave the first year. Furthermore, organizations such as consulting firms would suffer from deterioration in customer satisfaction due to regular changes in *Account Reps*and/or *Consultants* that would lead to loss of businesses with clients.

In this project, we’ll work on simulated HR data from [kaggle](https://www.kaggle.com/ludobenistant/hr-analytics-1" \t "_blank) to build a classifier that helps us predict what kind of employees will be more likely to leave given some attributes. Such classifier would help an organization predict employee turnover and be pro-active in helping to solve such costly matter.

We’ll restrict ourselves to use the most common classifiers: Random Forest, Gradient Boosting Trees, K-Nearest Neighbors, Logistic Regression and Support Vector Machine.

The data has 14,999 examples (samples). Below are the features and the definitions of each one:

* satisfaction\_level: Level of satisfaction {0–1}.
* last\_evaluationTime: Time since last performance evaluation (in years).
* number\_project: Number of projects completed while at work.
* average\_montly\_hours: Average monthly hours at workplace.
* time\_spend\_company: Number of years spent in the company.
* Work\_accident: Whether the employee had a workplace accident.
* left: Whether the employee left the workplace or not {0, 1}.
* promotion\_last\_5years: Whether the employee was promoted in the last five years.
* sales: Department the employee works for.
* salary: Relative level of salary {low, medium, high}.

**Basic Concepts**

**Data Preprocessing**

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

**Steps Involved in Data Preprocessing:**

**1. Data Cleaning:**  
The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

* **(a). Missing Data:**  
  This situation arises when some data is missing in the data. It can be handled in various ways.  
  Some of them are:
  1. **Ignore the tuples:**  
     This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.
  2. **Fill the Missing values:**  
     There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.
* **(b). Noisy Data:**  
  Noisy data is a meaningless data that can’t be interpreted by machines.It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways :
  1. **Binning Method:**  
     This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values can be used to complete the task.
  2. **Regression:**  
     Here data can be made smooth by fitting it to a regression function.The regression used may be linear (having one independent variable) or multiple (having multiple independent variables).
  3. **Clustering:**  
     This approach groups the similar data in a cluster. The outliers may be undetected or it will fall outside the clusters.

**2. Data Transformation:**  
This step is taken in order to transform the data in appropriate forms suitable for mining process. This involves following ways:

1. **Normalization:**  
   It is done in order to scale the data values in a specified range (-1.0 to 1.0 or 0.0 to 1.0)
2. **Attribute Selection:**  
   In this strategy, new attributes are constructed from the given set of attributes to help the mining process.
3. **Discretization:**  
   This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.
4. **Concept Hierarchy Generation:**  
   Here attributes are converted from level to higher level in hierarchy. For Example-The attribute “city” can be converted to “country”.

**3. Data Reduction:**  
Since data mining is a technique that is used to handle huge amount of data. While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we uses data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

The various steps to data reduction are:

1. **Data Cube Aggregation:**  
   Aggregation operation is applied to data for the construction of the data cube.
2. **Attribute Subset Selection:**  
   The highly relevant attributes should be used, rest all can be discarded. For performing attribute selection, one can use level of significance and p- value of the attribute.the attribute having p-value greater than significance level can be discarded.
3. **Dimensionality Reduction:**  
   This reduce the size of data by encoding mechanisms.It can be lossy or lossless. If after reconstruction from compressed data, original data can be retrieved, such reduction are called lossless reduction else it is called lossy reduction. The two effective methods of dimensionality reduction are:Wavelet transforms and PCA (Principal Componenet Analysis).

**Managing Imbalanced Data**

We have several [machine learning algorithms](https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/) at our disposal for model building. Doing data based prediction is now easier like never before. Whether it is a regression or classification problem, one can effortlessly achieve a reasonably high accuracy using a suitable algorithm. But, this is not the case everytime. Classification problems can sometimes get a bit tricky.

ML algorithms tend to tremble when faced with imbalanced classification data sets. Moreover, they result in biased predictions and misleading accuracies. But, why does it happen ? What factors deteriorate their performance ?

The answer is simple. With imbalanced data sets, an algorithm doesn’t get the necessary information about the minority class to make an accurate prediction. Hence, it is desirable to use ML algorithms with balanced data sets.

Imbalanced classification is a supervised learning problem where one class outnumbers other class by a large proportion. This problem is faced more frequently in binary classification problems than multi-level classification problems.

The term imbalanced refer to the disparity encountered in the dependent (response) variable. Therefore, an imbalanced classification problem is one in which the dependent variable has imbalanced proportion of classes. In other words, a data set that exhibits an unequal distribution between its classes is considered to be imbalanced.

For example: Consider a data set with 100,000 observations. This data set consist of candidates who applied for Internship in Harvard. Apparently, harvard is well-known for its extremely low acceptance rate. The dependent variable represents if a candidate has been shortlisted (1) or not shortlisted (0). After analyzing the data, it was found ~ 98% did not get shortlisted and only ~ 2% got lucky. This is a perfect case of imbalanced classification.

In real life, does such situations arise more ? Yes! For better understanding, here are some real life examples. Please note that the degree of imbalance varies per situations:

1. An automated inspection machine which detect products coming off manufacturing assembly line may find number of defective products significantly lower than non defective products.
2. A test done to detect cancer in residents of a chosen area may find the number of cancer affected people significantly less than unaffected people.
3. In credit card fraud detection, fraudulent transactions will be much lower than legitimate transactions.
4. A manufacturing operating under six sigma principle may encounter 10 in a million defected products.

Below are the reasons which leads to reduction in accuracy of ML algorithms on imbalanced data sets:

1. ML algorithms struggle with accuracy because of the unequal distribution in dependent variable.
2. This causes the performance of existing classifiers to get biased towards majority class.
3. The algorithms are accuracy driven i.e. they aim to minimize the overall error to which the minority class contributes very little.
4. ML algorithms assume that the data set has balanced class distributions.
5. They also assume that errors obtained from different classes have same cost.

The methods are widely known as ‘Sampling Methods’. Generally, these methods aim to modify an imbalanced data into balanced distribution using some mechanism. The modification occurs by altering the size of original data set and provide the same proportion of balance.

These methods have acquired higher importance after many researches have proved that balanced data results in improved overall classification performance compared to an imbalanced data set. Hence, it’s important to learn them.

Below are the methods used to treat imbalanced datasets:

1. Undersampling
2. Oversampling
3. Synthetic Data Generation
4. Cost Sensitive Learning

**Undersampling**

This method works with majority class. It reduces the number of observations from majority class to make the data set balanced. This method is best to use when the data set is huge and reducing the number of training samples helps to improve run time and storage troubles.

Undersampling methods are of 2 types: Random and Informative.

Random undersampling method randomly chooses observations from majority class which are eliminated until the data set gets balanced. Informative undersampling follows a pre-specified selection criterion to remove the observations from majority class.

Within informative undersampling, EasyEnsemble and BalanceCascade algorithms are known to produce good results. These algorithms are easy to understand and straightforward too.

EasyEnsemble: At first, it extracts several subsets of independent sample (with replacement) from majority class. Then, it develops multiple classifiers based on combination of each subset with minority class. As you see, it works just like a unsupervised learning algorithm.

BalanceCascade: It takes a supervised learning approach where it develops an ensemble of classifier and systematically selects which majority class to ensemble.

**Oversampling**

This method works with minority class. It replicates the observations from minority class to balance the data. It is also known as upsampling. Similar to undersampling, this method also can be divided into two types: Random Oversampling and Informative Oversampling.

Random oversampling balances the data by randomly oversampling the minority class. Informative oversampling uses a pre-specified criterion and synthetically generates minority class observations.

An advantage of using this method is that it leads to no information loss. The disadvantage of using this method is that, since oversampling simply adds replicated observations in original data set, it ends up adding multiple observations of several types, thus leading to overfitting. Although, the training accuracy of such data set will be high, but the accuracy on unseen data will be worse.

**Synthetic Data Generation**

In simple words, instead of replicating and adding the observations from the minority class, it overcome imbalances by generates artificial data. It is also a type of oversampling technique.

In regards to synthetic data generation, synthetic minority oversampling technique (SMOTE) is a powerful and widely used method. SMOTE algorithm creates artificial data based on feature space (rather than data space) similarities from minority samples. We can also say, it generates a random set of minority class observations to shift the classifier learning bias towards minority class.

To generate artificial data, it uses bootstrapping and k-nearest neighbors. Precisely, it works this way:

1. Take the difference between the feature vector (sample) under consideration and its nearest neighbor.
2. Multiply this difference by a random number between 0 and 1
3. Add it to the feature vector under consideration
4. This causes the selection of a random point along the line segment between two specific features

**Types of Machine Learning Algorithms**

**Supervised Learning**

This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, [Decision Tree](https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/), [Random Forest](https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/), KNN, Logistic Regression etc.

**Unsupervised Learning**

In this algorithm, we do not have any target or outcome variable to predict / estimate.  It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.

**Reinforcement Learning**

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process

**Some common machine learning algorithms**

Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem:

1. Linear Regression
2. Logistic Regression
3. Decision Tree
4. SVM
5. Naive Bayes
6. kNN
7. K-Means
8. Random Forest
9. Dimensionality Reduction Algorithms
10. Gradient Boosting algorithms

**Algorithms Used**

**Random Forest**

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we’ve collection of decision trees (so known as “Forest”). To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

**Gradient Boosting**

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to a build strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix.

**K Nearest Neighbours**

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

**Things to consider before selecting kNN:**

* KNN is computationally expensive
* Variables should be normalized else higher range variables can bias it
* Works on pre-processing stage more before going for kNN like outlier, noise removal

**Logistic Regression**

Don’t get confused by its name! It is a classification not a regression algorithm. It is used to estimate discrete values ( Binary values like 0/1, yes/no, true/false ) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a [logit function](https://en.wikipedia.org/wiki/Logistic_function" \t "_blank). Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

Again, let us try and understand this through a simple example.

Let’s say your friend gives you a puzzle to solve. There are only 2 outcome scenarios – either you solve it or you don’t. Now imagine, that you are being given wide range of puzzles / quizzes in an attempt to understand which subjects you are good at. The outcome to this study would be something like this – if you are given a trignometry based tenth grade problem, you are 70% likely to solve it. On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you.

Coming to the math, the log odds of the outcome is modeled as a linear combination of the predictor variables.

odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence

ln(odds) = ln(p/(1-p))

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk

Above, p is the probability of presence of the characteristic of interest. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression).

Now, you may ask, why take a log? For the sake of simplicity, let’s just say that this is one of the best mathematical way to replicate a step function. I can go in more details, but that will beat the purpose of this article.

**Support Vector Machine**

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we’d first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**).

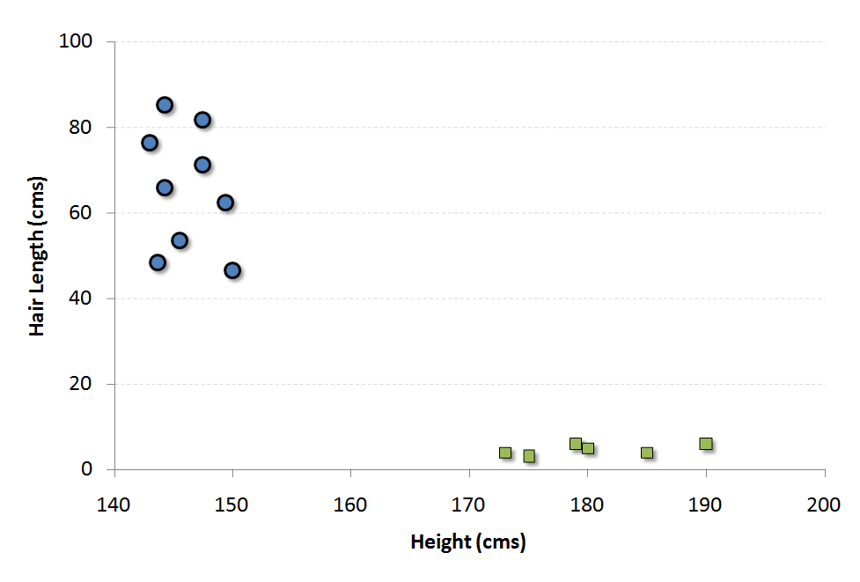


Figure : SVM Example

Now, we will find some line that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.

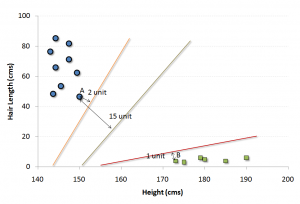


Figure : SVM Hyperplane

In the example shown above, the line which splits the data into two differently classified groups is the black line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that’s what class we can classify the new data as.

**Evaluating Classification Models**

**Area under the AUC-ROC Curve**

This is one of the popular metrics used in the industry.  The biggest advantage of using ROC curve is that it is independent of the change in proportion of responders. This statement will get clearer in the following sections.

Let’s first try to understand what is ROC (Receiver operating characteristic) curve. If we look at the confusion matrix below, we observe that for a probabilistic model, we get different value for each metric.

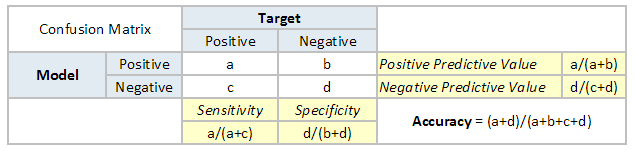


Figure : Confusion Matrix

Hence, for each sensitivity, we get a different specificity.The two vary as follows:

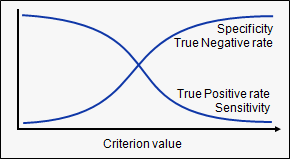


Figure : Specificity Vs. Sensitivity

The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate. Following is the ROC curve for the case in hand.

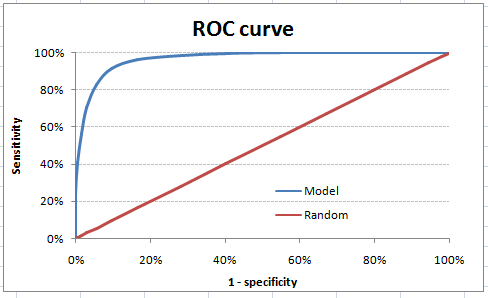
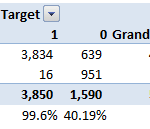


Figure : Roc Curve

Let’s take an example of threshold = 0.5 (refer to confusion matrix). Here is the confusion matrix :



As you can see, the sensitivity at this threshold is 99.6% and the (1-specificity) is ~60%. This coordinate becomes on point in our ROC curve. To bring this curve down to a single number, we find the area under this curve (AUC).

Note that the area of entire square is 1\*1 = 1. Hence AUC itself is the ratio under the curve and the total area. For the case in hand, we get AUC ROC as 96.4%. Following are a few thumb rules:

* .90-1 = excellent (A)
* .80-.90 = good (B)
* .70-.80 = fair (C)
* .60-.70 = poor (D)
* .50-.60 = fail (F)

We see that we fall under the excellent band for the current model. But this might simply be over-fitting. In such cases it becomes very important to to in-time and out-of-time validations.

**Points to Remember:**

1. For a model which gives class as output, will be represented as a single point in ROC plot.

2. Such models cannot be compared with each other as the judgement needs to be taken on a single metric and not using multiple metrics. For instance, model with parameters (0.2,0.8) and model with parameter (0.8,0.2) can be coming out of the same model, hence these metrics should not be directly compared.

3. In case of probabilistic model, we were fortunate enough to get a single number which was AUC-ROC. But still, we need to look at the entire curve to make conclusive decisions. It is also possible that one model performs better in some region and other performs better in other.

**Advantages of using ROC**

Why should you use ROC and not metrics like lift curve?

Lift is dependent on total response rate of the population. Hence, if the response rate of the population changes, the same model will give a different lift chart. A solution to this concern can be true lift chart (finding the ratio of lift and perfect model lift at each decile). But such ratio rarely makes sense for the business.

ROC curve on the other hand is almost independent of the response rate. This is because it has the two axis coming out from columnar calculations of confusion matrix. The numerator and denominator of both x and y axis will change on similar scale in case of response rate shift.

**Project Code**

**Importing all libraries**

import os

import matplotlib.pyplot as plt

from matplotlib.gridspec import GridSpec

import numpy as np

import pandas as pd

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.metrics import (accuracy\_score,

f1\_score,

roc\_auc\_score,

roc\_curve,

confusion\_matrix)

from sklearn.model\_selection import (cross\_val\_score,

GridSearchCV,

RandomizedSearchCV,

learning\_curve,

validation\_curve,

train\_test\_split)

from sklearn.pipeline import make\_pipeline

from sklearn.utils import resample

from warnings import filterwarnings

os.chdir("../")

from scripts.plot\_roc import plot\_conf\_matrix\_and\_roc, plot\_roc

%matplotlib inline

sns.set\_context("notebook")

plt.style.use("fivethirtyeight")

filterwarnings("ignore")

**Data Preprocessing**

Let's take a look at the data (check if there are missing values and the data type of each features):

In [2]:

*# Load the data*

df = pd.read\_csv("data/HR\_comma\_sep.csv")

*# Check both the datatypes and if there is missing values*

print("**\033**[1m" + "**\033**[94m" + "Data types:**\n**" + 11 \* "-")

print("**\033**[30m" + "**{}\n**".format(df.dtypes))

print("**\033**[1m" + "**\033**[94m" + "Sum of null values in each column:**\n**" + 35 \* "-")

print("**\033**[30m" + "**{}**".format(df.isnull().sum()))

df.head()

**Output**

Data types:

-----------

satisfaction\_level float64

last\_evaluation float64

number\_project int64

average\_montly\_hours int64

time\_spend\_company int64

Work\_accident int64

left int64

promotion\_last\_5years int64

sales object

salary object

dtype: object

Sum of null values in each column:

-----------------------------------

satisfaction\_level 0

last\_evaluation 0

number\_project 0

average\_montly\_hours 0

time\_spend\_company 0

Work\_accident 0

left 0

promotion\_last\_5years 0

sales 0

salary 0

dtype: int64

Since there are no missing values, we do not have to do any imputation. However, there are some data preprocessing needed:

1. Change **sales** feature name to **department**.
2. Convert **salary** into *ordinal categorical* feature since there is intrinsic order between: low, medium and high.
3. Create dummy features from **department** feature and drop the first one to avoid linear dependency where some learning algorithms may struggle with.

*# Rename sales feature into department*

df = df.rename(columns={"sales": "department"})

*# Map salary into integers*

salary\_map = {"low": 0, "medium": 1, "high": 2}

df["salary"] = df["salary"].map(salary\_map)

*# Create dummy variables for department feature*

df = pd.get\_dummies(df, columns=["department"], drop\_first=**True**)

df.head()

**Modeling**

Let's first take a look at the proportion of each class to see if we're dealing with balanced or imbalanced data since each one has its own set of tools to be used when fitting classifiers.

*# Get number of positve and negative examples*

pos = df[df["left"] == 1].shape[0]

neg = df[df["left"] == 0].shape[0]

print("Positive examples = **{}**".format(pos))

print("Negative examples = **{}**".format(neg))

print("Proportion of positive to negative examples = **{:.2f}**%".format((pos / neg) \* 100))

sns.countplot(df["left"])

plt.xticks((0, 1), ["Didn't leave", "Left"])

plt.xlabel("Left")

plt.ylabel("Count")

plt.title("Class counts");

**Output**

Positive examples = 3571

Negative examples = 11428

Proportion of positive to negative examples = 31.25%

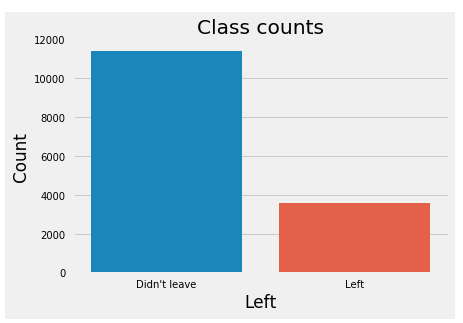


Figure : Class Count

As the graph shows, we have an imbalanced dataset. As a result, when we fit classifiers on such datasets, we should use metrics other than accuracy when comparing models such as f1-score or AUC (area under ROC curve). Moreover, class imbalance influences a learning algorithm during training by making the decision rule biased towards the majority class by implicitly learns a model that optimizes the predictions based on the majority class in the dataset. There are three ways to deal with this issue:

1. Assign a larger penalty to wrong predictions from the minority class.
2. Upsampling the minority class or downsampling the majority class.
3. Generate synthetic training examples.

Nonetheless, there is no definitive guide or best practices to deal with such situations. Therefore, we have to try them all and see which one works better on the problem at hand. We'll restrict ourselves to use the first two, i.e assign larger penalty to wrong predictions from the minority class using class\_weight in classifiers that allows us do that and evaluate upsampling/downsampling on the training data to see which gives higher performance.

First, split the data into training and test sets using 80/20 split; 80% of the data will be used to train the models and 20% to test the performance of the models. Second, Upsample the minority class and downsample the majority class. For this data set, positive class is the minority class and negative class is the majority class.

*# Convert dataframe into numpy objects and split them into*

*# train and test sets: 80/20*

X = df.loc[:, df.columns != "left"].values

y = df.loc[:, df.columns == "left"].values.flatten()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, stratify=y, random\_state=1)

*# Upsample minority class*

X\_train\_u, y\_train\_u = resample(X\_train[y\_train == 1],

y\_train[y\_train == 1],

replace=**True**,

n\_samples=X\_train[y\_train == 0].shape[0],

random\_state=1)

X\_train\_u = np.concatenate((X\_train[y\_train == 0], X\_train\_u))

y\_train\_u = np.concatenate((y\_train[y\_train == 0], y\_train\_u))

*# Downsample majority class*

X\_train\_d, y\_train\_d = resample(X\_train[y\_train == 0],

y\_train[y\_train == 0],

replace=**True**,

n\_samples=X\_train[y\_train == 1].shape[0],

random\_state=1)

X\_train\_d = np.concatenate((X\_train[y\_train == 1], X\_train\_d))

y\_train\_d = np.concatenate((y\_train[y\_train == 1], y\_train\_d))

print("Original shape:", X\_train.shape, y\_train.shape)

print("Upsampled shape:", X\_train\_u.shape, y\_train\_u.shape)

print("Downsampled shape:", X\_train\_d.shape, y\_train\_d.shape)

**Output**

Original shape: (11999, 17) (11999,)

Upsampled shape: (18284, 17) (18284,)

Downsampled shape: (5714, 17) (5714,)

I don't think we need to apply dimensionality reduction such as PCA because: 1) We want to know the importance of each feature in determining who will leave vs who won't (inference). 2) Dimension of the dataset is descent (17 features). However, it's good to see how many principal components needed to explain 90%, 95% and 99% of the variation in the data.

*# Build PCA using standarized trained data*

pca = PCA(n\_components=**None**, svd\_solver="full")

pca.fit(StandardScaler().fit\_transform(X\_train))

cum\_var\_exp = np.cumsum(pca.explained\_variance\_ratio\_)

plt.figure(figsize=(12, 6))

plt.bar(range(1, 18), pca.explained\_variance\_ratio\_, align="center",

color='red', label="Individual explained variance")

plt.step(range(1, 18), cum\_var\_exp, where="mid", label="Cumulative explained variance")

plt.xticks(range(1, 18))

plt.legend(loc="best")

plt.xlabel("Principal component index", {"fontsize": 14})

plt.ylabel("Explained variance ratio", {"fontsize": 14})

plt.title("PCA on training data", {"fontsize": 16});

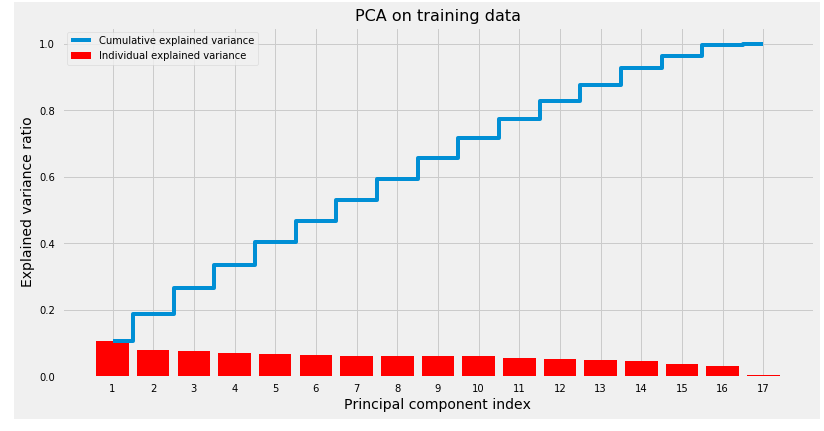
****

Figure : PCA

Looks like it needs 14, 15 and 16 principal components to capture 90%, 95% and 99% of the variation in the data respectively. In other words, this means that the data is already in a good space since eigenvalues are very close to each other and gives further evidence that we don't need to compress the data.

The methodology that we'll follow when building the classifiers goes as follows:

1. Build a pipeline that handles all the steps when fitting the classifier using scikit-learn's make\_pipeline which will have two steps:
   1. Standardizing the data to speed up convergence and make all features on the same scale.
   2. The classifier (estimator) we want to use to fit the model.
2. Use GridSearchCV to tune hyperparameters using 10-folds cross validation. We can use RandomizedSearchCV which is faster and may outperform GridSearchCV especially if we have more than two hyperparameters and the range for each one is very big; however, GridSearchCV will work just fine since we have only two hyperparameters and descent range.
3. Fit the model using training data.
4. Plot both confusion matrix and ROC curve for the best estimator using test data.

**Random Forest**

First, we will start by fitting a Random Forest classifier using unsampled, upsampled and downsampled data. Second, we will evaluate each method using cross validation (CV) f1-score and pick the one with the highest CV f1-score. Finally, we will use that method to fit the rest of the classifiers.

The only hyperparameters we'll tune are:

* max\_feature: how many features to consider randomly on each split. This will help avoid having few strong features to be picked on each split and let other features have the chance to contribute. Therefore, predictions will be less correlated and the variance of each tree will decrease.
* min\_samples\_leaf: how many examples to have for each split to be a final leaf node.

Random Forest is an ensemble model that has multiple trees (n\_estimators), where each tree is a weak learner. The final prediction would be a weighting average or mode of the predictions from all estimators. Note: high number of trees don't cause overfitting.

*# Build random forest classifier*

methods\_data = {"Original": (X\_train, y\_train),

"Upsampled": (X\_train\_u, y\_train\_u),

"Downsampled": (X\_train\_d, y\_train\_d)}

**for** method **in** methods\_data.keys():

pip\_rf = make\_pipeline(StandardScaler(),

RandomForestClassifier(n\_estimators=500,

class\_weight="balanced",

random\_state=123))

hyperparam\_grid = {

"randomforestclassifier\_\_n\_estimators": [10, 50, 100, 500],

"randomforestclassifier\_\_max\_features": ["sqrt", "log2", 0.4, 0.5],

"randomforestclassifier\_\_min\_samples\_leaf": [1, 3, 5],

"randomforestclassifier\_\_criterion": ["gini", "entropy"]}

gs\_rf = GridSearchCV(pip\_rf,

hyperparam\_grid,

scoring="f1",

cv=10,

n\_jobs=-1)

gs\_rf.fit(methods\_data[method][0], methods\_data[method][1])

print("**\033**[1m" + "**\033**[0m" + "The best hyperparameters for **{}** data:".format(method))

**for** hyperparam **in** gs\_rf.best\_params\_.keys():

print(hyperparam[hyperparam.find("\_\_") + 2:], ": ", gs\_rf.best\_params\_[hyperparam])

print("**\033**[1m" + "**\033**[94m" + "Best 10-folds CV f1-score: **{:.2f}**%.".format((gs\_rf.best\_score\_) \* 100))

**Output**

The best hyperparameters for Original data:

criterion : gini

max\_features : 0.5

min\_samples\_leaf : 1

n\_estimators : 500

**Best 10-folds CV f1-score: 98.19%.**

The best hyperparameters for Upsampled data:

criterion : entropy

max\_features : 0.4

min\_samples\_leaf : 1

n\_estimators : 50

**Best 10-folds CV f1-score: 99.80%.**

The best hyperparameters for Downsampled data:

criterion : entropy

max\_features : 0.4

min\_samples\_leaf : 1

n\_estimators : 500

**Best 10-folds CV f1-score: 98.44%.**

Upsampling yielded the highest CV f1-score with 99.80%. Therefore, we'll be using the upsampled data to fit the rest of the classifiers. The new data now has 18,284 examples with 50% of the examples belong to the positive class and the other 50% belong to the negative example.

Let's refit the Random Forest with Upsampled data using best hyperparameters tuned above and plot confusion matrix and ROC curve using test data.

*# Reassign original training data to upsampled data*

X\_train, y\_train = np.copy(X\_train\_u), np.copy(y\_train\_u)

*# Delete original and downsampled data*

**del** X\_train\_u, y\_train\_u, X\_train\_d, y\_train\_d

*# Refit RF classifier using best params*

clf\_rf = make\_pipeline(StandardScaler(),

RandomForestClassifier(n\_estimators=50,

criterion="entropy",

max\_features=0.4,

min\_samples\_leaf=1,

class\_weight="balanced",

n\_jobs=-1,

random\_state=123))

clf\_rf.fit(X\_train, y\_train)

*# Plot confusion matrix and ROC curve*

plot\_conf\_matrix\_and\_roc(clf\_rf, X\_test, y\_test)

**Output**

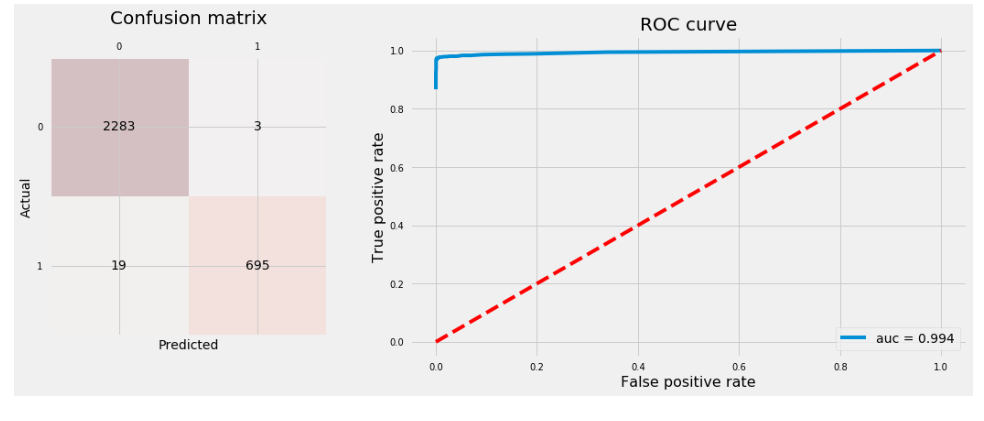
****

Figure : Random Forest ROC Curve

**Gradient Boosting Trees**

Gradient Boosting trees are the same as Random Forest except for:

* It starts with small tree and start learning from grown trees by taking into account the residual of grown trees.
* More trees can lead to overfitting; opposite to Random Forest.

The two other hyperparameters than max\_features and n\_estimators that we're going to tune are:

* learning\_rate: rate the tree learns, the slower the better.
* max\_depth: number of split each time a tree is growing which limits the number of nodes in each tree.

Let's fit GB classifier and plot confusion matrix and ROC curve using test data.

*# Build Gradient Boosting classifier*

pip\_gb = make\_pipeline(StandardScaler(),

GradientBoostingClassifier(loss="deviance",

random\_state=123))

hyperparam\_grid = {"gradientboostingclassifier\_\_max\_features": ["log2", 0.5],

"gradientboostingclassifier\_\_n\_estimators": [100, 300, 500],

"gradientboostingclassifier\_\_learning\_rate": [0.001, 0.01, 0.1],

"gradientboostingclassifier\_\_max\_depth": [1, 2, 3]}

gs\_gb = GridSearchCV(pip\_gb,

param\_grid=hyperparam\_grid,

scoring="f1",

cv=10,

n\_jobs=-1)

gs\_gb.fit(X\_train, y\_train)

print("**\033**[1m" + "**\033**[0m" + "The best hyperparameters:")

print("-" \* 25)

**for** hyperparam **in** gs\_gb.best\_params\_.keys():

print(hyperparam[hyperparam.find("\_\_") + 2:], ": ", gs\_gb.best\_params\_[hyperparam])

print("**\033**[1m" + "**\033**[94m" + "Best 10-folds CV f1-score: **{:.2f}**%.".format((gs\_gb.best\_score\_) \* 100))

*# Plot confusion matrix and ROC curve*

plot\_conf\_matrix\_and\_roc(gs\_gb, X\_test, y\_test)

**Output**

The best hyperparameters:

-------------------------

learning\_rate : 0.1

max\_depth : 3

max\_features : 0.5

n\_estimators : 500

**Best 10-folds CV f1-score: 97.88%.**

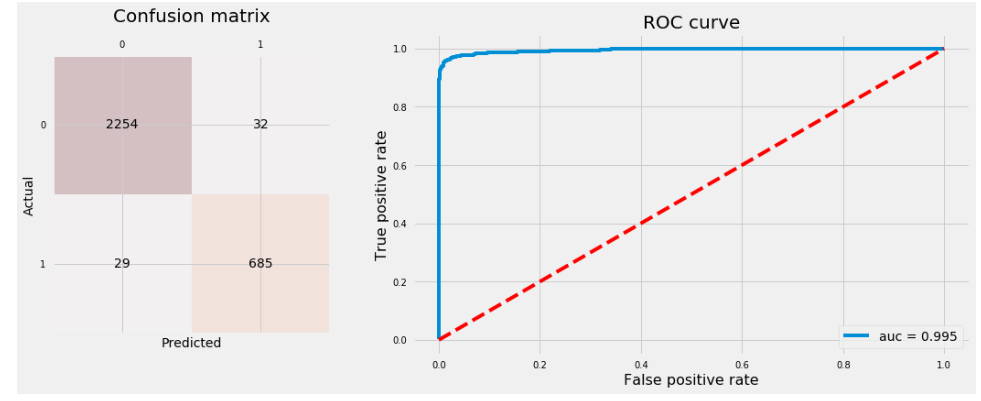
****

Figure : Gradient Boosting ROC Curve

**K Nearest Neighbours**

KNN is called a lazy learning algorithm because it doesn't learn or fit any parameter. It takes n\_neighbors points from the training data closest to the point we're interested to predict it's class and take the mode (majority vote) of the classes for the neighboring point as its class. The two hyperparameters we're going to tune are:

* n\_neighbors: number of neighbors to use in prediction.
* weights: how much weight to assign neighbors based on:
  + "uniform": all neighboring points have the same weight.
  + "distance": use the inverse of euclidean distance of each neighboring point used in prediction.

Let's fit KNN classifier and plot confusion matrix and ROC curve.

*# Build KNN classifier*

pip\_knn = make\_pipeline(StandardScaler(), KNeighborsClassifier())

hyperparam\_range = range(1, 20)

gs\_knn = GridSearchCV(pip\_knn,

param\_grid={"kneighborsclassifier\_\_n\_neighbors": hyperparam\_range,

"kneighborsclassifier\_\_weights": ["uniform", "distance"]},

scoring="f1",

cv=10,

n\_jobs=-1)

gs\_knn.fit(X\_train, y\_train)

print("**\033**[1m" + "**\033**[0m" + "The best hyperparameters:")

print("-" \* 25)

**for** hyperparam **in** gs\_knn.best\_params\_.keys():

print(hyperparam[hyperparam.find("\_\_") + 2:], ": ", gs\_knn.best\_params\_[hyperparam])

print("**\033**[1m" + "**\033**[94m" + "Best 10-folds CV f1-score: **{:.2f}**%.".format((gs\_knn.best\_score\_) \* 100))

plot\_conf\_matrix\_and\_roc(gs\_knn, X\_test, y\_test)

**Output**

The best hyperparameters:

-------------------------

n\_neighbors : 1

weights : uniform

**Best 10-folds CV f1-score: 98.24%.**

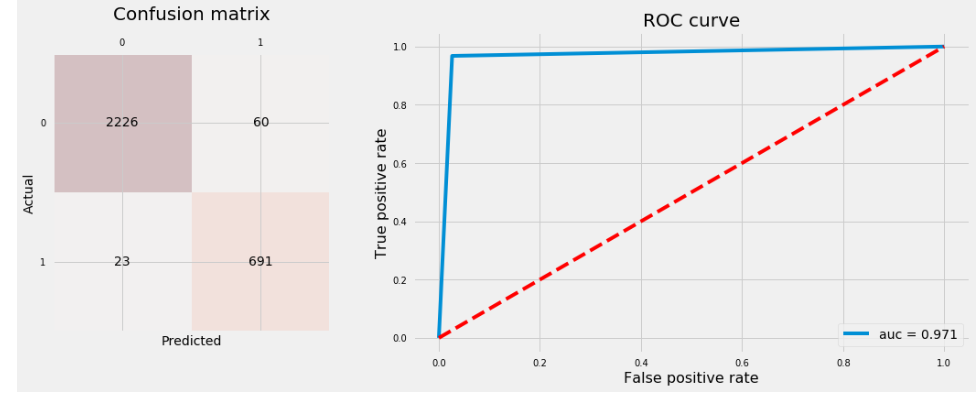
****

Figure : KNN ROC Curve

**Logistic Regression**

*# Build logistic model classifier*

pip\_logmod = make\_pipeline(StandardScaler(),

LogisticRegression(class\_weight="balanced"))

hyperparam\_range = np.arange(0.5, 20.1, 0.5)

hyperparam\_grid = {"logisticregression\_\_penalty": ["l1", "l2"],

"logisticregression\_\_C": hyperparam\_range,

"logisticregression\_\_fit\_intercept": [**True**, **False**]

}

gs\_logmodel = GridSearchCV(pip\_logmod,

hyperparam\_grid,

scoring="accuracy",

cv=2,

n\_jobs=-1)

gs\_logmodel.fit(X\_train, y\_train)

print("**\033**[1m" + "**\033**[0m" + "The best hyperparameters:")

print("-" \* 25)

**for** hyperparam **in** gs\_logmodel.best\_params\_.keys():

print(hyperparam[hyperparam.find("\_\_") + 2:], ": ", gs\_logmodel.best\_params\_[hyperparam])

print("**\033**[1m" + "**\033**[94m" + "Best 10-folds CV f1-score: **{:.2f}**%.".format((gs\_logmodel.best\_score\_) \* 100))

The best hyperparameters:

plot\_conf\_matrix\_and\_roc(gs\_logmodel, X\_test, y\_test)

**Output**

The best hyperparameters:

-------------------------

C : 1.0

fit\_intercept : True

penalty : l1

**Best 10-folds CV f1-score: 77.20%.**

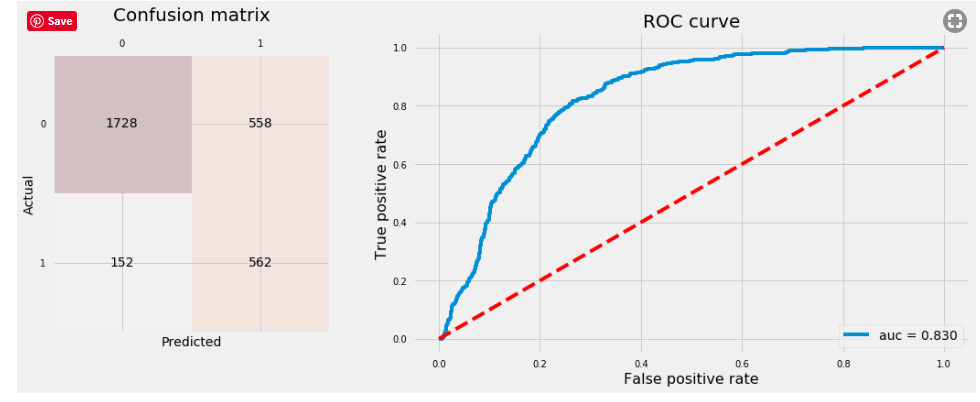
****

Figure : Logistic Regression ROC Curve

**Support Vector Machine (SVM)**

SVM is comutationally very expensive to tune it's hyperparameters for two reasons:

1. With big datasets, it becomes very slow.
2. It has good number of hyperparameters to tune that takes very long time to tune on a CPU.

Therefore, we'll use recommended hyperparameters' values from the paper we mentioned before that showed to yield the best performane on Penn Machine Learning Benchmark 165 datasets. The hyperparameters that we usually look to tune are:

* C, gamma, kernel, degree and coef0

*# Build SVM classifier*

clf\_svc = make\_pipeline(StandardScaler(),

SVC(C=0.01,

gamma=0.1,

kernel="poly",

degree=5,

coef0=10,

probability=**True**))

clf\_svc.fit(X\_train, y\_train)

svc\_cv\_scores = cross\_val\_score(clf\_svc,

X=X\_train,

y=y\_train,

scoring="f1",

cv=10,

n\_jobs=-1)

*# Print CV*

print("**\033**[1m" + "**\033**[94m" + "The 10-folds CV f1-score is: **{:.2f}**%".format(

np.mean(svc\_cv\_scores) \* 100))

plot\_conf\_matrix\_and\_roc(clf\_svc, X\_test, y\_test)

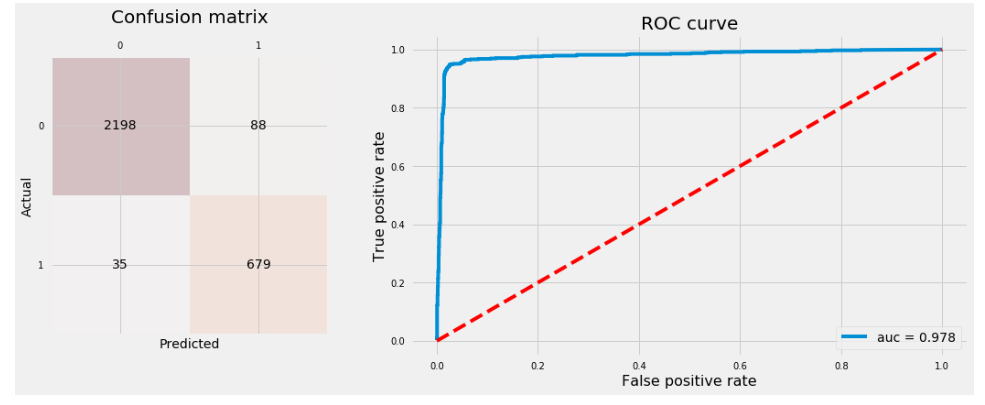
****

Figure : SVM ROC Curve

**Comparing all ROC curves**

*# Plot ROC curves for all classifiers*

estimators = {"RF": clf\_rf,

"LR": gs\_logmodel,

"SVC": clf\_svc,

"GBT": gs\_gb,

"KNN": gs\_knn}

plot\_roc(estimators, X\_test, y\_test, (12, 8))

*# Print out accuracy score on test data*

print("The accuracy rate and f1-score on test data are:")

**for** estimator **in** estimators.keys():

print("**{}**: **{:.2f}**%, **{:.2f}**%.".format(estimator,

accuracy\_score(y\_test, estimators[estimator].predict(X\_test)) \* 100,

f1\_score(y\_test, estimators[estimator].predict(X\_test)) \* 100))

**Output**

The accuracy rate and f1-score on test data are:

RF: 99.27%, 98.44%.

LR: 76.33%, 61.29%.

SVC: 95.90%, 91.69%.

GBT: 97.97%, 95.74%.

KNN: 97.23%, 94.33%.

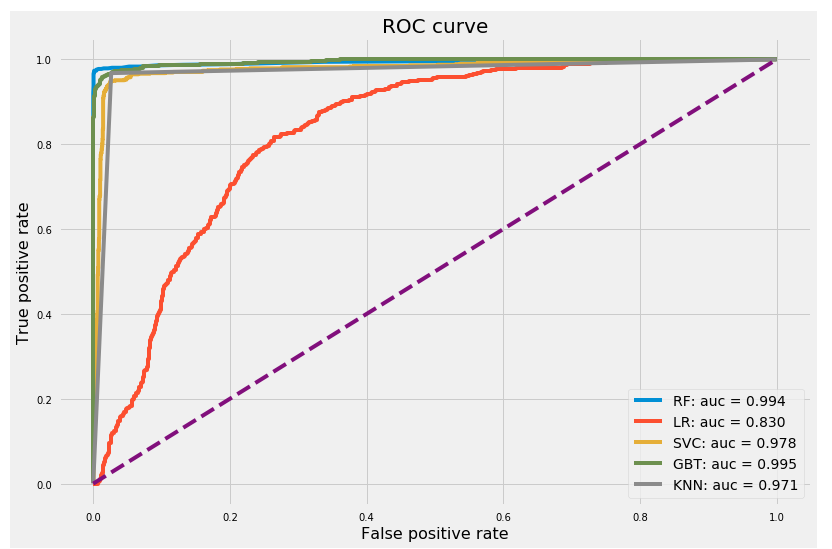
****

Figure : Comparing all ROC curves

Even though Random Forest and Gradient Boosting Trees have almost equal auc, Random Forest has higher accuracy rate and an f1-score with 99.27% and 99.44% respectively. Therefore, we safely say Random Forest outperforms the rest of the classifiers. Let's have a look of feature importances from Random Forest classifier.

*# Refit RF classifier*

clf\_rf = RandomForestClassifier(n\_estimators=50,

criterion="entropy",

max\_features=0.4,

min\_samples\_leaf=1,

class\_weight="balanced",

n\_jobs=-1,

random\_state=123)

clf\_rf.fit(StandardScaler().fit\_transform(X\_train), y\_train)

**Feature Extraction**

*# Plot features importance*

importances = clf\_rf.feature\_importances\_

indices = np.argsort(clf\_rf.feature\_importances\_)[::-1]

plt.figure(figsize=(12, 6))

plt.bar(range(1, 18), importances[indices], align="center")

plt.xticks(range(1, 18), df.columns[df.columns != "left"][indices], rotation=90)

plt.title("Feature Importance", {"fontsize": 16});

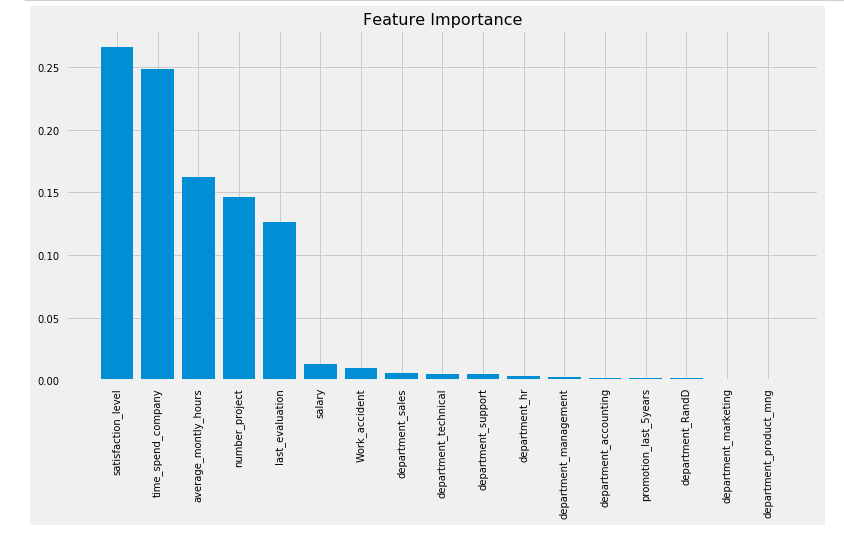
****

Figure : Feature importance graph

Looks like the five most important features are:

* satisfaction\_level
* time\_spend\_company
* average\_montly\_hours
* number\_project
* lats\_evaluation

The following can be noted:

* When dealing with imbalanced classes, accuracy is not a good method for model evaluation. AUC and f1-score are examples of metrics we can use.
* Upsampling/downsampling, data synthetic and using balanced class weights are good strategies to try to improve the accuracy of a classifier for imbalanced classes datasets.
* GridSearchCV helps tune hyperparameters for each learning algorithm. RandomizedSearchCV is faster and may outperform GridSearchCV especially when we have more than two hyperparameters to tune.
* Principal Component Analysis (PCA) isn't always recommended especially if the data is in a good feature space and their eigen values are very close to each other.
* As expected, ensemble models outperforms other learning algorithms in most cases.

**References**

* <https://www.analyticsvidhya.com/>
* <https://www.coursera.org/learn/machine-learning>
* <https://en.wikipedia.org/wiki/Machine_learning>