**A Comprehensive Guide to Data Exploration**

[**SUNIL RAY**](https://www.analyticsvidhya.com/blog/author/sunil-ray/)**, JANUARY 10, 2016**

**Introduction**

There are no shortcuts for data exploration. If you are in a state of mind, that machine learning can sail you away from every data storm, trust me, it won’t. After some point of time, you’ll realize that you are struggling at improving model’s accuracy. In such situation, data exploration techniques will come to your rescue.

I can confidently say this, because I’ve been through such situations, a lot.

I have been a Business Analytics professional for close to three years now. In my initial days, one of my mentor suggested me to spend significant time on exploration and analyzing data. Following his advice has served me well.

I’ve created this tutorial to help you understand the underlying techniques of data exploration. As always, I’ve tried my best to explain these concepts in the simplest manner. For better understanding, I’ve taken up few examples to demonstrate the complicated concepts.

**Table of Contents**

1. [**Steps of Data Exploration and Preparation**](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/#one)
2. [**Missing Value Treatment**](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/#two)
   * Why missing value treatment is required ?
   * Why data has missing values?
   * Which are the methods to treat missing value ?
3. [**Techniques of Outlier Detection and Treatment**](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/#three)
   * What is an outlier?
   * What are the types of outliers ?
   * What are the causes of outliers ?
   * What is the impact of outliers on dataset ?
   * How to detect outlier ?
   * How to remove outlier ?
4. [**The Art of Feature Engineering**](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/#four)
   * What is Feature Engineering ?
   * What is the process of Feature Engineering ?
   * What is Variable Transformation ?
   * When should we use variable transformation ?
   * What are the common methods of variable transformation ?
   * What is feature variable creation and its benefits ?

**Let’s get started.**

**1. Steps of Data Exploration and Preparation**

Remember the quality of your inputs decide the quality of your output. So, once you have got your business hypothesis ready, it makes sense to spend lot of time and efforts here. With my personal estimate, data exploration, cleaning and preparation can take up to 70% of your total project time.

Below are the steps involved to understand, clean and prepare your data for building your predictive model:

1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Missing values treatment
5. Outlier treatment
6. Variable transformation
7. Variable creation

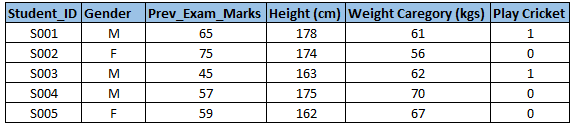
Finally, we will need to iterate over steps 4 – 7 multiple times before we come up with our refined model.

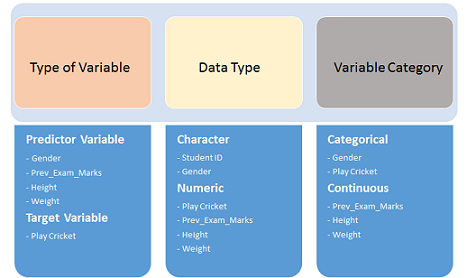
Let’s now study each stage in  detail:-

**Variable Identification**

First, identify **Predictor** (Input) and **Target** (output) variables. Next, identify the data type and category of the variables.

Let’s understand this step more clearly by taking an example.

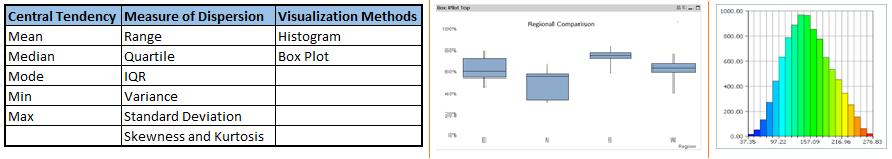
Example:- Suppose, we want to predict, whether the students will play cricket or not (refer below data set). Here you need to identify predictor variables, target variable, data type of variables and category of variables.[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_11.png)Below, the variables have been defined in different category:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_2.png)

**Univariate Analysis**

At this stage, we explore variables one by one. Method to perform uni-variate analysis will depend on whether the variable type is categorical or continuous. Let’s look at these methods and statistical measures for categorical and continuous variables individually:

**Continuous Variables:-** In case of continuous variables, we need to understand the central tendency and spread of the variable. These are measured using various statistical metrics visualization methods as shown below:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_31.png)**Note:** Univariate analysis is also used to highlight missing and outlier values. In the upcoming part of this series, we will look at methods to handle missing and outlier values. To know more about these methods, you can refer course [descriptive statistics from Udacity](https://www.udacity.com/course/ud827).

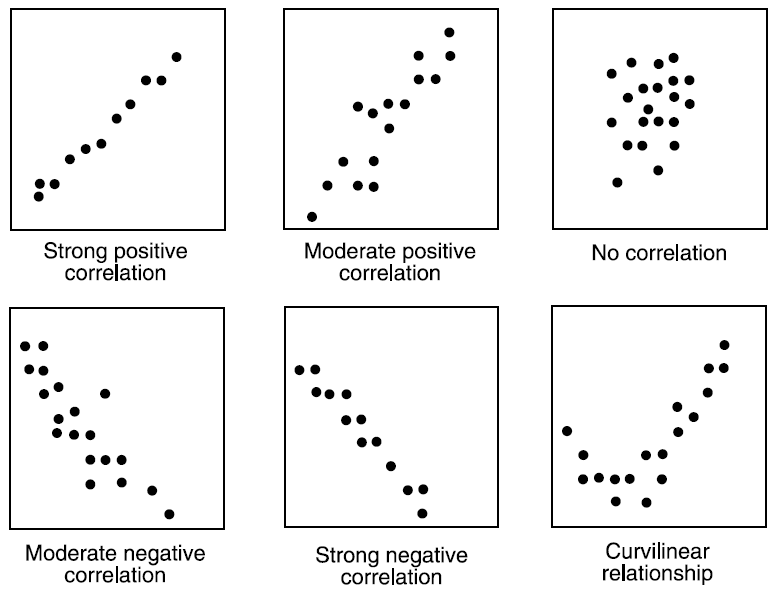
**Categorical Variables:-**For categorical variables, we’ll use frequency table to understand distribution of each category. We can also read as percentage of values under each category. It can be be measured using two metrics, **Count** and **Count%** against each category. Bar chart can be used as visualization.

**Bi-variate Analysis**

Bi-variate Analysis finds out the relationship between two variables. Here, we look for association and disassociation between variables at a pre-defined significance level. We can perform bi-variate analysis for any combination of categorical and continuous variables. The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous. Different methods are used to tackle these combinations during analysis process.

Let’s understand the possible combinations in detail:

**Continuous & Continuous:**While doing bi-variate analysis between two continuous variables, we should look at scatter plot. It is a nifty way to find out the relationship between two variables. The pattern of scatter plot indicates the relationship between variables. The relationship can be linear or non-linear.

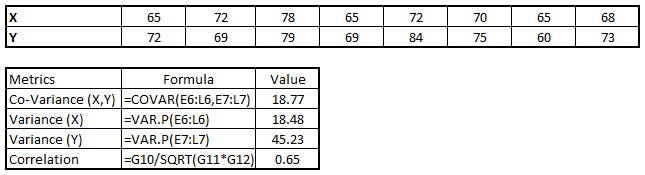
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_4.png)Scatter plot shows the relationship between two variable but does not indicates the strength of relationship amongst them. To find the strength of the relationship, we use Correlation. Correlation varies between -1 and +1.

* -1: perfect negative linear correlation
* +1:perfect positive linear correlation and
* 0: No correlation

Correlation can be derived using following formula:

**Correlation = Covariance(X,Y) / SQRT( Var(X)\* Var(Y))**

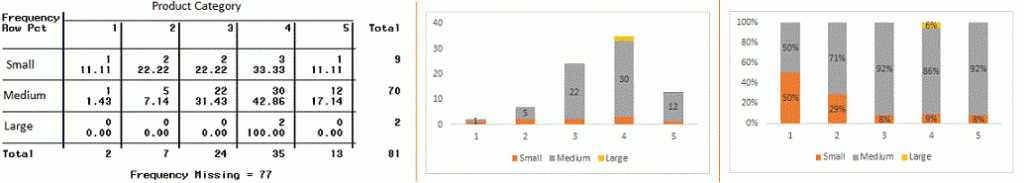
Various tools have function or functionality to identify correlation between variables. In Excel, function CORREL() is used to return the correlation between two variables and SAS uses procedure PROC CORR to identify the correlation. These function returns Pearson Correlation value to identify the relationship between two variables:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_51.png)

In above example, we have good positive relationship(0.65) between two variables X and Y.

**Categorical & Categorical:**To find the relationship between two categorical variables, we can use following methods:

* **Two-way table:** We can start analyzing the relationship by creating a two-way table of count and count%. The rows represents the category of one variable and the columns represent the categories of the other variable. We show count or count% of observations available in each combination of row and column categories.
* **Stacked Column Chart:**This method is more of a visual form of Two-way table.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_6.gif)

* **Chi-Square Test:** This test is used to derive the statistical significance of relationship between the variables. Also, it tests whether the evidence in the sample is strong enough to generalize that the relationship for a larger population as well. Chi-square is based on the difference between the expected and observed frequencies in one or more categories in the two-way table. It returns probability for the computed chi-square distribution with the degree of freedom.

Probability of 0: It indicates that both categorical variable are dependent

Probability of 1: It shows that both variables are independent.

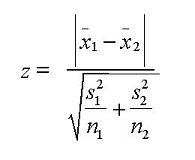
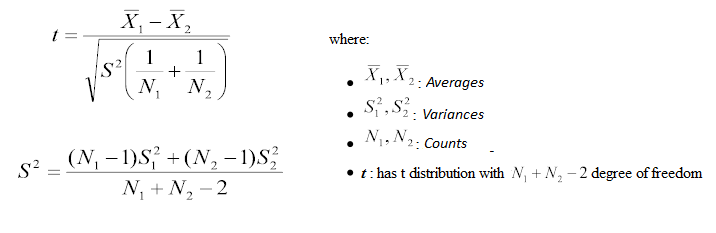
Probability less than 0.05: It indicates that the relationship between the variables is significant at 95% confidence. The chi-square test statistic for a test of independence of two categorical variables is found by:

[Data Exploration, Chi Square, Business Analytics](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_7.png)where *O* represents the observed frequency. *E* is the expected frequency under the null hypothesis and computed by:  
[Data Exploration, Chi Square, Business Analytics](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_exploration_8.png)  
From previous two-way table, the expected count for product category 1 to be of small size is  0.22. It is derived by taking the row total for Size (9) times the column total for Product category (2) then dividing by the sample size (81). This is procedure is conducted for each cell. Statistical Measures used to analyze the power of relationship are:

* Cramer’s V for Nominal Categorical Variable
* Mantel-Haenszed Chi-Square for ordinal categorical variable.

Different data science language and tools have specific methods to perform chi-square test. In SAS, we can use **Chisq** as an option with **Proc freq** to perform this test.

**Categorical & Continuous:**While exploring relation between categorical and continuous variables, we can draw box plots for each level of categorical variables. If levels are small in number, it will not show the statistical significance. To look at the statistical significance we can perform Z-test, T-test or ANOVA.

* **Z-Test/ T-Test:-** Either test assess whether mean of two groups are statistically different from each other or not.[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/ztestformula1.jpg)If the probability of Z is small then the difference of two averages is more significant. The T-test is very similar to Z-test but it is used when number of observation for both categories is less than 30.  
  [](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/ttest.png)
* **ANOVA:-**It assesses whether the average of more than two groups is statistically different.

**Example:** Suppose, we want to test the effect of five different exercises. For this, we recruit 20 men and assign one type of exercise to 4 men (5 groups). Their weights are recorded after a few weeks. We need to find out whether the effect of these exercises on them is significantly different or not. This can be done by comparing the weights of the 5 groups of 4 men each.

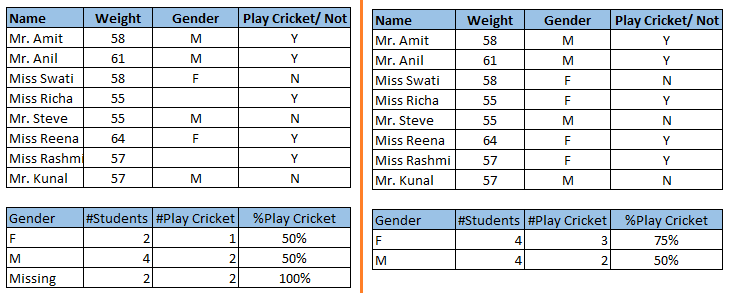
Till here, we have understood the first three stages of Data Exploration, Variable Identification, Uni-Variate and Bi-Variate analysis. We also looked at various statistical and visual methods to identify the relationship between variables.

Now, we will look at the methods of Missing values Treatment. More importantly, we will also look at why missing values occur in our data and why treating them is necessary.

**2. Missing Value Treatment**

**Why missing values treatment is required?**

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_Exploration_2_11.png)

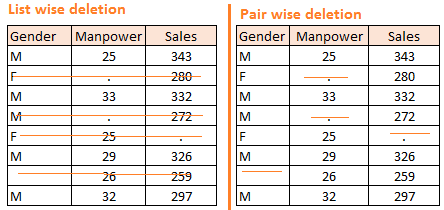
Notice the missing values in the image shown above: In the left scenario, we have not treated missing values. The inference from this data set is that the chances of playing cricket by males is higher than females. On the other hand, if you look at the second table, which shows data after treatment of missing values (based on gender), we can see that females have higher chances of playing cricket compared to males.

**Why my data has missing values?**

We looked at the importance of treatment of missing values in a dataset. Now, let’s identify the reasons for occurrence of these missing values. They may occur at two stages:

1. **Data Extraction**: It is possible that there are problems with extraction process. In such cases, we should double-check for correct data with data guardians. Some hashing procedures can also be used to make sure data extraction is correct. Errors at data extraction stage are typically easy to find and can be corrected easily as well.
2. **Data collection**: These errors occur at time of data collection and are harder to correct. They can be categorized in four types:
   * **Missing completely at random:** This is a case when the probability of missing variable is same for all observations. For example: respondents of data collection process decide that they will declare their earning after tossing a fair coin. If an head occurs, respondent declares his / her earnings & vice versa. Here each observation has equal chance of missing value.
   * **Missing at random:**This is a case when variable is missing at random and missing ratio varies for different values / level of other input variables. For example: We are collecting data for age and female has higher missing value compare to male.
   * **Missing that depends on unobserved predictors:** This is a case when the missing values are not random and are related to the unobserved input variable. For example: In a medical study, if a particular diagnostic causes discomfort, then there is higher chance of drop out from the study. This missing value is not at random unless we have included “discomfort” as an input variable for all patients.
   * **Missing that depends on the missing value itself:**This is a case when the probability of missing value is directly correlated with missing value itself. For example: People with higher or lower income are likely to provide non-response to their earning.

**Which are the methods to treat missing values ?**

1. **Deletion:** It is of two types: List Wise Deletion and Pair Wise Deletion.
   * In list wise deletion, we delete observations where any of the variable is missing. Simplicity is one of the major advantage of this method, but this method reduces the power of model because it reduces the sample size.
   * In pair wise deletion, we perform analysis with all cases in which the variables of interest are present. Advantage of this method is, it keeps as many cases available for analysis. One of the disadvantage of this method, it uses different sample size for different variables.  
      [](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Data_Exploration_2_2.png)
   * Deletion methods are used when the nature of missing data is “**Missing completely at random**” else non random missing values can bias the model output.
2. **Mean/ Mode/ Median Imputation:**Imputation is a method to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. Mean / Mode / Median imputation is one of the most frequently used methods. It consists of replacing the missing data for a given attribute by the mean or median (quantitative attribute) or mode (qualitative attribute) of all known values of that variable. It can be of two types:-
   * **Generalized Imputation:** In this case, we calculate the mean or median for all non missing values of that variable then replace missing value with mean or median. Like in above table, variable “**Manpower”** is missing so we take average of all non missing values of “**Manpower”**  (**28.33**) and then replace missing value with it.
   * **Similar case Imputation:** In this case, we calculate average for gender “**Male”**(29.75) and “**Female**” (25) individually of non missing values then replace the missing value based on gender. For “**Male**“, we will replace missing values of manpower with 29.75 and for “**Female**” with 25.
3. **Prediction Model**:  Prediction model is one of the sophisticated method for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data.  In this case, we divide our data set into two sets: One set with no missing values for the variable and another one with missing values. First data set become training data set of the model while second data set with missing values is test data set and variable with missing values is treated as target variable. Next, we create a model to predict target variable based on other attributes of the training data set and populate missing values of test data set.We can use regression, ANOVA, Logistic regression and various modeling technique to perform this. There are 2 drawbacks for this approach:
   * The model estimated values are usually more well-behaved than the true values
   * If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values.
4. **KNN Imputation:** In this method of imputation, the missing values of an attribute are imputed using the given number of attributes that are most similar to the attribute whose values are missing. The similarity of two attributes is determined using a distance function. It is also known to have certain advantage & disadvantages.
   * **Advantages:**
     + k-nearest neighbour can predict both qualitative & quantitative attributes
     + Creation of predictive model for each attribute with missing data is not required
     + Attributes with multiple missing values can be easily treated
     + Correlation structure of the data is taken into consideration
   * **Disadvantage:**
     + KNN algorithm is very time-consuming in analyzing large database. It searches through all the dataset looking for the most similar instances.
     + Choice of k-value is very critical. Higher value of k would include attributes which are significantly different from what we need whereas lower value of k implies missing out of significant attributes.

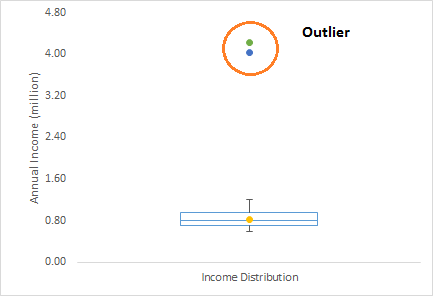
After dealing with missing values, the next task is to deal with outliers. Often, we tend to neglect outliers while building models. This is a discouraging practice. Outliers tend to make your data skewed and reduces accuracy. Let’s learn more about outlier treatment.

**3. Techniques of Outlier Detection and Treatment**

**What is an Outlier?**

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

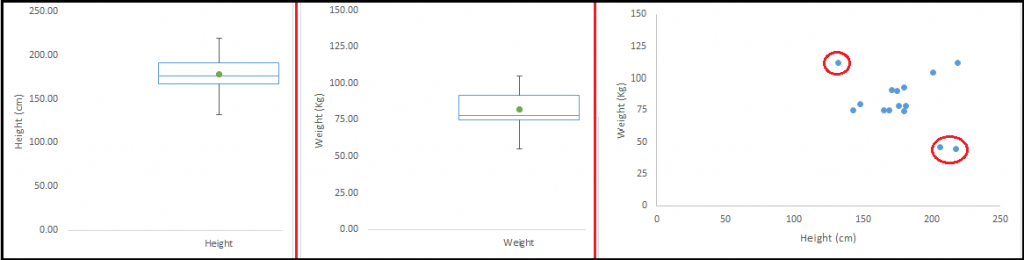
Let’s take an example, we do customer profiling and find out that the average annual income of customers is $0.8 million. But, there are two customers having annual income of $4 and $4.2 million. These two customers annual income is much higher than rest of the population. These two observations will be seen as Outliers.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Outlier.png)

**What are the types of Outliers?**

Outlier can be of two types: **Univariate** and **Multivariate**. Above, we have discussed the example of univariate outlier. These outliers can be found when we look at distribution of a single variable. Multi-variate outliers are outliers in an n-dimensional space. In order to find them, you have to look at distributions in multi-dimensions.

Let us understand this with an example. Let us say we are understanding the relationship between height and weight. Below, we have univariate and bivariate distribution for Height, Weight. Take a look at the box plot. We do not have any outlier (above and below 1.5\*IQR, most common method). Now look at the scatter plot. Here, we have two values below and one above the average in a specific segment of weight and height.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Outlier_21.png)

**What causes Outliers?**

Whenever we come across outliers, the ideal way to tackle them is to find out the reason of having these outliers. The method to deal with them would then depend on the reason of their occurrence. Causes of outliers can be classified in two broad categories:

1. **Artificial (Error) / Non-natural**
2. **Natural**.

Let’s understand various types of outliers in more detail:

* **Data Entry Errors:-** Human errors such as errors caused during data collection, recording, or entry can cause outliers in data. For example: Annual income of a customer is $100,000. Accidentally, the data entry operator puts an additional zero in the figure. Now the income becomes $1,000,000 which is 10 times higher. Evidently, this will be the outlier value when compared with rest of the population.
* **Measurement Error:**It is the most common source of outliers. This is caused when the measurement instrument used turns out to be faulty. For example: There are 10 weighing machines. 9 of them are correct, 1 is faulty. Weight measured by people on the faulty machine will be higher / lower than the rest of people in the group. The weights measured on faulty machine can lead to outliers.
* **Experimental Error:** Another cause of outliers is experimental error. For example: In a 100m sprint of 7 runners, one runner missed out on concentrating on the ‘Go’ call which caused him to start late. Hence, this caused the runner’s run time to be more than other runners. His total run time can be an outlier.
* **Intentional Outlier*:***This is commonly found in self-reported measures that involves sensitive data. For example: Teens would typically under report the amount of alcohol that they consume. Only a fraction of them would report actual value. Here actual values might look like outliers because rest of the teens are under reporting the consumption.
* **Data Processing Error:**Whenever we perform data mining, we extract data from multiple sources. It is possible that some manipulation or extraction errors may lead to outliers in the dataset.
* **Sampling error:** For instance, we have to measure the height of athletes. By mistake, we include a few basketball players in the sample. This inclusion is likely to cause outliers in the dataset.
* **Natural Outlier:**When an outlier is not artificial (due to error), it is a natural outlier. For instance: In my last assignment with one of the renowned insurance company, I noticed that the performance of top 50 financial advisors was far higher than rest of the population. Surprisingly, it was not due to any error. Hence, whenever we perform any data mining activity with advisors, we used to treat this segment separately.

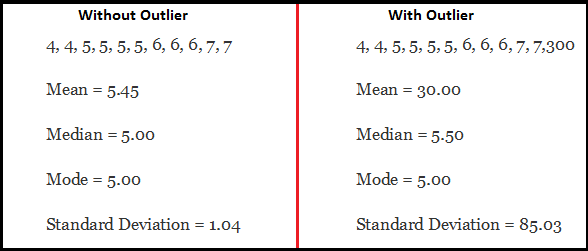
**What is the impact of Outliers on a dataset?**

Outliers can drastically change the results of the data analysis and statistical modeling. There are numerous unfavourable impacts of outliers in the data set:

* It increases the error variance and reduces the power of statistical tests
* If the outliers are non-randomly distributed, they can decrease normality
* They can bias or influence estimates that may be of substantive interest
* They can also impact the basic assumption of Regression, ANOVA and other statistical model assumptions.

To understand the impact deeply, let’s take an example to check what happens to a data set with and without outliers in the data set.

**Example:**

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Outlier_31.png)

As you can see, data set with outliers has significantly different mean and standard deviation. In the first scenario, we will say that average is 5.45. But with the outlier, average soars to 30. This would change the estimate completely.

**How to detect Outliers?**

Most commonly used method to detect outliers is visualization. We use various visualization methods, like **Box-plot**, **Histogram**, **Scatter Plot** (above, we have used box plot and scatter plot for visualization). Some analysts also various thumb rules to detect outliers. Some of them are:

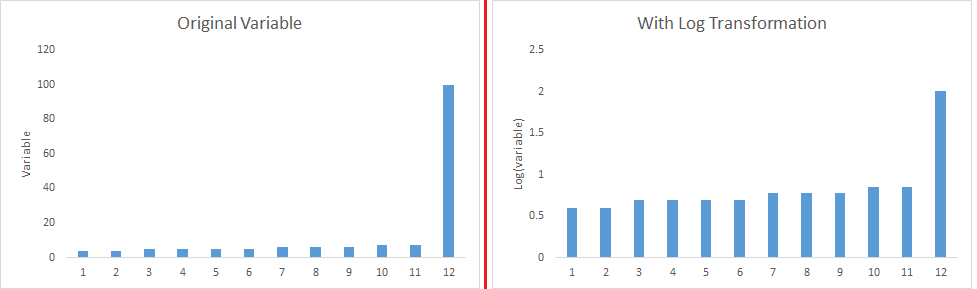
* Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR
* Use capping methods. Any value which out of range of 5th and 95th percentile can be considered as outlier
* Data points, three or more standard deviation away from mean are considered outlier
* Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding
* Bivariate and multivariate outliers are typically measured using either an index of influence or leverage, or distance. Popular indices such as Mahalanobis’ distance and Cook’s *D* are frequently used to detect outliers.
* In SAS, we can use PROC Univariate, PROC SGPLOT. To identify outliers and influential observation, we also look at statistical measure like STUDENT, COOKD, RSTUDENT and others.

**How to remove Outliers?**

Most of the ways to deal with outliers are similar to the methods of missing values like deleting observations, transforming them, binning them, treat them as a separate group, imputing values and other statistical methods. Here, we will discuss the common techniques used to deal with outliers:

**Deleting observations:**We delete outlier values if it is due to data entry error, data processing error or outlier observations are very small in numbers. We can also use trimming at both ends to remove outliers.

**Transforming and binning values:**Transforming variables can also eliminate outliers. Natural log of a value reduces the variation caused by extreme values. Binning is also a form of variable transformation. Decision Tree algorithm allows to deal with outliers well due to binning of variable. We can also use the process of assigning weights to different observations.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/Transformation_1.png)

**Imputing:**Like [imputation of missing values](https://www.analyticsvidhya.com/blog/2015/02/7-steps-data-exploration-preparation-building-model-part-2/), we can also impute outliers. We can use mean, median, mode imputation methods. Before imputing values, we should analyse if it is natural outlier or artificial. If it is artificial, we can go with imputing values. We can also use statistical model to predict values of outlier observation and after that we can impute it with predicted values.

**Treat separately:**If there are significant number of outliers, we should treat them separately in the statistical model. One of the approach is to treat both groups as two different groups and build individual model for both groups and then combine the output.

Till here, we have learnt about steps of data exploration, missing value treatment and techniques of outlier detection and treatment. These 3 stages will make your raw data better in terms of information availability and accuracy. Let’s now proceed to the final stage of data exploration. It is Feature Engineering.

**4. The Art of Feature Engineering**

**What is Feature Engineering?**

Feature engineering is the science (and art) of extracting more information from existing data. You are not adding any new data here, but you are actually making the data you already have more useful.

For example, let’s say you are trying to predict foot fall in a shopping mall based on dates. If you try and use the dates directly, you may not be able to extract meaningful insights from the data. This is because the foot fall is less affected by the day of the month than it is by the day of the week. Now this information about day of week is implicit in your data. You need to bring it out to make your model better.

This exercising of bringing out information from data in known as feature engineering.

**What is the process of Feature Engineering ?**

You perform feature engineering once you have completed the first 5 steps in data exploration – [Variable Identification, Univariate, Bivariate Analysis](https://www.analyticsvidhya.com/blog/2015/02/data-exploration-preparation-model/), [Missing Values Imputation](https://www.analyticsvidhya.com/blog/2015/02/7-steps-data-exploration-preparation-building-model-part-2/) and [Outliers Treatment](https://www.analyticsvidhya.com/blog/2015/02/outliers-detection-treatment-dataset/). Feature engineering itself can be divided in 2 steps:

* Variable transformation.
* Variable / Feature creation.

These two techniques are vital in data exploration and have a remarkable impact on the power of prediction. Let’s understand each of this step in more details.

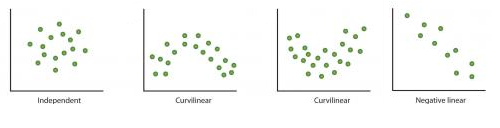
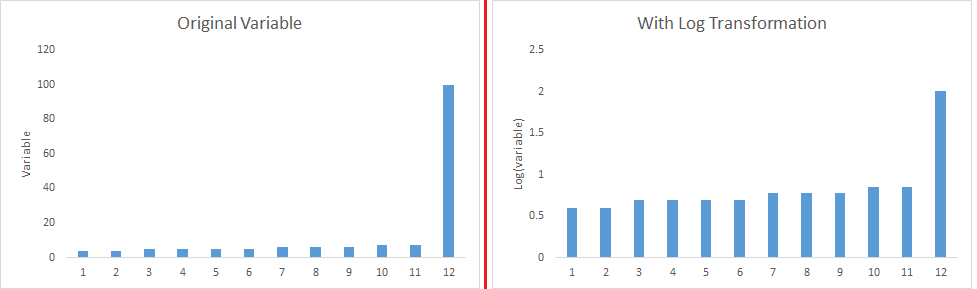
**What is Variable Transformation?**

In data modelling, transformation refers to the replacement of a variable by a function. For instance, replacing a variable x by the square / cube root or logarithm x is a transformation. In other words, transformation is a process that changes the distribution or relationship of a variable with others.

Let’s look at the situations when  variable transformation is useful.

**When should we use Variable Transformation?**

Below are the situations where variable transformation is a requisite:

* When we want to **change the scale** of a variable or standardize the values of a variable for better understanding. While this transformation is a must if you have data in different scales, this transformation does not change the shape of the variable distribution
* When we can **transform complex non-linear relationships into linear relationships**. Existence of a linear relationship between variables is easier to comprehend compared to a non-linear or curved relation. Transformation helps us to convert a non-linear relation into linear relation. Scatter plot can be used to find the relationship between two continuous variables. These transformations also improve the prediction. Log transformation is one of the commonly used transformation technique used in these situations.
* [](https://www.analyticsvidhya.com/wp-content/uploads/2015/03/Relation.png)**Symmetric distribution is preferred over skewed distribution** as it is easier to interpret and generate inferences. Some modeling techniques requires normal distribution of variables. So, whenever we have a skewed distribution, we can use transformations which reduce skewness. For right skewed distribution, we take square / cube root or logarithm of variable and for left skewed, we take square / cube or exponential of variables.[](https://www.analyticsvidhya.com/wp-content/uploads/2015/03/Transformation_1.png)
* Variable Transformation is also done from an**implementation point of view** (Human involvement). Let’s understand it more clearly. In one of my project on employee performance, I found that age has direct correlation with performance of the employee i.e. higher the age, better the performance. From an implementation stand point, launching age based progamme might present implementation challenge. However, categorizing the sales agents in three age group buckets of <30 years, 30-45 years and >45  and then formulating three different strategies for each group is a judicious approach. This categorization technique is known as Binning of Variables.

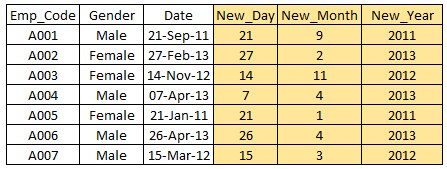
**What are the common methods of Variable Transformation?**

There are various methods used to transform variables. As discussed, some of them include square root, cube root, logarithmic, binning, reciprocal and many others. Let’s look at these methods in detail by highlighting the pros and cons of these transformation methods.

* **Logarithm:**Log of a variable is a common transformation method used to change the shape of distribution of the variable on a distribution plot. It is generally used for reducing right skewness of variables. Though, It can’t be applied to zero or negative values as well.
* **Square / Cube root:**The square and cube root of a variable has a sound effect on variable distribution. However, it is not as significant as logarithmic transformation. Cube root has its own advantage. It can be applied to negative values including zero. Square root can be applied to positive values including zero.
* **Binning:**It is used to categorize variables. It is performed on original values, percentile or frequency. Decision of categorization technique is based on business understanding. For example, we can categorize income in three categories, namely: High, Average and Low.We can also perform co-variate binning which depends on the value of more than one variables.

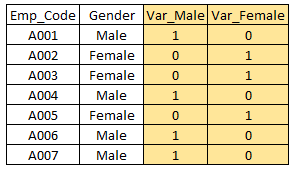
**What is Feature / Variable Creation & its Benefits?**

Feature / Variable creation is a process to generate a new variables / features based on existing variable(s). For example, say, we have date(dd-mm-yy) as an input variable in a data set. We can generate new variables like day, month, year, week, weekday that may have better relationship with target variable. This step is used to highlight the hidden relationship in a variable:



There are various techniques to create new features. Let’s look at the some of the commonly used methods:

* **Creating derived variables:** This refers to creating new variables from existing variable(s) using set of functions or different methods. Let’s look at it through “[**Titanic – Kaggle competition**](https://www.kaggle.com/c/titanic-gettingStarted/data)”. In this data set, variable age has missing values. To predict missing values, we used the salutation (Master, Mr, Miss, Mrs) of name as a new variable. How do we decide which variable to create? Honestly, this depends on business understanding of the analyst, his curiosity and the set of hypothesis he might have about the problem. Methods such as taking log of variables, binning variables and other methods of variable transformation can also be used to create new variables.
* **Creating dummy variables:**One of the most common application of dummy variable is to convert categorical variable into numerical variables. Dummy variables are also called Indicator Variables. It is useful to take categorical variable as a predictor in statistical models.  Categorical variable can take values 0 and 1. Let’s take a variable ‘gender’. We can produce two variables, namely, “**Var\_Male**” with values 1 (Male) and 0 (No male) and “**Var\_Female**” with values 1 (Female) and 0 (No Female). We can also create dummy variables for more than two classes of a categorical variables with n or n-1 dummy variables.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/03/Dummy.png)

**For further read, here is a**[**list of transformation / creation ideas**](https://www.analyticsvidhya.com/blog/2013/11/simple-manipulations-extract-data/)**which can be applied to your data.**

# Build a Predictive Model in 10 Minutes (using Python)

[**SUNIL RAY**](https://www.analyticsvidhya.com/blog/author/sunil-ray/)**, SEPTEMBER 23, 2015**

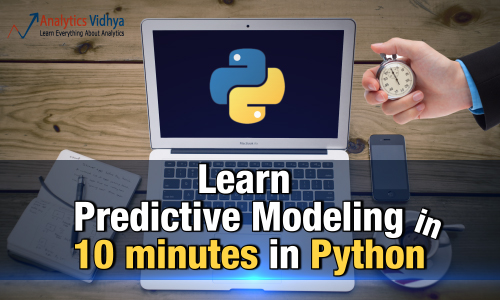
## Introduction

I came across this strategic virtue from Sun Tzu recently:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/09/6-strategic-principles-by-sun-tzu-13-728.jpg)

What has this to do with a data science blog? This is the essence of how you win competitions and [hackathons](http://datahack.analyticsvidhya.com/). You come in the competition better prepared than the competitors, you execute quickly, learn and iterate to bring out the best in you.

Last week, we published “[Perfect way to build a Predictive Model in less than 10 minutes using R](https://www.analyticsvidhya.com/blog/2015/09/perfect-build-predictive-model-10-minutes/)“. Any one can guess a quick follow up to this article. Given the rise of Python in last few years and its simplicity, it makes sense to have this tool kit ready for the Pythonists in the data science world. I will follow similar structure as previous article with my additional inputs at different stages of model building. These two articles will help you to build your first predictive model faster with better power. Most of the top data scientists and Kagglers build their first effective model quickly and submit. This not only helps them get a head start on the leader board, but also provides a bench mark solution to beat.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/09/predective1.jpg)

## Breaking Down the process of Predictive Modeling

I always focus on investing quality time during initial phase of model building like hypothesis generation / brain storming session(s) / discussion(s) or understanding the domain. All these activities help me to relate to the problem, which eventually leads me to design more powerful business solutions. There are good reasons why you should spend this time up front:

1. You have enough time to invest and you are fresh ( It has an impact)
2. You are not biased with other data points or thoughts (I always suggest, do hypothesis generation before deep diving in data)
3. At later stage, you would be in a hurry to complete the project and not able to spend quality time

This stage will need a quality time so I am not mentioning the timeline here, I would recommend you to make this as a standard practice. It will help you to build a better predictive models and result in less iteration of work at later stages. Let’s look at the remaining stages in first model build with timelines:

1. **Descriptive analysis on the Data – 50% time**
2. **Data treatment (Missing value and outlier fixing) – 40% time**
3. **Data Modelling – 4% time**
4. **Estimation of performance – 6% time**

P.S. This is the split of time spent only for the first model build

Let’s go through the process step by step (with estimates of time spent in each step):

### Stage 1: Descriptive Analysis / Data Exploration:

In my initial days as data scientist, data exploration used to take a lot of time for me. With time, I have automated a lot of operations on the data. Given that data prep takes up 50% of the work in building a first model, the benefits of automation are obvious. You can look at “[7 Steps of data exploration](https://www.analyticsvidhya.com/blog/2015/02/data-exploration-preparation-model/)” to look at the most common operations of data exploration.

Tavish has already mentioned in his article that with advanced [machine learning tools](https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/) coming in race, time taken to perform this task has been significantly reduced. Since this is our first benchmark model, we do away with any kind of feature engineering. Hence, the time you might need to do descriptive analysis is restricted to know missing values and big features which are directly visible. In my methodology, you will ***need 2 minutes*** to complete this step (Assumption, 100,000 observations in data set).

The operations I perform for my first model include:

1. Identify ID, Input and Target features
2. Identify categorical and numerical features
3. Identify columns with missing values

### Stage 2: Data Treatment (Missing values treatment):

There are various ways to deal with it. For our first model, we will focus on the smart and quick techniques to build your first effective model (These are already discussed by Tavish in his [article](https://www.analyticsvidhya.com/blog/2015/09/perfect-build-predictive-model-10-minutes/), I am adding a few methods)

* Create dummy flags for missing value(s) : It works, sometimes missing values itself carry a good amount of information.
* Impute missing value with mean/ median/ any other easiest method : Mean and Median imputation performs well, mostly people prefer to impute with mean value but in case of skewed distribution I would suggest you to go with median. Other Intelligent methods are imputing values by similar case mean and median imputation using other relevant features or building a model. For Example: In Titanic survival challenge, you can impute missing values of Age using salutation of passengers name Like “Mr.”, “Miss.”,”Mrs.”,”Master” and others and this has shown good impact on model performance.
* Impute missing value of categorical variable: Create a new level to impute categorical variable so that all missing value is coded as a single value say “New\_Cat” or you can look at the frequency mix and impute the missing value with value having higher frequency.

With such simple methods of data treatment, you can reduce the time to treat data to ***3-4 minutes***.

### Stage 3. Data Modelling :

I recommend to use any one of [GBM](https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/) / [Random Forest](https://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/) techniques, depending on the business problem. These two techniques are extremely effective to create a benchmark solution. I have seen data scientist are using these two methods often as their first model and in some cases it acts as a final model also. This will take maximum amount of time (***~4-5 minutes***).

### Stage 4. Estimation of Performance:

There are various methods to validate your model performance, I would suggest you to divide your train data set into Train and validate (ideally 70:30) and build model based on 70% of train data set. Now, cross-validate it using 30% of validate data set and evaluate the performance using evaluation metric. This finally takes ***1-2 minutes*** to execute and document.

Intent of this article is not to win the competition, but to establish a benchmark for our self. Let’s look at the python codes to perform above steps and build your first model with higher impact.

## Let’s start putting this into action

I have assumed you have done all the hypothesis generation first and you are good with basic data science using python.  I am illustrating this with an example of data science challenge. Let’s look at the structure:

**Step 1** : Import required libraries and read test and train data set. Append both.

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

import random

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

train=pd.read\_csv('C:/Users/Analytics Vidhya/Desktop/challenge/Train.csv')

test=pd.read\_csv('C:/Users/Analytics Vidhya/Desktop/challenge/Test.csv')

train['Type']='Train' #Create a flag for Train and Test Data set

test['Type']='Test'

fullData = pd.concat([train,test],axis=0) #Combined both Train and Test Data set

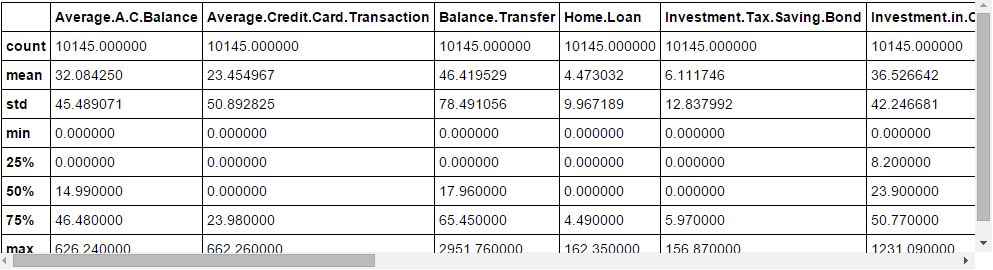
**Step 2**: Step 2 of the framework is not required in Python. On to the next step.

**Step 3**: View the column names / summary of the dataset

fullData.columns # This will show all the column names

fullData.head(10) # Show first 10 records of dataframe

fullData.describe() #You can look at summary of numerical fields by using describe() function

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/09/Capture_10.png)

**Step 4**: Identify the a) ID variables b)  Target variables c) Categorical Variables d) Numerical Variables e) Other Variables

ID\_col = ['REF\_NO']

target\_col = ["Account.Status"]

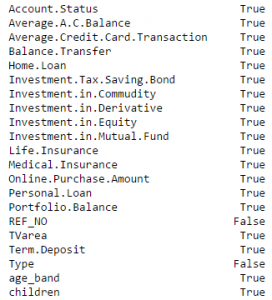
cat\_cols = ['children','age\_band','status','occupation','occupation\_partner','home\_status','family\_income','self\_employed', 'self\_employed\_partner','year\_last\_moved','TVarea','post\_code','post\_area','gender','region']

num\_cols= list(set(list(fullData.columns))-set(cat\_cols)-set(ID\_col)-set(target\_col)-set(data\_col))

other\_col=['Type'] #Test and Train Data set identifier

**Step 5** : Identify the variables with missing values and create a flag for those

fullData.isnull().any()#Will return the feature with True or False,True means have missing value else False

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/09/Capture11.png)

num\_cat\_cols = num\_cols+cat\_cols # Combined numerical and Categorical variables

#Create a new variable for each variable having missing value with VariableName\_NA

# and flag missing value with 1 and other with 0

for var in num\_cat\_cols:

if fullData[var].isnull().any()==True:

fullData[var+'\_NA']=fullData[var].isnull()\*1

**Step 6** : Impute Missing values

#Impute numerical missing values with mean

fullData[num\_cols] = fullData[num\_cols].fillna(fullData[num\_cols].mean(),inplace=True)

#Impute categorical missing values with -9999

fullData[cat\_cols] = fullData[cat\_cols].fillna(value = -9999)

**Step 7** : Create a label encoders for categorical variables and split the data set to train & test, further split the train data set to Train and Validate

#create label encoders for categorical features

for var in cat\_cols:

number = LabelEncoder()

fullData[var] = number.fit\_transform(fullData[var].astype('str'))

#Target variable is also a categorical so convert it

fullData["Account.Status"] = number.fit\_transform(fullData["Account.Status"].astype('str'))

train=fullData[fullData['Type']=='Train']

test=fullData[fullData['Type']=='Test']

train['is\_train'] = np.random.uniform(0, 1, len(train)) <= .75

Train, Validate = train[train['is\_train']==True], train[train['is\_train']==False]

**Step 8** : Pass the imputed and dummy (missing values flags) variables into the modelling process. I am using random forest to predict the class

features=list(set(list(fullData.columns))-set(ID\_col)-set(target\_col)-set(other\_col))

x\_train = Train[list(features)].values

y\_train = Train["Account.Status"].values

x\_validate = Validate[list(features)].values

y\_validate = Validate["Account.Status"].values

x\_test=test[list(features)].values

random.seed(100)

rf = RandomForestClassifier(n\_estimators=1000)

rf.fit(x\_train, y\_train)

**Step 9** : Check performance and make predictions

status = rf.predict\_proba(x\_validate)

fpr, tpr, \_ = roc\_curve(y\_validate, status[:,1])

roc\_auc = auc(fpr, tpr)

print roc\_auc

final\_status = rf.predict\_proba(x\_test)

test["Account.Status"]=final\_status[:,1]

test.to\_csv('C:/Users/Analytics Vidhya/Desktop/model\_output.csv',columns=['REF\_NO','Account.Status'])

# A Comprehensive Guide to Understand and Implement Text Classification in Python

[**SHIVAM BANSAL**](https://www.analyticsvidhya.com/blog/author/shivam5992/)**, APRIL 23, 2018**

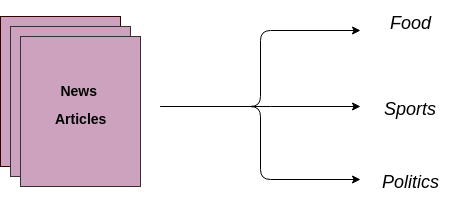
## Introduction

One of the widely used natural language processing task in different business problems is “Text Classification”. The goal of text classification is to automatically classify the text documents into one or more defined categories. Some examples of text classification are:

* Understanding audience sentiment from social media,
* Detection of spam and non-spam emails,
* Auto tagging of customer queries, and
* Categorization of news articles into defined topics.

## Table of Contents

In this article, I will explain about the text classification and the step by step process to implement it in python.



Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for train a classifier. An end-to-end text classification pipeline is composed of three main components:

**1. Dataset Preparation:**The first step is the Dataset Preparation step which includes the process of loading a dataset and performing basic pre-processing. The dataset is then splitted into train and validation sets.  
**2. Feature Engineering:**The next step is the Feature Engineering in which the raw dataset is transformed into flat features which can be used in a machine learning model. This step also includes the process of creating new features from the existing data.  
**3. Model Training:**The final step is the Model Building step in which a machine learning model is trained on a labelled dataset.

**4. Improve Performance of Text Classifier:**In this article, we will also look at the different ways to improve the performance of text classifiers.

***Note*** : This article does not narrate NLP tasks in depth. If you want to revise the basics and come back here, you can always go through [*this article.*](https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/)

## Getting your machine ready

Lets implement basic components in a step by step manner in order to create a text classification framework in python. To start with, import all the required libraries.

You would need requisite libraries to run this code – you can install them at their individual official links

* [Pandas](https://pandas.pydata.org/pandas-docs/stable/install.html)
* [Scikit-learn](http://scikit-learn.org/stable/install.html)
* [XGBoost](http://xgboost.readthedocs.io/en/latest/build.html)
* [TextBlob](http://textblob.readthedocs.io/en/dev/install.html)
* [Keras](https://keras.io/#installation)

# libraries for dataset preparation, feature engineering, model training

from sklearn import model\_selection, preprocessing, linear\_model, naive\_bayes, metrics, svm

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn import decomposition, ensemble

import pandas, xgboost, numpy, textblob, string

from keras.preprocessing import text, sequence

from keras import layers, models, optimizers

## 1. Dataset preparation

For the purpose of this article, I am the using dataset of amazon reviews which can be [downloaded at this link](https://gist.github.com/kunalj101/ad1d9c58d338e20d09ff26bcc06c4235). The dataset consists of 3.6M text reviews and their labels, we will use only a small fraction of data. To prepare the dataset, load the downloaded data into a pandas dataframe containing two columns – text and label. ([Source](https://drive.google.com/drive/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M))

# load the dataset

data = open('data/corpus').read()

labels, texts = [], []

for i, line in enumerate(data.split("\n")):

content = line.split()

labels.append(content[0])

texts.append(" ".join(content[1:]))

# create a dataframe using texts and lables

trainDF = pandas.DataFrame()

trainDF['text'] = texts

trainDF['label'] = labels

Next, we will split the dataset into training and validation sets so that we can train and test classifier. Also, we will encode our target column so that it can be used in machine learning models.

# split the dataset into training and validation datasets

train\_x, valid\_x, train\_y, valid\_y = model\_selection.train\_test\_split(trainDF['text'], trainDF['label'])

# label encode the target variable

encoder = preprocessing.LabelEncoder()

train\_y = encoder.fit\_transform(train\_y)

valid\_y = encoder.fit\_transform(valid\_y)

## 2. Feature Engineering

The next step is the feature engineering step. In this step, raw text data will be transformed into feature vectors and new features will be created using the existing dataset. We will implement the following different ideas in order to obtain relevant features from our dataset.

2.1 Count Vectors as features  
2.2 TF-IDF Vectors as features

* Word level
* N-Gram level
* Character level

2.3 Word Embeddings as features  
2.4 Text / NLP based features  
2.5 Topic Models as features

Lets look at the implementation of these ideas in detail.

### 2.1 Count Vectors as features

Count Vector is a matrix notation of the dataset in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document.

# create a count vectorizer object

count\_vect = CountVectorizer(analyzer='word', token\_pattern=r'\w{1,}')

count\_vect.fit(trainDF['text'])

# transform the training and validation data using count vectorizer object

xtrain\_count = count\_vect.transform(train\_x)

xvalid\_count = count\_vect.transform(valid\_x)

### 2.2 TF-IDF Vectors as features

TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed by two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)  
IDF(t) = log\_e(Total number of documents / Number of documents with term t in it)

TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

**a. Word Level TF-IDF :** Matrix representing tf-idf scores of every term in different documents  
**b. N-gram Level TF-IDF :** N-grams are the combination of N terms together. This Matrix representing tf-idf scores of N-grams  
**c. Character Level TF-IDF :** Matrix representing tf-idf scores of character level n-grams in the corpus

# word level tf-idf

tfidf\_vect = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', max\_features=5000)

tfidf\_vect.fit(trainDF['text'])

xtrain\_tfidf = tfidf\_vect.transform(train\_x)

xvalid\_tfidf = tfidf\_vect.transform(valid\_x)

# ngram level tf-idf

tfidf\_vect\_ngram = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram.fit(trainDF['text'])

xtrain\_tfidf\_ngram = tfidf\_vect\_ngram.transform(train\_x)

xvalid\_tfidf\_ngram = tfidf\_vect\_ngram.transform(valid\_x)

# characters level tf-idf

tfidf\_vect\_ngram\_chars = TfidfVectorizer(analyzer='char', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram\_chars.fit(trainDF['text'])

xtrain\_tfidf\_ngram\_chars = tfidf\_vect\_ngram\_chars.transform(train\_x)

xvalid\_tfidf\_ngram\_chars = tfidf\_vect\_ngram\_chars.transform(valid\_x)

### 2.3 Word Embeddings

A word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. Word embeddings can be trained using the input corpus itself or can be generated using pre-trained word embeddings such as **Glove, FastText,**and**Word2Vec**. Any one of them can be downloaded and used as transfer learning. One can read more about word embeddings [here.](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)

Following snnipet shows how to use pre-trained word embeddings in the model. There are four essential steps:

1. Loading the pretrained word embeddings
2. Creating a tokenizer object
3. Transforming text documents to sequence of tokens and pad them
4. Create a mapping of token and their respective embeddings

You can download the pre-trained word embeddings from [here](https://s3-us-west-1.amazonaws.com/fasttext-vectors/wiki-news-300d-1M.vec.zip)

# load the pre-trained word-embedding vectors

embeddings\_index = {}

for i, line in enumerate(open('data/wiki-news-300d-1M.vec')):

values = line.split()

embeddings\_index[values[0]] = numpy.asarray(values[1:], dtype='float32')

# create a tokenizer

token = text.Tokenizer()

token.fit\_on\_texts(trainDF['text'])

word\_index = token.word\_index

# convert text to sequence of tokens and pad them to ensure equal length vectors

train\_seq\_x = sequence.pad\_sequences(token.texts\_to\_sequences(train\_x), maxlen=70)

valid\_seq\_x = sequence.pad\_sequences(token.texts\_to\_sequences(valid\_x), maxlen=70)

# create token-embedding mapping

embedding\_matrix = numpy.zeros((len(word\_index) + 1, 300))

for word, i in word\_index.items():

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

### 2.4 Text / NLP based features

A number of extra text based features can also be created which sometimes are helpful for improving text classification models. Some examples are:

1. Word Count of the documents – total number of words in the documents
2. Character Count of the documents – total number of characters in the documents
3. Average Word Density of the documents – average length of the words used in the documents
4. Puncutation Count in the Complete Essay – total number of punctuation marks in the documents
5. Upper Case Count in the Complete Essay – total number of upper count words in the documents
6. Title Word Count in the Complete Essay – total number of proper case (title) words in the documents
7. Frequency distribution of Part of Speech Tags:
   * Noun Count
   * Verb Count
   * Adjective Count
   * Adverb Count
   * Pronoun Count

These features are highly experimental ones and should be used according to the problem statement only.

trainDF['char\_count'] = trainDF['text'].apply(len)

trainDF['word\_count'] = trainDF['text'].apply(lambda x: len(x.split()))

trainDF['word\_density'] = trainDF['char\_count'] / (trainDF['word\_count']+1)

trainDF['punctuation\_count'] = trainDF['text'].apply(lambda x: len("".join(\_ for \_ in x if \_ in string.punctuation)))

trainDF['title\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.istitle()]))

trainDF['upper\_case\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.isupper()]))

pos\_family = {

'noun' : ['NN','NNS','NNP','NNPS'],

'pron' : ['PRP','PRP$','WP','WP$'],

'verb' : ['VB','VBD','VBG','VBN','VBP','VBZ'],

'adj' : ['JJ','JJR','JJS'],

'adv' : ['RB','RBR','RBS','WRB']

}

# function to check and get the part of speech tag count of a words in a given sentence

def check\_pos\_tag(x, flag):

cnt = 0

try:

wiki = textblob.TextBlob(x)

for tup in wiki.tags:

ppo = list(tup)[1]

if ppo in pos\_family[flag]:

cnt += 1

except:

pass

return cnt

trainDF['noun\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'noun'))

trainDF['verb\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'verb'))

trainDF['adj\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'adj'))

trainDF['adv\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'adv'))

trainDF['pron\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'pron'))

### 2.5 Topic Models as features

Topic Modelling is a technique to identify the groups of words (called a topic) from a collection of documents that contains best information in the collection. I have used **Latent Dirichlet Allocation** for generating Topic Modelling Features. LDA is an iterative model which starts from a fixed number of topics. Each topic is represented as a distribution over words, and each document is then represented as a distribution over topics. Although the tokens themselves are meaningless, the probability distributions over words provided by the topics provide a sense of the different ideas contained in the documents. One can read more about topic modelling [here](https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/)

Lets see its implementation:

# train a LDA Model

lda\_model = decomposition.LatentDirichletAllocation(n\_components=20, learning\_method='online', max\_iter=20)

X\_topics = lda\_model.fit\_transform(xtrain\_count)

topic\_word = lda\_model.components\_

vocab = count\_vect.get\_feature\_names()

# view the topic models

n\_top\_words = 10

topic\_summaries = []

for i, topic\_dist in enumerate(topic\_word):

topic\_words = numpy.array(vocab)[numpy.argsort(topic\_dist)][:-(n\_top\_words+1):-1]

topic\_summaries.append(' '.join(topic\_words))

## 3. Model Building

The final step in the text classification framework is to train a classifier using the features created in the previous step. There are many different choices of machine learning models which can be used to train a final model. We will implement following different classifiers for this purpose:

1. Naive Bayes Classifier
2. Linear Classifier
3. Support Vector Machine
4. Bagging Models
5. Boosting Models
6. Shallow Neural Networks
7. Deep Neural Networks
   * Convolutional Neural Network (CNN)
   * Long Short Term Modelr (LSTM)
   * Gated Recurrent Unit (GRU)
   * Bidirectional RNN
   * Recurrent Convolutional Neural Network (RCNN)
   * Other Variants of Deep Neural Networks

Lets implement these models and understand their details. The following function is a utility function which can be used to train a model. It accepts the classifier, feature\_vector of training data, labels of training data and feature vectors of valid data as inputs. Using these inputs, the model is trained and accuracy score is computed.

def train\_model(classifier, feature\_vector\_train, label, feature\_vector\_valid, is\_neural\_net=False):

# fit the training dataset on the classifier

classifier.fit(feature\_vector\_train, label)

# predict the labels on validation dataset

predictions = classifier.predict(feature\_vector\_valid)

if is\_neural\_net:

predictions = predictions.argmax(axis=-1)

return metrics.accuracy\_score(predictions, valid\_y)

### 3.1 Naive Bayes

Implementing a naive bayes model using sklearn implementation with different features

Naive Bayes is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature [here .](https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/)

# Naive Bayes on Count Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_count, train\_y, xvalid\_count)

print "NB, Count Vectors: ", accuracy

# Naive Bayes on Word Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf, train\_y, xvalid\_tfidf)

print "NB, WordLevel TF-IDF: ", accuracy

# Naive Bayes on Ngram Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram)

print "NB, N-Gram Vectors: ", accuracy

# Naive Bayes on Character Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram\_chars, train\_y, xvalid\_tfidf\_ngram\_chars)

print "NB, CharLevel Vectors: ", accuracy

NB, Count Vectors: 0.7004

NB, WordLevel TF-IDF: 0.7024

NB, N-Gram Vectors: 0.5344

NB, CharLevel Vectors: 0.6872

### 3.2 Linear Classifier

Implementing a Linear Classifier (Logistic Regression)

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function. One can read more about logistic regression [here](https://www.analyticsvidhya.com/blog/2015/10/basics-logistic-regression/)

# Linear Classifier on Count Vectors

accuracy = train\_model(linear\_model.LogisticRegression(), xtrain\_count, train\_y, xvalid\_count)

print "LR, Count Vectors: ", accuracy

# Linear Classifier on Word Level TF IDF Vectors

accuracy = train\_model(linear\_model.LogisticRegression(), xtrain\_tfidf, train\_y, xvalid\_tfidf)

print "LR, WordLevel TF-IDF: ", accuracy

# Linear Classifier on Ngram Level TF IDF Vectors

accuracy = train\_model(linear\_model.LogisticRegression(), xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram)

print "LR, N-Gram Vectors: ", accuracy

# Linear Classifier on Character Level TF IDF Vectors

accuracy = train\_model(linear\_model.LogisticRegression(), xtrain\_tfidf\_ngram\_chars, train\_y, xvalid\_tfidf\_ngram\_chars)

print "LR, CharLevel Vectors: ", accuracy

LR, Count Vectors: 0.7048

LR, WordLevel TF-IDF: 0.7056

LR, N-Gram Vectors: 0.4896

LR, CharLevel Vectors: 0.7012

### 3.3 Implementing a SVM Model

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The model extracts a best possible hyper-plane / line that segregates the two classes. One can read more about it [here](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

# SVM on Ngram Level TF IDF Vectors

accuracy = train\_model(svm.SVC(), xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram)

print "SVM, N-Gram Vectors: ", accuracy

SVM, N-Gram Vectors: 0.5296

### 3.4 Bagging Model

Implementing a Random Forest Model

Random Forest models are a type of ensemble models, particularly bagging models. They are part of the tree based model family. One can read more about Bagging and random forests [here](https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/)

# RF on Count Vectors

accuracy = train\_model(ensemble.RandomForestClassifier(), xtrain\_count, train\_y, xvalid\_count)

print "RF, Count Vectors: ", accuracy

# RF on Word Level TF IDF Vectors

accuracy = train\_model(ensemble.RandomForestClassifier(), xtrain\_tfidf, train\_y, xvalid\_tfidf)

print "RF, WordLevel TF-IDF: ", accuracy

RF, Count Vectors: 0.6972

RF, WordLevel TF-IDF: 0.6988

### 3.5 Boosting Model

Implementing Xtereme Gradient Boosting Model

Boosting models are another type of ensemble models part of tree based models. Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones. A weak learner is defined to be a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing). Read more about these models [here](https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/)

# Extereme Gradient Boosting on Count Vectors

accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_count.tocsc(), train\_y, xvalid\_count.tocsc())

print "Xgb, Count Vectors: ", accuracy

# Extereme Gradient Boosting on Word Level TF IDF Vectors

accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_tfidf.tocsc(), train\_y, xvalid\_tfidf.tocsc())

print "Xgb, WordLevel TF-IDF: ", accuracy

# Extereme Gradient Boosting on Character Level TF IDF Vectors

accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_tfidf\_ngram\_chars.tocsc(), train\_y, xvalid\_tfidf\_ngram\_chars.tocsc())

print "Xgb, CharLevel Vectors: ", accuracy

/usr/local/lib/python2.7/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.

if diff:

/usr/local/lib/python2.7/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.

if diff:

Xgb, Count Vectors: 0.6324

Xgb, WordLevel TF-IDF: 0.6364

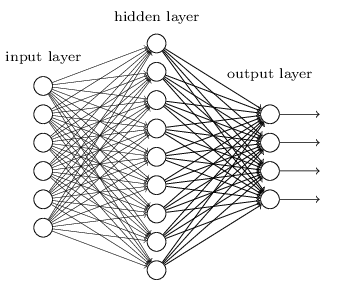
Xgb, CharLevel Vectors: 0.6548

/usr/local/lib/python2.7/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.

if diff:

### 3.6 Shallow Neural Networks

A neural network is a mathematical model that is designed to behave similar to biological neurons and nervous system. These models are used to recognize complex patterns and relationships that exists within a labelled data. A shallow neural network contains mainly three types of layers – input layer, hidden layer, and output layer. Read more about neural networks [here](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)



def create\_model\_architecture(input\_size):

# create input layer

input\_layer = layers.Input((input\_size, ), sparse=True)

# create hidden layer

hidden\_layer = layers.Dense(100, activation="relu")(input\_layer)

# create output layer

output\_layer = layers.Dense(1, activation="sigmoid")(hidden\_layer)

classifier = models.Model(inputs = input\_layer, outputs = output\_layer)

classifier.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return classifier

classifier = create\_model\_architecture(xtrain\_tfidf\_ngram.shape[1])

accuracy = train\_model(classifier, xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram, is\_neural\_net=True)

print "NN, Ngram Level TF IDF Vectors", accuracy

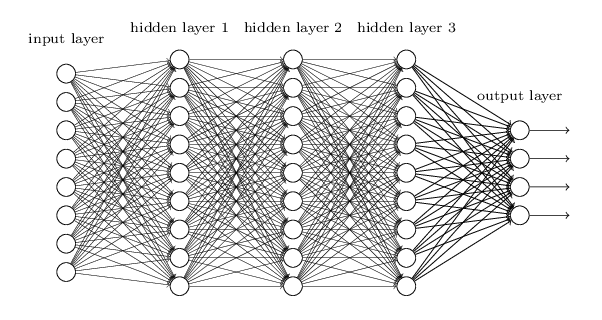
Epoch 1/1

7500/7500 [==============================] - 1s 67us/step - loss: 0.6909

NN, Ngram Level TF IDF Vectors 0.5296

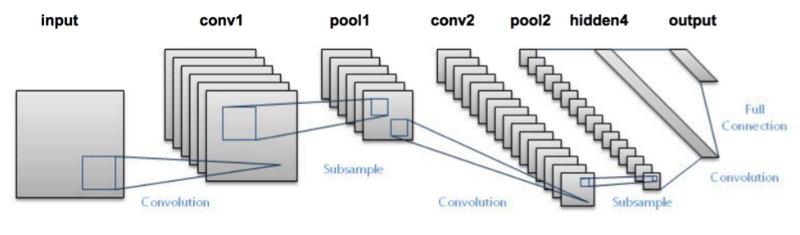
### 3.7 Deep Neural Networks

Deep Neural Networks are more complex neural networks in which the hidden layers performs much more complex operations than simple sigmoid or relu activations. Different types of deep learning models can be applied in text classification problems.



#### 3.7.1 Convolutional Neural Network

In Convolutional neural networks, convolutions over the input layer are used to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters and combines their results.



Read more about Convolutional Neural Networks [here](https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/)

def create\_cnn():

# Add an Input Layer

input\_layer = layers.Input((70, ))

# Add the word embedding Layer

embedding\_layer = layers.Embedding(len(word\_index) + 1, 300, weights=[embedding\_matrix], trainable=False)(input\_layer)

embedding\_layer = layers.SpatialDropout1D(0.3)(embedding\_layer)

# Add the convolutional Layer

conv\_layer = layers.Convolution1D(100, 3, activation="relu")(embedding\_layer)

# Add the pooling Layer

pooling\_layer = layers.GlobalMaxPool1D()(conv\_layer)

# Add the output Layers

output\_layer1 = layers.Dense(50, activation="relu")(pooling\_layer)

output\_layer1 = layers.Dropout(0.25)(output\_layer1)

output\_layer2 = layers.Dense(1, activation="sigmoid")(output\_layer1)

# Compile the model

model = models.Model(inputs=input\_layer, outputs=output\_layer2)

model.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return model

classifier = create\_cnn()

accuracy = train\_model(classifier, train\_seq\_x, train\_y, valid\_seq\_x, is\_neural\_net=True)

print "CNN, Word Embeddings", accuracy

Epoch 1/1

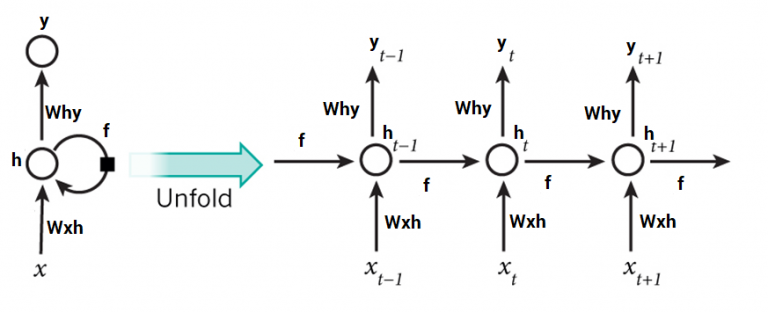
7500/7500 [==============================] - 12s 2ms/step - loss: 0.5847

CNN, Word Embeddings 0.5296

#### 3.7.2 Recurrent Neural Network – LSTM

Unlike Feed-forward neural networks in which activation outputs are propagated only in one direction, the activation outputs from neurons propagate in both directions (from inputs to outputs and from outputs to inputs) in Recurrent Neural Networks. This creates loops in the neural network architecture which acts as a ‘memory state’ of the neurons. This state allows the neurons an ability to remember what have been learned so far.

The memory state in RNNs gives an advantage over traditional neural networks but a problem called Vanishing Gradient is associated with them. In this problem, while learning with a large number of layers, it becomes really hard for the network to learn and tune the parameters of the earlier layers. To address this problem, A new type of RNNs called LSTMs (Long Short Term Memory) Models have been developed.



Read more about LSTMs [here](https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

def create\_rnn\_lstm():

# Add an Input Layer

input\_layer = layers.Input((70, ))

# Add the word embedding Layer

embedding\_layer = layers.Embedding(len(word\_index) + 1, 300, weights=[embedding\_matrix], trainable=False)(input\_layer)

embedding\_layer = layers.SpatialDropout1D(0.3)(embedding\_layer)

# Add the LSTM Layer

lstm\_layer = layers.LSTM(100)(embedding\_layer)

# Add the output Layers

output\_layer1 = layers.Dense(50, activation="relu")(lstm\_layer)

output\_layer1 = layers.Dropout(0.25)(output\_layer1)

output\_layer2 = layers.Dense(1, activation="sigmoid")(output\_layer1)

# Compile the model

model = models.Model(inputs=input\_layer, outputs=output\_layer2)

model.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return model

classifier = create\_rnn\_lstm()

accuracy = train\_model(classifier, train\_seq\_x, train\_y, valid\_seq\_x, is\_neural\_net=True)

print "RNN-LSTM, Word Embeddings", accuracy

Epoch 1/1

7500/7500 [==============================] - 22s 3ms/step - loss: 0.6899

RNN-LSTM, Word Embeddings 0.5124

#### 3.7.3 Recurrent Neural Network – GRU

Gated Recurrent Units are another form of recurrent neural networks. Lets add a layer of GRU instead of LSTM in our network.

def create\_rnn\_gru():

# Add an Input Layer

input\_layer = layers.Input((70, ))

# Add the word embedding Layer

embedding\_layer = layers.Embedding(len(word\_index) + 1, 300, weights=[embedding\_matrix], trainable=False)(input\_layer)

embedding\_layer = layers.SpatialDropout1D(0.3)(embedding\_layer)

# Add the GRU Layer

lstm\_layer = layers.GRU(100)(embedding\_layer)

# Add the output Layers

output\_layer1 = layers.Dense(50, activation="relu")(lstm\_layer)

output\_layer1 = layers.Dropout(0.25)(output\_layer1)

output\_layer2 = layers.Dense(1, activation="sigmoid")(output\_layer1)

# Compile the model

model = models.Model(inputs=input\_layer, outputs=output\_layer2)

model.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return model

classifier = create\_rnn\_gru()

accuracy = train\_model(classifier, train\_seq\_x, train\_y, valid\_seq\_x, is\_neural\_net=True)

print "RNN-GRU, Word Embeddings", accuracy

Epoch 1/1

7500/7500 [==============================] - 19s 3ms/step - loss: 0.6898

RNN-GRU, Word Embeddings 0.5124

#### 3.7.4 Bidirectional RNN

RNN layers can be wrapped in Bidirectional layers as well. Lets wrap our GRU layer in bidirectional layer.

def create\_bidirectional\_rnn():

# Add an Input Layer

input\_layer = layers.Input((70, ))

# Add the word embedding Layer

embedding\_layer = layers.Embedding(len(word\_index) + 1, 300, weights=[embedding\_matrix], trainable=False)(input\_layer)

embedding\_layer = layers.SpatialDropout1D(0.3)(embedding\_layer)

# Add the LSTM Layer

lstm\_layer = layers.Bidirectional(layers.GRU(100))(embedding\_layer)

# Add the output Layers

output\_layer1 = layers.Dense(50, activation="relu")(lstm\_layer)

output\_layer1 = layers.Dropout(0.25)(output\_layer1)

output\_layer2 = layers.Dense(1, activation="sigmoid")(output\_layer1)

# Compile the model

model = models.Model(inputs=input\_layer, outputs=output\_layer2)

model.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return model

classifier = create\_bidirectional\_rnn()

accuracy = train\_model(classifier, train\_seq\_x, train\_y, valid\_seq\_x, is\_neural\_net=True)

print "RNN-Bidirectional, Word Embeddings", accuracy

Epoch 1/1

7500/7500 [==============================] - 32s 4ms/step - loss: 0.6889

RNN-Bidirectional, Word Embeddings 0.5124

#### 3.7.5 Recurrent Convolutional Neural Network

Once the essential architectures have been tried out, one can try different variants of these layers such as recurrent convolutional neural network. Another variants can be:

1. Hierarichial Attention Networks
2. Sequence to Sequence Models with Attention
3. Bidirectional Recurrent Convolutional Neural Networks
4. CNNs and RNNs with more number of layers

def create\_rcnn():

# Add an Input Layer

input\_layer = layers.Input((70, ))

# Add the word embedding Layer

embedding\_layer = layers.Embedding(len(word\_index) + 1, 300, weights=[embedding\_matrix], trainable=False)(input\_layer)

embedding\_layer = layers.SpatialDropout1D(0.3)(embedding\_layer)

# Add the recurrent layer

rnn\_layer = layers.Bidirectional(layers.GRU(50, return\_sequences=True))(embedding\_layer)

# Add the convolutional Layer

conv\_layer = layers.Convolution1D(100, 3, activation="relu")(embedding\_layer)

# Add the pooling Layer

pooling\_layer = layers.GlobalMaxPool1D()(conv\_layer)

# Add the output Layers

output\_layer1 = layers.Dense(50, activation="relu")(pooling\_layer)

output\_layer1 = layers.Dropout(0.25)(output\_layer1)

output\_layer2 = layers.Dense(1, activation="sigmoid")(output\_layer1)

# Compile the model

model = models.Model(inputs=input\_layer, outputs=output\_layer2)

model.compile(optimizer=optimizers.Adam(), loss='binary\_crossentropy')

return model

classifier = create\_rcnn()

accuracy = train\_model(classifier, train\_seq\_x, train\_y, valid\_seq\_x, is\_neural\_net=True)

print "CNN, Word Embeddings", accuracy

Epoch 1/1

7500/7500 [==============================] - 11s 1ms/step - loss: 0.6902

CNN, Word Embeddings 0.5124

## Improving Text Classification Models

While the above framework can be applied to a number of text classification problems, but to achieve a good accuracy some improvements can be done in the overall framework. For example, following are some tips to improve the performance of text classification models and this framework.

**1. Text Cleaning :** text cleaning can help to reducue the noise present in text data in the form of stopwords, punctuations marks, suffix variations etc. This [article](https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/) can help to understand how to implement text classification in detail.

**2. Hstacking Text / NLP features with text feature vectors :** In the feature engineering section, we generated a number of different feature vectros, combining them together can help to improve the accuracy of the classifier.

**3. Hyperparamter Tuning in modelling :** Tuning the paramters is an important step, a number of parameters such as tree length, leafs, network paramters etc can be fine tuned to get a best fit model.

**4. Ensemble Models :** Stacking different models and blending their outputs can help to further improve the results. Read more about ensemble models [here](https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/)

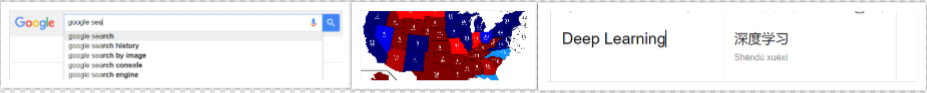
# An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec

[**NSS**](https://www.analyticsvidhya.com/blog/author/nss/)**, JUNE 4, 2017**

## Introduction

Before we start, have a look at the below examples.

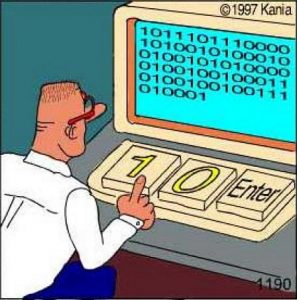
1. You open Google and search for a news article on the ongoing Champions trophy and get hundreds of search results in return about it.
2. Nate silver analysed millions of tweets and correctly predicted the results of 49 out of 50 states in 2008 U.S Presidential Elections.
3. You type a sentence in google translate in English and get an Equivalent Chinese conversion.



So what do the above examples have in common?

You possible guessed it right – **TEXT processing**. All the above three scenarios deal with humongous amount of text to perform different range of tasks like clustering in the google search example, classification in the second and Machine Translation in the third.

Humans can deal with text format quite intuitively but provided we have millions of documents being generated in a single day, we cannot have humans performing the above the three tasks. It is neither scalable nor effective.



So, how do we make computers of today perform clustering, classification etc on a text data since we know that they are generally inefficient at handling and processing strings or texts for any fruitful outputs?

Sure, a computer can match two strings and tell you whether they are same or not. But how do we make computers tell you about football or Ronaldo when you search for Messi? How do you make a computer understand that “Apple” in “Apple is a tasty fruit” is a fruit that can be eaten and not a company?

The answer to the above questions lie in creating a representation for words that capture their meanings, semantic relationships and the different types of contexts they are used in.

And all of these are implemented by using Word Embeddings or numerical representations of texts so that computers may handle them.

Below, we will see formally what are Word Embeddings and their different types and how we can actually implement them to perform the tasks like returning efficient Google search results.

## Table of Contents

1. What are Word Embeddings?
2. Different types of Word Embeddings  
   2.1  Frequency based Embedding  
   2.1.1 Count Vectors  
   2.1.2 TF-IDF  
   2.1.3 Co-Occurrence Matrix  
   2.2  Prediction based Embedding  
   2.2.1 CBOW  
   2.2.2 Skip-Gram
3. Word Embeddings use case scenarios(what all can be done using word embeddings? eg: similarity, odd one out etc.)
4. Using pre-trained Word Vectors
5. Training your own Word Vectors
6. End Notes

## 1. What are Word Embeddings?

In very simplistic terms, Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text. But before we dive into the details of Word Embeddings, the following question should be asked – Why do we need Word Embeddings?

As it turns out, many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression etc. in broad terms. And with the huge amount of data that is present in the text format, it is imperative to extract knowledge out of it and build applications. Some real world applications of text applications are – sentiment analysis of reviews by Amazon etc., document or news classification or clustering by Google etc.

Let us now define Word Embeddings formally. A Word Embedding format generally tries to map a word using a dictionary to a vector. Let us break this sentence down into finer details to have a clear view.

Take a look at this example – **sentence**=” Word Embeddings are Word converted into numbers ”

A word in this **sentence** may be “Embeddings” or “numbers ” etc.

A dictionary may be the list of all unique words in the **sentence.**So, a dictionary may look like – [‘Word’,’Embeddings’,’are’,’Converted’,’into’,’numbers’]

A vector representation of a word may be a one-hot encoded vector where 1 stands for the position where the word exists and 0 everywhere else. The vector representation of “numbers”in this format according to the above dictionary is [0,0,0,0,0,1] and of converted is[0,0,0,1,0,0].

This is just a very simple method to represent a word in the vector form. Let us look at different types of Word Embeddings or Word Vectors and their advantages and disadvantages over the rest.

## 2. Different types of Word Embeddings

The different types of word embeddings can be broadly classified into two categories-

1. Frequency based Embedding
2. Prediction based Embedding

Let us try to understand each of these methods in detail.

### 2.1 Frequency based Embedding

There are generally three types of vectors that we encounter under this category.

1. Count Vector
2. TF-IDF Vector
3. Co-Occurrence Vector

Let us look into each of these vectorization methods in detail.

#### 2.1.1 Count Vector

Consider a Corpus C of D documents {d1,d2…..dD} and N unique tokens extracted out of the corpus C. The N tokens will form our dictionary and the size of the Count Vector matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i).

Let us understand this using a simple example.

D1: He is a lazy boy. She is also lazy.

D2: Neeraj is a lazy person.

The dictionary created may be a list of unique tokens(words) in the corpus =[‘He’,’She’,’lazy’,’boy’,’Neeraj’,’person’]

Here, D=2, N=6

The count matrix M of size 2 X 6 will be represented as –

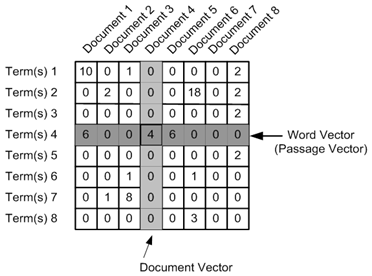
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | He | She | lazy | boy | Neeraj | person |
| D1 | 1 | 1 | 2 | 1 | 0 | 0 |
| D2 | 0 | 0 | 1 | 0 | 1 | 1 |

Now, a column can also be understood as word vector for the corresponding word in the matrix M. For example, the word vector for ‘lazy’ in the above matrix is [2,1] and so on.Here, the rows correspond to the documents in the corpus and the columns correspond to the tokens in the dictionary. The second row in the above matrix may be read as – D2 contains ‘lazy’: once, ‘Neeraj’: once and ‘person’ once.

Now there may be quite a few variations while preparing the above matrix M. The variations will be generally in-

1. The way dictionary is prepared.  
   Why? Because in real world applications we might have a corpus which contains millions of documents. And with millions of document, we can extract hundreds of millions of unique words. So basically, the matrix that will be prepared like above will be a very sparse one and inefficient for any computation. So an alternative to using every unique word as a dictionary element would be to pick say top 10,000 words based on frequency and then prepare a dictionary.
2. The way count is taken for each word.  
   We may either take the frequency (number of times a word has appeared in the document) or the presence(has the word appeared in the document?) to be the entry in the count matrix M. But generally, frequency method is preferred over the latter.

Below is a representational image of the matrix M for easy understanding.



#### 2.1.2 TF-IDF vectorization

This is another method which is based on the frequency method but it is different to the count vectorization in the sense that it takes into account not just the occurrence of a word in a single document but in the entire corpus. So, what is the rationale behind this? Let us try to understand.

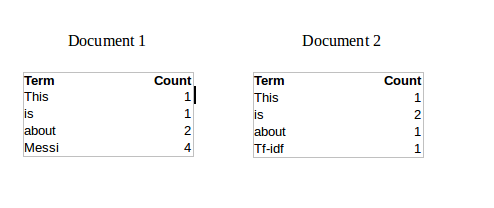
Common words like ‘is’, ‘the’, ‘a’ etc. tend to appear quite frequently in comparison to the words which are important to a document. For example, a document **A** on Lionel Messi is going to contain more occurences of the word “Messi” in comparison to other documents. But common words like “the” etc. are also going to be present in higher frequency in almost every document.

Ideally, what we would want is to down weight the common words occurring in almost all documents and give more importance to words that appear in a subset of documents.

TF-IDF works by penalising these common words by assigning them lower weights while giving importance to words like Messi in a particular document.

So, how exactly does TF-IDF work?

Consider the below sample table which gives the count of terms(tokens/words) in two documents.



Now, let us define a few terms related to TF-IDF.

TF = (Number of times term t appears in a document)/(Number of terms in the document)

So, TF(This,Document1) = 1/8

TF(This, Document2)=1/5

It denotes the contribution of the word to the document i.e words relevant to the document should be frequent. eg: A document about Messi should contain the word ‘Messi’ in large number.

IDF = log(N/n), where, N is the number of documents and n is the number of documents a term t has appeared in.

where N is the number of documents and n is the number of documents a term t has appeared in.

So, IDF(This) = log(2/2) = 0.

So, how do we explain the reasoning behind IDF? Ideally, if a word has appeared in all the document, then probably that word is not relevant to a particular document. But if it has appeared in a subset of documents then probably the word is of some relevance to the documents it is present in.

Let us compute IDF for the word ‘Messi’.

IDF(Messi) = log(2/1) = 0.301.

Now, let us compare the TF-IDF for a common word ‘This’ and a word ‘Messi’ which seems to be of relevance to Document 1.

TF-IDF(This,Document1) = (1/8) \* (0) = 0

TF-IDF(This, Document2) = (1/5) \* (0) = 0

TF-IDF(Messi, Document1) = (4/8)\*0.301 = 0.15

As, you can see for Document1 , TF-IDF method heavily penalises the word ‘This’ but assigns greater weight to ‘Messi’. So, this may be understood as ‘Messi’ is an important word for Document1 from the context of the entire corpus.

#### 2.1.3 Co-Occurrence Matrix with a fixed context window

**The big idea** – Similar words tend to occur together and will have similar context for example – Apple is a fruit. Mango is a fruit.  
Apple and mango tend to have a similar context i.e fruit.

Before I dive into the details of how a co-occurrence matrix is constructed, there are two concepts that need to be clarified – Co-Occurrence and Context Window.

Co-occurrence – For a given corpus, the co-occurrence of a pair of words say w1 and w2 is the number of times they have appeared together in a Context Window.

Context Window – Context window is specified by a number and the direction. So what does a context window of 2 (around) means? Let us see an example below,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quick | Brown | Fox | Jump | Over | The | Lazy | Dog |

The green words are a 2 (around) context window for the word ‘Fox’ and for calculating the co-occurrence only these words will be counted. Let us see context window for the word ‘Over’.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quick | Brown | Fox | Jump | Over | The | Lazy | Dog |

Now, let us take an example corpus to calculate a co-occurrence matrix.

Corpus = He is not lazy. He is intelligent. He is smart.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **He** | **is** | **not** | **lazy** | **intelligent** | **smart** |
| **He** | 0 | 4 | 2 | 1 | 2 | 1 |
| **is** | 4 | 0 | 1 | 2 | 2 | 1 |
| **not** | 2 | 1 | 0 | 1 | 0 | 0 |
| **lazy** | 1 | 2 | 1 | 0 | 0 | 0 |
| **intelligent** | 2 | 2 | 0 | 0 | 0 | 0 |
| **smart** | 1 | 1 | 0 | 0 | 0 | 0 |

Let us understand this co-occurrence matrix by seeing two examples in the table above. Red and the blue box.

Red box- It is the number of times ‘He’ and ‘is’ have appeared in the context window 2 and it can be seen that the count turns out to be 4. The below table will help you visualise the count.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| He | is | not | lazy | He | is | intelligent | He | is | smart |
|  |  |  |  |  |  |  |  |  |  |
| He | is | not | lazy | He | is | intelligent | He | is | smart |
|  |  |  |  |  |  |  |  |  |  |
| He | is | not | lazy | He | is | intelligent | He | is | smart |
|  |  |  |  |  |  |  |  |  |  |
| He | is | not | lazy | He | is | intelligent | He | is | smart |

while the word ‘lazy’ has never appeared with ‘intelligent’ in the context window and therefore has been assigned 0 in the blue box.

**Variations of Co-occurrence Matrix**

Let’s say there are V unique words in the corpus. So Vocabulary size = V. The columns of the Co-occurrence matrix form the context words. The different variations of Co-Occurrence Matrix are-

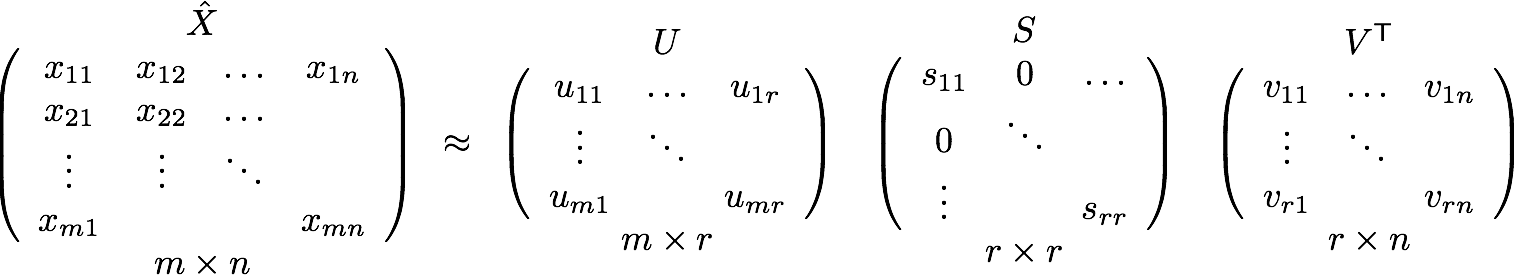
1. A co-occurrence matrix of size V X V. Now, for even a decent corpus V gets very large and difficult to handle. So generally, this architecture is never preferred in practice.
2. A co-occurrence matrix of size V X N where N is a subset of V and can be obtained by removing irrelevant words like stopwords etc. for example. This is still very large and presents computational difficulties.

But, remember this co-occurrence matrix is not the word vector representation that is generally used. Instead, this Co-occurrence matrix is decomposed using techniques like PCA, SVD etc. into factors and combination of these factors forms the word vector representation.

Let me illustrate this more clearly. For example, you perform PCA on the above matrix of size VXV. You will obtain V principal components. You can choose k components out of these V components. So, the new matrix will be of the form V X k.

And, a single word, instead of being represented in V dimensions will be represented in k dimensions while still capturing almost the same semantic meaning. k is generally of the order of hundreds.

So, what PCA does at the back is decompose Co-Occurrence matrix into three matrices, U,S and V where U and V are both orthogonal matrices. What is of importance is that dot product of U and S gives the word vector representation and V gives the word context representation.



**Advantages of Co-occurrence Matrix**

1. It preserves the semantic relationship between words. i.e man and woman tend to be closer than man and apple.
2. It uses SVD at its core, which produces more accurate word vector representations than existing methods.
3. It uses factorization which is a well-defined problem and can be efficiently solved.
4. It has to be computed once and can be used anytime once computed. In this sense, it is faster in comparison to others.

**Disadvantages of Co-Occurrence Matrix**

1. It requires huge memory to store the co-occurrence matrix.  
   But, this problem can be circumvented by factorizing the matrix out of the system for example in Hadoop clusters etc. and can be saved.

## 2.2 Prediction based Vector

**Pre-requisite**: This section assumes that you have a working knowledge of how a neural network works and the mechanisms by which weights in an NN are updated. If you are new to Neural Network, I would suggest you go through [this awesome article](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/) by Sunil to gain a very good understanding of how NN works.

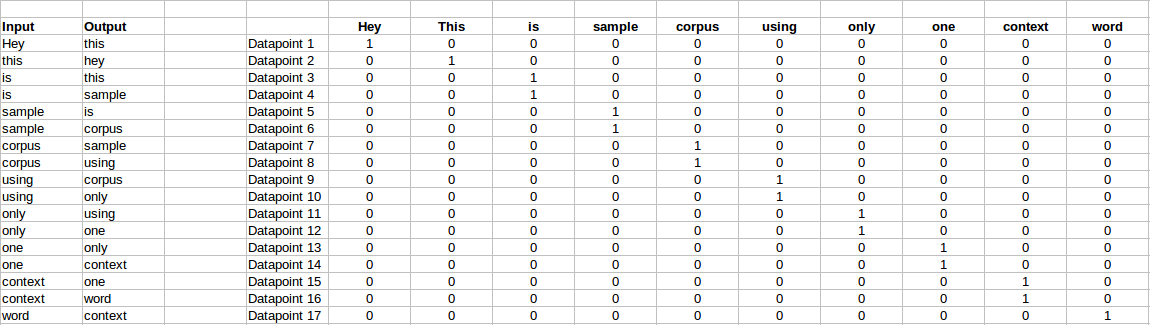
So far, we have seen deterministic methods to determine word vectors. But these methods proved to be limited in their word representations until Mitolov etc. el introduced word2vec to the NLP community. These methods were prediction based in the sense that they provided probabilities to the words and proved to be state of the art for tasks like word analogies and word similarities. They were also able to achieve tasks like King -man +woman = Queen, which was considered a result almost magical. So let us look at the word2vec model used as of today to generate word vectors.

Word2vec is not a single algorithm but a combination of two techniques – CBOW(Continuous bag of words) and Skip-gram model. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations. Let us discuss both these methods separately and gain intuition into their working.

### 2.2.1 CBOW (Continuous Bag of words)

The way CBOW work is that it tends to predict the probability of a word given a context. A context may be a single word or a group of words. But for simplicity, I will take a single context word and try to predict a single target word.

Suppose, we have a corpus C = “Hey, this is sample corpus using only one context word.” and we have defined a context window of 1. This corpus may be converted into a training set for a CBOW model as follow. The input is shown below. The matrix on the right in the below image contains the one-hot encoded from of the input on the left.

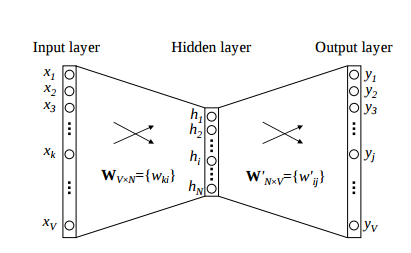


The target for a single datapoint say Datapoint 4 is shown as below

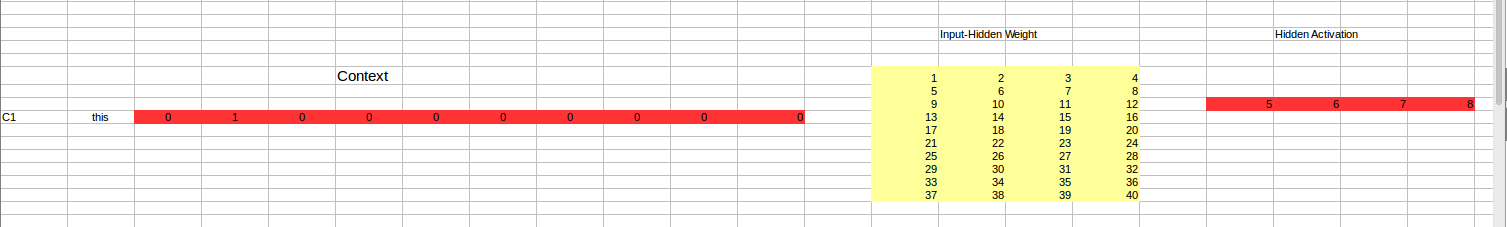
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hey | this | is | sample | corpus | using | only | one | context | word |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

This matrix shown in the above image is sent into a shallow neural network with three layers: an input layer, a hidden layer and an output layer. The output layer is a softmax layer which is used to sum the probabilities obtained in the output layer to 1. Now let us see how the forward propagation will work to calculate the hidden layer activation.

Let us first see a diagrammatic representation of the CBOW model.



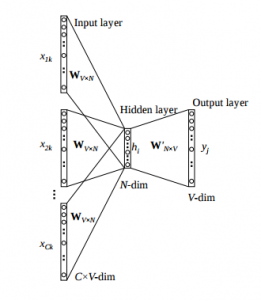
The matrix representation of the above image for a single data point is below.



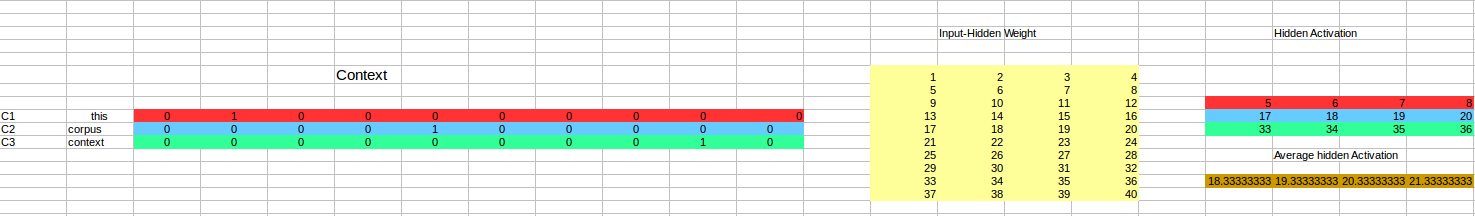
The flow is as follows:

1. The input layer and the target, both are one- hot encoded of size [1 X V]. Here V=10 in the above example.
2. There are two sets of weights. one is between the input and the hidden layer and second between hidden and output layer.  
   Input-Hidden layer matrix size =[V X N] , hidden-Output layer matrix  size =[N X V] : Where N is the number of dimensions we choose to represent our word in. It is arbitary and a hyper-parameter for a Neural Network. Also, N is the number of neurons in the hidden layer. Here, N=4.
3. There is a no activation function between any layers.( More specifically, I am referring to linear activation)
4. The input is multiplied by the input-hidden weights and called hidden activation. It is simply the corresponding row in the input-hidden matrix copied.
5. The hidden input gets multiplied by hidden- output weights and output is calculated.
6. Error between output and target is calculated and propagated back to re-adjust the weights.
7. The weight  between the hidden layer and the output layer is taken as the word vector representation of the word.

We saw the above steps for a single context word. Now, what about if we have multiple context words? The image below describes the architecture for multiple context words.



Below is a matrix representation of the above architecture for an easy understanding.



The image above takes 3 context words and predicts the probability of a target word. The input can be assumed as taking three one-hot encoded vectors in the input layer as shown above in red, blue and green.

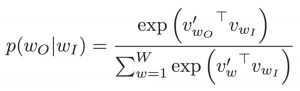
So, the input layer will have 3 [1 X V] Vectors in the input as shown above and 1 [1 X V] in the output layer. Rest of the architecture is same as for a 1-context CBOW.

The steps remain the same, only the calculation of hidden activation changes. Instead of just copying the corresponding rows of the input-hidden weight matrix to the hidden layer, an average is taken over all the corresponding rows of the matrix. We can understand this with the above figure. The average vector calculated becomes the hidden activation. So, if we have three context words for a single target word, we will have three initial hidden activations which are then averaged element-wise to obtain the final activation.

In both a single context word and multiple context word, I have shown the images till the calculation of the hidden activations since this is the part where CBOW differs from a simple MLP network. The steps after the calculation of hidden layer are same as that of the MLP as mentioned in this article – [Understanding and Coding Neural Networks from scratch](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/).

The differences between MLP and CBOW are  mentioned below for clarification:

1. The objective function in MLP is a MSE(mean square error) whereas in CBOW it is negative log likelihood of a word given a set of context i.e -log(p(wo/wi)), where p(wo/wi) is given as



wo : output word  
wi: context words

2. The gradient of error with respect to hidden-output weights and input-hidden weights are different since MLP has  sigmoid activations(generally) but CBOW has linear activations. The method however to calculate the gradient is same as an MLP.

**Advantages of CBOW:**

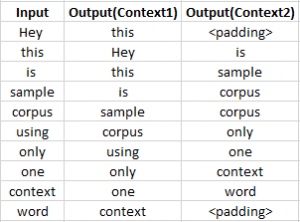
1. Being probabilistic is nature, it is supposed to perform superior to deterministic methods(generally).
2. It is low on memory. It does not need to have huge RAM requirements like that of co-occurrence matrix where it needs to store three huge matrices.

**Disadvantages of CBOW:**

1. CBOW takes the average of the context of a word (as seen above in calculation of hidden activation). For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies.
2. Training a CBOW from scratch can take forever if not properly optimized.

## 2.2.2 Skip – Gram model

Skip – gram follows the same topology as of CBOW. It just flips CBOW’s architecture on its head. The aim of skip-gram is to predict the context given a word. Let us take the same corpus that we built our CBOW model on. C=”Hey, this is sample corpus using only one context word.” Let us construct the training data.

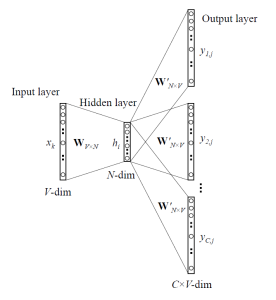


The input vector for skip-gram is going to be similar to a 1-context CBOW model. Also, the calculations up to hidden layer activations are going to be the same. The difference will be in the target variable. Since we have defined a context window of 1 on both the sides, there will be “**two” one hot encoded target variables** and “**two” corresponding outputs** as can be seen by the blue section in the image.

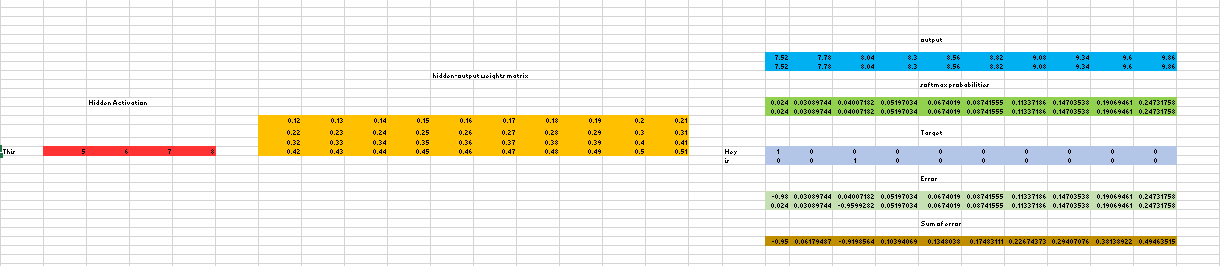
Two separate errors are calculated with respect to the two target variables and the two error vectors obtained are added element-wise to obtain a final error vector which is propagated back to update the weights.

The weights between the input and the hidden layer are taken as the word vector representation after training. The loss function or the objective is of the same type as of the CBOW model.

The skip-gram architecture is shown below.



For a better understanding, matrix style structure with calculation has been shown below.



Let us break down the above image.

Input layer  size – [1 X V], Input hidden weight matrix size – [V X N], Number of neurons in hidden layer – N, Hidden-Output weight matrix size – [N X V], Output layer size – C [1 X V]

In the above example, C is the number of context words=2, V= 10, N=4

1. The row in red is the hidden activation corresponding to the input one-hot encoded vector. It is basically the corresponding row of input-hidden matrix copied.
2. The yellow matrix is the weight between the hidden layer and the output layer.
3. The blue matrix is obtained by the matrix multiplication of hidden activation and the hidden output weights. There will be two rows calculated for two target(context) words.
4. Each row of the blue matrix is converted into its softmax probabilities individually as shown in the green box.
5. The grey matrix contains the one hot encoded vectors of the two context words(target).
6. Error is calculated by substracting the first row of the grey matrix(target) from the first row of the green matrix(output) element-wise. This is repeated for the next row. Therefore, for **n**target context words, we will have **n**error vectors.
7. Element-wise sum is taken over all the error vectors to obtain a final error vector.
8. This error vector is propagated back to update the weights.

### Advantages of Skip-Gram Model

1. Skip-gram model can capture two semantics for a single word. i.e it will have two vector representations of Apple. One for the company and other for the fruit.
2. Skip-gram with negative sub-sampling outperforms every other method generally.

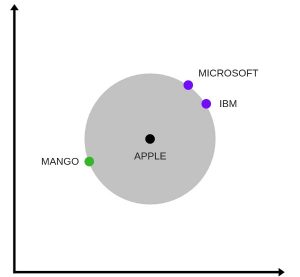
[This](http://bit.ly/wevi-online) is an excellent interactive tool to visualise CBOW and skip gram in action. I would suggest you to really go through this link for a better understanding.

## 3. Word Embeddings use case scenarios

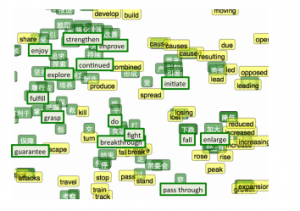
Since word embeddings or word Vectors are numerical representations of contextual similarities between words, they can be manipulated and made to perform amazing tasks like-

1. Finding the degree of similarity between two words.  
   model.similarity('woman','man')  
   0.73723527
2. Finding odd one out.  
   model.doesnt\_match('breakfast cereal dinner lunch';.split())  
   'cereal'
3. Amazing things like woman+king-man =queen  
   model.most\_similar(positive=['woman','king'],negative=['man'],topn=1)  
   queen: 0.508
4. Probability of a text under the model  
   model.score(['The fox jumped over the lazy dog'.split()])  
   0.21

Below is one interesting visualisation of word2vec.



The above image is a t-SNE representation of word vectors in 2 dimension and you can see that two contexts of apple have been captured. One is a fruit and the other company.

5.  It can be used to perform Machine Translation.  


The above graph is a bilingual embedding with chinese in green and english in yellow. If we know the words having similar meanings in chinese and english, the above bilingual embedding can be used to translate one language into the other.

## 4. Using pre-trained word vectors

We are going to use google’s pre-trained model. It contains word vectors for a vocabulary of 3 million words trained on around 100 billion words from the google news dataset. The downlaod link for the model is [this](https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit). Beware it is a 1.5 GB download.

from gensim.models import Word2Vec

#loading the downloaded model  
model = Word2Vec.load\_word2vec\_format('GoogleNews-vectors-negative300.bin', binary=True, norm\_only=True)

#the model is loaded. It can be used to perform all of the tasks mentioned above.

# getting word vectors of a word  
dog = model['dog']

#performing king queen magic  
print(model.most\_similar(positive=['woman', 'king'], negative=['man']))

#picking odd one out  
print(model.doesnt\_match("breakfast cereal dinner lunch".split()))

#printing similarity index  
print(model.similarity('woman', 'man'))

## 5. Training your own word vectors

We will be training our own word2vec on a custom corpus. For training the model we will be using gensim and the steps are illustrated as below.

word2Vec requires that a format of list of list for training where every document is contained in a list and every list contains list of tokens of that documents. I won’t be covering the pre-preprocessing part here. So let’s take an example list of list to train our word2vec model.

sentence=[[‘Neeraj’,’Boy’],[‘Sarwan’,’is’],[‘good’,’boy’]]

#training word2vec on 3 sentences  
model = gensim.models.Word2Vec(sentence, min\_count=1,size=300,workers=4)

Let us try to understand the parameters of this model.

sentence – list of list of our corpus  
min\_count=1 -the threshold value for the words. Word with frequency greater than this only are going to be included into the model.  
size=300 – the number of dimensions in which we wish to represent our word. This is the size of the word vector.  
workers=4 – used for parallelization

#using the model  
#The new trained model can be used similar to the pre-trained ones.

#printing similarity index  
print(model.similarity('woman', 'man'))