Contents

[**To pull up all functions in a module** 3](#_Toc511053285)

[**Natural language processing:** 3](#_Toc511053286)

[**Important terms:** 4](#_Toc511053287)

[**Tokenization** – text into token 4](#_Toc511053288)

[**Tokens** – words or entities present in the text 4](#_Toc511053289)

[**Text** **object** – a sentence or a phrase or a word or an article 4](#_Toc511053290)

[**Text Preprocessing:** 4](#_Toc511053291)

[**Noise removal** (removing stop words, urls, punctuation, mentions etc..) 4](#_Toc511053292)

[**Lexicon normalization** (Tokenization, lemmatization, stemming) 5](#_Toc511053293)

[Stemming:  This process strips the suffixes (“ing”, “ly”, “es”, “s” etc) from a word. 5](#_Toc511053294)

[Lemmatization: Organized & step by step procedure to obtain the root form of the word, 5](#_Toc511053295)

[**Example for lemmatization and stemming** 5](#_Toc511053296)

[**Object standardization** (Regular exp, lookup tables) 6](#_Toc511053297)

[**Encoding-decoding noise:** 6](#_Toc511053298)

[**Grammar checker:** 6](#_Toc511053299)

[**Spelling correction:** 6](#_Toc511053300)

[**Text to Features (Feature Engineering on text data)** 6](#_Toc511053301)

[**Syntactic Parsing** 7](#_Toc511053302)

[Dependency trees 7](#_Toc511053303)

[Part of speech tagging : 8](#_Toc511053304)

[**Word sense disambiguation:** Some language words have multiple meanings according to their usage. 9](#_Toc511053305)

[**Improving word-based features:** 9](#_Toc511053306)

[**Normalization and Lemmatization**: POS tags are the basis of lemmatization process for converting a word to its base form (lemma). 9](#_Toc511053307)

[**Efficient stopword removal** 9](#_Toc511053308)

[Entity Extraction (Entities as features) 9](#_Toc511053309)

[Named Entity Recognition (NER) 10](#_Toc511053310)

[**Noun phrase identification:** 10](#_Toc511053311)

[**Phrase classification:** 10](#_Toc511053312)

[**Entity disambiguation:** 11](#_Toc511053313)

[**Topic Modeling:** Topic modeling is a process of automatically identifying the topics present in a text corpus 11](#_Toc511053314)

[N-Grams as Features: A combination of N words together are called N-Grams. 12](#_Toc511053315)

[Statistical Features 13](#_Toc511053316)

[Term Frequency – Inverse Document Frequency (TF – IDF) 13](#_Toc511053317)

[Count / Density / Readability Features 14](#_Toc511053318)

[Word Embedding (text vectors) 14](#_Toc511053319)

[**Important tasks of NLP** 15](#_Toc511053320)

[**Text Classification(Classify the text object in one fixed category)** 15](#_Toc511053321)

[Text Matching / Similarity 17](#_Toc511053322)

[**Important applications of text matching includes** 18](#_Toc511053323)

[**Levenshtein Distance** 18](#_Toc511053324)

[**Phonetic Matching** 18](#_Toc511053325)

[Flexible String Matching 19](#_Toc511053326)

[Cosine Similarity 19](#_Toc511053327)

[Coreference Resolution 20](#_Toc511053328)

[Other NLP problems / tasks 20](#_Toc511053329)

[Important Libraries for NLP (python) 20](#_Toc511053330)

[Essentials of Machine Learning Algorithms 21](#_Toc511053331)

[**High level understanding about various machine learning algorithms along with Python codes to run them** 21](#_Toc511053332)

[3 types of Machine Learning Algorithms 21](#_Toc511053333)

[Supervised Learning 21](#_Toc511053334)

[Unsupervised Learning 22](#_Toc511053335)

[Reinforcement Learning: 22](#_Toc511053336)

[List of Common Machine Learning Algorithms 22](#_Toc511053337)

[Linear Regression 23](#_Toc511053338)

[Logistic Regression Or Logit regression 25](#_Toc511053339)

[Decision Tree 27](#_Toc511053340)

[SVM (Support Vector Machine) 28](#_Toc511053341)

[Naive Bayes 30](#_Toc511053342)

[kNN (k- Nearest Neighbors) 32](#_Toc511053343)

[K-Means 33](#_Toc511053344)

[**K-means Clustering from datascience site:** 35](#_Toc511053345)

[Algorithm 36](#_Toc511053346)

[Choosing *K* 37](#_Toc511053347)

[Example: Applying K-Means Clustering to Delivery Fleet Data 38](#_Toc511053348)

[Step 1: Clean and Transform Your Data 38](#_Toc511053349)

[Step 2: Choose K and Run the Algorithm 39](#_Toc511053350)

[Step 3: Review the Results 40](#_Toc511053351)

[Step 4: Iterate Over Several Values of K 40](#_Toc511053352)

[Additional Notes and Alternatives 41](#_Toc511053353)

[Feature Engineering 41](#_Toc511053354)

[Random Forest 42](#_Toc511053355)

[Dimensionality Reduction Algorithms 43](#_Toc511053356)

[Gradient Boosting Algorithms 44](#_Toc511053357)

[10.1. GBM 44](#_Toc511053358)

[10.2. XGBoost 44](#_Toc511053359)

[10.3. LightGBM 45](#_Toc511053360)

[10.4. Catboost 46](#_Toc511053361)

[End Notes 47](#_Toc511053362)

# **To pull up all functions in a module**

import nltk

for i in dir(nltk): print (i)

# **Natural language processing:**

21% of data is unstructured in the world

Data is being generated as we speak, as we tweet, as we send messages on Whatsapp and in various other activities. Majority of this data exists in the textual form, which is highly unstructured in nature. In order to produce significant and actionable insights from text data, it is important to get acquainted with the techniques and principles of Natural Language Processing (NLP).

**NLP is a branch of data science for systematic processing for**

- analyzing

- Understanding

- deriving information out of text data in smart & efficient manner

**Wide range of problem solving:**

- Automatic summarization

- Machine translation

- named entity recognition

- Relationship extraction

- Sentiment analysis

- Speech recognition

- Topic segmentation etc.

# **Important terms:**

## **Tokenization** – text into token

## **Tokens** – words or entities present in the text

## **Text** **object** – a sentence or a phrase or a word or an article

# **Text Preprocessing:**

This includes three steps for removing noised as it is unstructured

Here we go. Raw data goes through below three process to clean unwanted things.

## **Noise removal** (removing stop words, urls, punctuation, mentions etc..)

|  |
| --- |
| # Sample code to remove noisy words from a text  noise\_list = ["is", "a", "this", "..."]  def \_remove\_noise(input\_text):  words = input\_text.split()  noise\_free\_words = [word for word in words if word not in noise\_list]  noise\_free\_text = " ".join(noise\_free\_words)  return noise\_free\_text  \_remove\_noise("this is a sample text") |

|  |
| --- |
| ```  # Sample code to remove a regex pattern  import re  def \_remove\_regex(input\_text, regex\_pattern):  urls = re.finditer(regex\_pattern, input\_text)  for i in urls:  input\_text = re.sub(i.group().strip(), '', input\_text)  return input\_text  regex\_pattern = "#[\w]\*"  \_remove\_regex("remove this #hashtag from analytics vidhya", regex\_pattern)  >>> "remove this from analytics vidhya"  ``` |

## **Lexicon normalization** (Tokenization, lemmatization, stemming)

Another type of textual noise is about the multiple representations exhibited by single word.

For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma).

most common lexicon normalization practices

### Stemming:  This process strips the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.

Lemmatization: Organized & step by step procedure to obtain the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

### **Example for lemmatization and stemming**

|  |
| --- |
| ```  from nltk.stem.wordnet import WordNetLemmatizer  lem = WordNetLemmatizer()  from nltk.stem.porter import PorterStemmer  stem = PorterStemmer()  word = "multiplying"  lem.lemmatize(word, "v")  >> "multiply"  stem.stem(word)  >> "multipli"  ``` |

## **Object standardization** (Regular exp, lookup tables)

Text data often contains words or phrases which are not present in any standard lexical dictionaries

Some of the examples are – acronyms, hashtags with attached words, and colloquial slangs. With the help of regular expressions and manually prepared data dictionaries, this type of noise can be fixed, the code below uses a dictionary lookup method to replace social media slangs from a text.

Examle of Object standardization

|  |
| --- |
| ```  lookup\_dict = {'rt':'Retweet', 'dm':'direct message', "awsm" : "awesome", "luv" :"love", "..."}  def \_lookup\_words(input\_text):  words = input\_text.split()  new\_words = []  for word in words:  if word.lower() in lookup\_dict:  word = lookup\_dict[word.lower()]  new\_words.append(word) new\_text = " ".join(new\_words)  return new\_text  \_lookup\_words("RT this is a retweeted tweet by Shivam Bansal")  >> "Retweet this is a retweeted tweet by Shivam Bansal" |

https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/

## **Encoding-decoding noise:**

## **Grammar checker:**

## **Spelling correction:**

## **Text to Features (Feature Engineering on text data)**

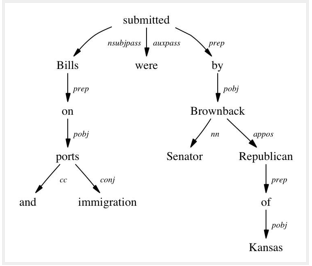
To analyse the preprocessed data. It needs be converted into features.

There are different techniques for this functionality

# **Syntactic Parsing**

## Dependency trees

* Sentences are composed of some words sewed together.
* The relationship among the words in a sentence is determined by the basic dependency grammar.
* Dependency grammar is a class of syntactic text analysis that deals with (labeled) asymmetrical binary relations between two lexical items (words).
* Every relation can be represented in the form of a triplet (relation, governor, dependent). For example: consider the sentence – “Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.” The relationship among the words can be observed in the form of a tree representation as shown:



* The tree shows that “submitted” is the root word of this sentence, and is linked by two sub-trees (subject and object subtrees).
* Each subtree is a itself a dependency tree with relations such as – (“Bills” <-> “ports” <by> “proposition” relation), (“ports” <-> “immigration” <by> “conjugation” relation).
* This type of tree, when parsed recursively in top-down manner gives grammar relation triplets as output which can be used as features for many nlp problems like entity wise sentiment analysis, actor & entity identification, and text classification.
* The python wrapper [StanfordCoreNLP](http://stanfordnlp.github.io/CoreNLP/) (by Stanford NLP Group, only commercial license) and NLTK dependency grammars can be used to generate dependency trees.

Part of speech tagging :

* Apart from the grammar relations, every word in a sentence is also associated with a part of speech (pos) tag (nouns, verbs, adjectives, adverbs etc) that defines the usage and function of a word in the sentence.
* H ere is a list of all possible pos-tags defined by Pennsylvania university. Following code using NLTK performs pos tagging annotation on input text. (it provides several implementations, the default one is perceptron tagger)

|  |
| --- |
| ```  from nltk import word\_tokenize, pos\_tag  text = "I am learning Natural Language Processing on Analytics Vidhya"  tokens = word\_tokenize(text)  print pos\_tag(tokens)  >>> [('I', 'PRP'), ('am', 'VBP'), ('learning', 'VBG'), ('Natural', 'NNP'),('Language', 'NNP'),  ('Processing', 'NNP'), ('on', 'IN'), ('Analytics', 'NNP'),('Vidhya', 'NNP')]  ``` |

### **Word sense disambiguation:** Some language words have multiple meanings according to their usage.

For example, in the two sentences below:

I. “Please book my flight for Delhi”

II. “I am going to read this book in the flight”

“Book” is used with different context, however the part of speech tag for both of the cases are different. In sentence I, the word “book” is used as **verb**, while in II it is used as **noun**. ([Lesk Algorithm](https://en.wikipedia.org/wiki/Lesk_algorithm) is also used for similar purposes)

### **Improving word-based features:**

A learning model could learn different contexts of a word when used word as the features, however if the part of speech tag is linked with them, the context is preserved, thus making strong features. For example:

**Sentence -“book my flight, I will read this book”**

**Tokens – (“book”, 2), (“my”, 1), (“flight”, 1), (“I”, 1), (“will”, 1), (“read”, 1), (“this”, 1)**

**Tokens with POS – (“book\_VB”, 1), (“my\_PRP$”, 1), (“flight\_NN”, 1), (“I\_PRP”, 1), (“will\_MD”, 1), (“read\_VB”, 1), (“this\_DT”, 1), (“book\_NN”, 1)**

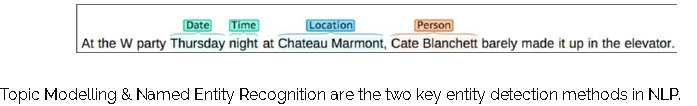
### **Normalization and Lemmatization**: POS tags are the basis of lemmatization process for converting a word to its base form (lemma).

**Efficient stopword removal:** POS tags are also useful in efficient removal of stopwords.

For example, there are some tags which always define the low frequency / less important words of a language. For example: (**IN** – “within”, “upon”, “except”), (**CD** – “one”,”two”, “hundred”), (**MD** – “may”, “must” etc)

## Entity Extraction (Entities as features)

* Entities are defined as the most important chunks of a sentence – noun phrases, verb phrases or both. Entity Detection algorithms are generally ensemble models of rule based parsing, dictionary lookups, pos tagging and dependency parsing.
* **The applicability of entity detection can be seen in the** 
  + automated chat bots
  + content analyzers
  + Consumer insights.



## Named Entity Recognition (NER)

The process of detecting the named entities such as person names, location names, company names etc from the text is called as NER. For example :

**Sentence – Sergey Brin, the manager of Google Inc. is walking in the streets of New York.**

**Named Entities –  ( “person” : “Sergey Brin” ), (“org” : “Google Inc.”), (“location” : “New York”)**

A typical NER model consists of three blocks:

**Noun phrase identification:** This step deals with extracting all the noun phrases from a text using dependency parsing and part of speech tagging.

**Phrase classification:**

* All the extracted noun phrases in the previous step are classified into respective categories (locations, names etc).
* Google Maps API provides a good path to disambiguate locations,
* Then, the open databases from dbpedia, wikipedia can be used to identify person names or company names.
* Apart from this, one can curate the lookup tables and dictionaries by combining information from different sources.

**Entity disambiguation:**

Sometimes it is possible that entities are misclassified, hence creating a validation layer on top of the results is useful. Use of knowledge graphs can be exploited for this purposes. The popular knowledge graphs are –

* Google Knowledge Graph
* IBM Watson
* Wikipedia.

## **Topic Modeling:** Topic modeling is a process of automatically identifying the topics present in a text corpus

* it derives the hidden patterns among the words in the corpus in an unsupervised manner.
* Topics are defined as “a repeating pattern of co-occurring terms in a corpus”.
* A good topic model results in – “health”, “doctor”, “patient”, “hospital” for a topic – Healthcare, and “farm”, “crops”, “wheat” for a topic – “Farming”.

Latent Dirichlet Allocation (LDA) is the most popular topic modelling technique, Following is the code to implement topic modeling using LDA in python. For a detailed explanation about its working and implementation, check the complete article [here.](https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/)

|  |
| --- |
| ```  doc1 = "Sugar is bad to consume. My sister likes to have sugar, but not my father."  doc2 = "My father spends a lot of time driving my sister around to dance practice."  doc3 = "Doctors suggest that driving may cause increased stress and blood pressure."  doc\_complete = [doc1, doc2, doc3]  doc\_clean = [doc.split() for doc in doc\_complete]  import gensim from gensim  import corpora  # Creating the term dictionary of our corpus, where every unique term is assigned an index.  dictionary = corpora.Dictionary(doc\_clean)  # Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.  doc\_term\_matrix = [dictionary.doc2bow(doc) for doc in doc\_clean]  # Creating the object for LDA model using gensim library  Lda = gensim.models.ldamodel.LdaModel  # Running and Training LDA model on the document term matrix  ldamodel = Lda(doc\_term\_matrix, num\_topics=3, id2word = dictionary, passes=50)  # Results  print(ldamodel.print\_topics())  ``` |

## N-Grams as Features: A combination of N words together are called N-Grams.

N grams (N > 1) are generally more informative as compared to words (Unigrams) as features. Also, bigrams (N = 2) are considered as the most important features of all the others. The following code generates bigram of a text.

|  |
| --- |
| ```  def generate\_ngrams(text, n):  words = text.split()  output = []  for i in range(len(words)-n+1):  output.append(words[i:i+n])  return output  >>> generate\_ngrams('this is a sample text', 2)  # [['this', 'is'], ['is', 'a'], ['a', 'sample'], , ['sample', 'text']]  ``` |

## Statistical Features

Text data can also be quantified directly into numbers using several techniques described in this section:

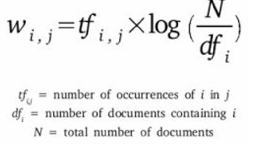
## Term Frequency – Inverse Document Frequency (TF – IDF)

* TF-IDF is a weighted model commonly used for information retrieval problems.
* It aims to convert the text documents into vector models on the basis of occurrence of words in the documents without taking considering the exact ordering.
* For Example – let say there is a dataset of N text documents, In any document “D”, TF and IDF will be defined as –

**Term Frequency (TF)** – TF for a term “t” is defined as the count of a term “t” in a document “D”

**Inverse Document Frequency (IDF**) – IDF for a term is defined as logarithm of ratio of total documents available in the corpus and number of documents containing the term T.

**TF . IDF –** TF IDF formula gives the relative importance of a term in a corpus (list of documents), given by the following formula below. Following is the code using python’s scikit learn package to convert a text into tf idf vectors:



|  |
| --- |
| ```  from sklearn.feature\_extraction.text import TfidfVectorizer  obj = TfidfVectorizer()  corpus = ['This is sample document.', 'another random document.', 'third sample document text']  X = obj.fit\_transform(corpus)  print X  >>>  (0, 1) 0.345205016865  (0, 4) ... 0.444514311537  (2, 1) 0.345205016865  (2, 4) 0.444514311537  ```  The model creates a vocabulary dictionary and assigns an index to each word. Each row in the output contains a tuple (i,j) and a tf-idf value of word at index j in document i. |

## Count / Density / Readability Features

* Count or Density based features can also be used in models and analysis.
* These features might seem trivial but shows a great impact in learning models.
* Some of the features are: Word Count, Sentence Count, Punctuation Counts and Industry specific word counts. Other types of measures include readability measures such as syllable counts, smog index and flesch reading ease.
* Refer to [Textstat](https://github.com/shivam5992/textstat) library to create such features.

## Word Embedding (text vectors)

* Word embedding is the modern way of representing words as vectors and aims to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus.
* They are widely used in deep learning models such as Convolutional Neural Networks and Recurrent Neural Networks.

[Word2Vec](https://code.google.com/archive/p/word2vec/) and [GloVe](http://nlp.stanford.edu/projects/glove/) are the two popular models to create word embedding of a text. These models takes a text corpus as input and produces the word vectors as output.

Word2Vec model is composed of preprocessing module, a shallow neural network model called **Continuous Bag of Words** and another shallow neural network model called **skip-gram**. These models are widely used for all other nlp problems. It first constructs a vocabulary from the training corpus and then learns word embedding representations. Following code using **gensim** package prepares the word embedding as the vectors.

|  |
| --- |
| ```  from gensim.models import Word2Vec  sentences = [['data', 'science'], ['vidhya', 'science', 'data', 'analytics'],['machine', 'learning'], ['deep', 'learning']]  # train the model on your corpus  model = Word2Vec(sentences, min\_count = 1)  print model.similarity('data', 'science')  >>> 0.11222489293  print model['learning']  >>> array([ 0.00459356  0.00303564 -0.00467622  0.00209638, ...])  ``` |

They can be used as feature vectors for ML model, used to measure text similarity using cosine similarity techniques, words clustering and text classification techniques.

## **Important tasks of NLP**

Various use cases in natural language processing

### **Text Classification(Classify the text object in one fixed category)**

**Text classification is one of the classical problem of NLP**

1. Email Spam Identification
2. topic classification of news
3. sentiment classification
4. organization of web pages by search engines.

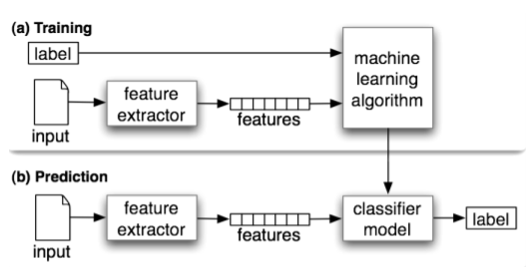
**Text classification, in common words is defined as a technique to systematically classify a text object (document or sentence) in one of the fixed category.** It is really helpful when the amount of data is too large, especially for

* organizing
* information filtering
* storage purposes.

Natural language classifier has two parts

1. Training
2. Prediction

A typical natural language classifier consists of two parts: (a) Training (b) Prediction as shown in image below. Firstly the text input is processes and features are created. The machine learning models then learn these features and is used for predicting against the new text.



|  |
| --- |
| #Here is a code that uses naive bayes classifier using text blob library (built on top of nltk).  ```  from textblob.classifiers import NaiveBayesClassifier as NBC  from textblob import TextBlob  training\_corpus = [  ('I am exhausted of this work.', 'Class\_B'),  ("I can't cooperate with this", 'Class\_B'),  ('He is my badest enemy!', 'Class\_B'),  ('My management is poor.', 'Class\_B'),  ('I love this burger.', 'Class\_A'),  ('This is an brilliant place!', 'Class\_A'),  ('I feel very good about these dates.', 'Class\_A'),  ('This is my best work.', 'Class\_A'),  ("What an awesome view", 'Class\_A'),  ('I do not like this dish', 'Class\_B')]  test\_corpus = [  ("I am not feeling well today.", 'Class\_B'),  ("I feel brilliant!", 'Class\_A'),  ('Gary is a friend of mine.', 'Class\_A'),  ("I can't believe I'm doing this.", 'Class\_B'),  ('The date was good.', 'Class\_A'), ('I do not enjoy my job', 'Class\_B')]  model = NBC(training\_corpus)  print(model.classify("Their codes are amazing."))  >>> "Class\_A"  print(model.classify("I don't like their computer."))  >>> "Class\_B"  print(model.accuracy(test\_corpus))  >>> 0.83  ``` |

Scikit.Learn also provides a pipeline framework for text classification:

|  |
| --- |
| ```  from sklearn.feature\_extraction.text  import TfidfVectorizer from sklearn.metrics  import classification\_report  from sklearn import svm  # preparing data for SVM model (using the same training\_corpus, test\_corpus from naive bayes example)  train\_data = []  train\_labels = []  for row in training\_corpus:  train\_data.append(row[0])  train\_labels.append(row[1])  test\_data = []  test\_labels = []  for row in test\_corpus:  test\_data.append(row[0])  test\_labels.append(row[1])  # Create feature vectors  vectorizer = TfidfVectorizer(min\_df=4, max\_df=0.9)  # Train the feature vectors  train\_vectors = vectorizer.fit\_transform(train\_data)  # Apply model on test data  test\_vectors = vectorizer.transform(test\_data)  # Perform classification with SVM, kernel=linear  model = svm.SVC(kernel='linear')  model.fit(train\_vectors, train\_labels)  prediction = model.predict(test\_vectors)  >>> ['Class\_A' 'Class\_A' 'Class\_B' 'Class\_B' 'Class\_A' 'Class\_A']  print (classification\_report(test\_labels, prediction)) |

The text classification model are heavily dependent upon the quality and quantity of features, while applying any machine learning model it is always a good practice to include more and more training data. H ere are some tips that I wrote about improving the text classification accuracy in one of my previous article.

## Text Matching / Similarity

One of the main areas of NLP to find the matching text object to find similarities.

### **Important applications of text matching includes**

* automatic spelling correction
* data de-duplication
* genome analysis etc.

### **Levenshtein Distance**

**Levenshtein Distance of two strings is defined as the minimum number of edit required to transform one string to another** with the allowable edit operations being **insertion, deletion, or substitution of a single character**

First, install the following:

|  |
| --- |
| pip install editdistance  import editdistance  editdistance.eval(list1, list2) |
| nltk.edit\_distance("aa bbbb cc", "aa b cc") |
| def levenshtein(s1,s2):  if len(s1) > len(s2):  s1,s2 = s2,s1  distances = range(len(s1) + 1)  for index2,char2 in enumerate(s2):  newDistances = [index2+1]  for index1,char1 in enumerate(s1):  if char1 == char2:  newDistances.append(distances[index1])  else:  newDistances.append(1 + min((distances[index1], distances[index1+1], newDistances[-1])))  distances = newDistances  return distances[-1]  print(levenshtein("analyze","analyse")) |

## **Phonetic Matching**

A Phonetic matching algorithm takes a keyword as input (person’s name, location name etc) and produces a character string that identifies a set of words that are (roughly) phonetically similar

Soundex and Metaphone are two main phonetic algorithms used for this purpose. Python’s module Fuzzy is used to compute soundex strings for different words, for example –

|  |
| --- |
| ```  import fuzzy  soundex = fuzzy.Soundex(4)  print soundex('ankit')  >>> “A523”  print soundex('aunkit')  >>> “A523”  ``` |

Flexible String Matching :

* A complete text matching system includes different algorithms pipelined together to compute variety of text variations.
* Regular expressions are really helpful for this purposes as well. Another common techniques include –
  + exact string matching,
  + lemmatized matching,
  + compact matching (takes care of spaces, punctuation’s, slangs etc).

Cosine Similarity

* When the text is represented as vector notation, a general cosine similarity can also be applied in order to measure vectorized similarity.
* Following code converts a text to vectors (using term frequency) and applies cosine similarity to provide closeness among two text.

|  |
| --- |
| import math  from collections import Counter  def get\_cosine(vec1, vec2):  common = set(vec1.keys()) & set(vec2.keys())  numerator = sum([vec1[x] \* vec2[x] for x in common])  sum1 = sum([vec1[x]\*\*2 for x in vec1.keys()])  sum2 = sum([vec2[x]\*\*2 for x in vec2.keys()])  denominator = math.sqrt(sum1) \* math.sqrt(sum2)    if not denominator:  return 0.0  else:  return float(numerator) / denominator  def text\_to\_vector(text):  words = text.split()  return Counter(words)  text1 = 'This is an article on analytics vidhya'  text2 = 'article on analytics vidhya is about natural language processing'  vector1 = text\_to\_vector(text1)  vector2 = text\_to\_vector(text2)  cosine = get\_cosine(vector1, vector2)  >>> 0.62  ``` |

## Coreference Resolution

Coreference Resolution is a process of finding relational links among the words (or phrases) within the sentences. Consider an example sentence: ” Donald went to John’s office to see the new table. He looked at it for an hour.“

Humans can quickly figure out that “he” denotes Donald (and not John), and that “it” denotes the table (and not John’s office). Coreference Resolution is the component of NLP that does this job automatically. It is used in document summarization, question answering, and information extraction. Stanford CoreNLP provides a python [wrapper](https://github.com/Wordseer/stanford-corenlp-python) for commercial purposes.

## Other NLP problems / tasks

* **Text Summarization** – Given a text article or paragraph, summarize it automatically to produce most important and relevant sentences in order.
* **Machine Translation** – Automatically translate text from one human language to another by taking care of grammar, semantics and information about the real world, etc.
* **Natural Language Generation and Understanding** – Convert information from computer databases or semantic intents into readable human language is called language generation. Converting chunks of text into more logical structures that are easier for computer programs to manipulate is called language understanding.
* **Optical Character Recognition** – Given an image representing printed text, determine the corresponding text.
* **Document to Information** – This involves parsing of textual data present in documents (websites, files, pdfs and images) to analyzable and clean format.

## Important Libraries for NLP (python)

* Scikit-learn: Machine learning in Python
* Natural Language Toolkit (NLTK): The complete toolkit for all NLP techniques.
* Pattern – A web mining module for the with tools for NLP and machine learning.
* TextBlob – Easy to use nl p tools API, built on top of NLTK and Pattern.
* spaCy – Industrial strength N LP with Python and Cython.
* Gensim – Topic Modelling for Humans
* Stanford Core NLP – NLP services and packages by Stanford NLP Group.

# Essentials of Machine Learning Algorithms

* We are probably living in the most defining period of human history.
* The period when computing moved from large mainframes to PCs to cloud.
* But what makes it defining is not what has happened, but what is coming our way in years to come.
* What makes this period exciting for someone like me is the democratization of the tools and techniques, which followed the boost in computing.
* Today, as a data scientist, I can build data crunching machines with complex algorithms for a few dollars per hour. But, reaching here wasn’t easy! I had my dark days and nights.

## **High level understanding about various machine learning algorithms along with Python codes to run them**

# 3 types of Machine Learning Algorithms

## Supervised Learning

**Definition 1:**

* The majority of practical machine learning uses supervised learning.
* Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.
* Y = f(X)
* The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Definition2 :

* **Supervised learning** is the [machine learning](https://en.wikipedia.org/wiki/Machine_learning) task of learning a function that maps an input to an output based on example input-output pairs

Definition 3:

* This algorithm consists of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables).
* Using these set of variables, we generate a function that map inputs to desired outputs.
* The training process continues until the model achieves a desired level of accuracy on the training data.

Examples of supervised learning:

* Regression,
* [Decision Tree](https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/)
* [Random Forest](https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/)
* KNN
* Logistic Regression etc.

### Unsupervised Learning

* In this algorithm, we do not have any target or outcome variable to predict / estimate.
* It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention.

**Examples of Unsupervised Learning:**

* Apriori algorithm
* K-means.

### Reinforcement Learning:

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

**Example of Reinforcement Learning:**

* Markov Decision Process

# **List of Common Machine Learning Algorithms**

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

# Linear Regression

It is used to estimate real values based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a \*X + b.

Criterial

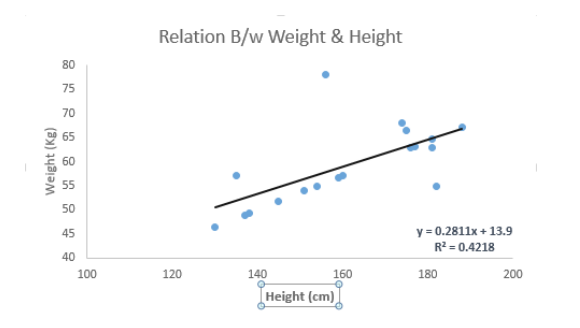
The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

* **Y – Dependent Variable(**criterion variable)
* **a – Slope**(The slope of a regression line (b) represents the rate of change in y as x changes)
* **X – Independent variable(predictor variable)**
* **b** **–** **Intercept**(The intercept (often labeled the constant) is the expected mean value of Y when all X=0.)

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation **y=0.2811x+13.9**. Now using this equation, we can find the weight, knowing the height of a person.



**Linear Regression is of mainly two types:**

* **Simple Linear Regression** – It is characterized by one independent variable.
* **Multiple Linear Regression** - It is characterized by multiple (more than 1) independent variables. While finding best fit line, you can fit a **polynomial** or **curvilinear** regression. And these are known as **polynomial** or **curvilinear** **regression**.

**Python Code**

|  |
| --- |
| #Import Library  #Import other necessary libraries like pandas, numpy...  from sklearn import linear\_model  #Load Train and Test datasets  #Identify feature and response variable(s) and values must be numeric and numpy arrays  x\_train=input\_variables\_values\_training\_datasets  y\_train=target\_variables\_values\_training\_datasets  x\_test=input\_variables\_values\_test\_datasets  # Create linear regression object  linear = linear\_model.LinearRegression()  # Train the model using the training sets and check score  linear.fit(x\_train, y\_train)  linear.score(x\_train, y\_train)  #Equation coefficient and Intercept  print('Coefficient: \n', linear.coef\_)  print('Intercept: \n', linear.intercept\_)  #Predict Output  predicted= linear.predict(x\_test) |

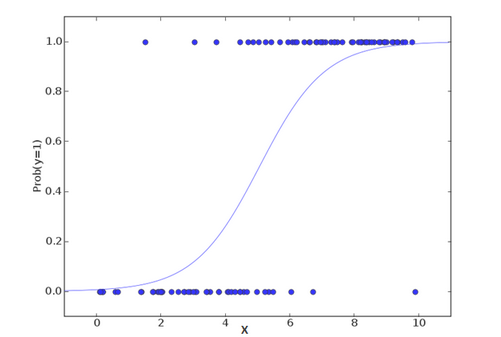
# Logistic Regression Or Logit regression

* Don’t get confused by its name! It is a classification not a regression algorithm.
* It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false ) based on given set of independent variable(s).
* In simple words, it predicts the probability of occurrence of an event by fitting data to a [logit function](https://en.wikipedia.org/wiki/Logistic_function). Hence, it is also known as **logit regression**.
* Since, it predicts the probability, its output values lies between 0 and 1 (as expected).
* Let’s say your friend gives you a puzzle to solve. There are only 2 outcome scenarios – either you solve it or you don’t.
* Now imagine, that you are being given wide range of puzzles / quizzes in an attempt to understand which subjects you are good at.
* The outcome to this study would be something like this – if you are given a trignometry based tenth grade problem, you are 70% likely to solve it.
* On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you.
* Coming to the math, the log odds of the outcome is modeled as a linear combination of the predictor variables.

|  |
| --- |
| odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence  ln(odds) = ln(p/(1-p))  logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk |

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

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



**Python Code**

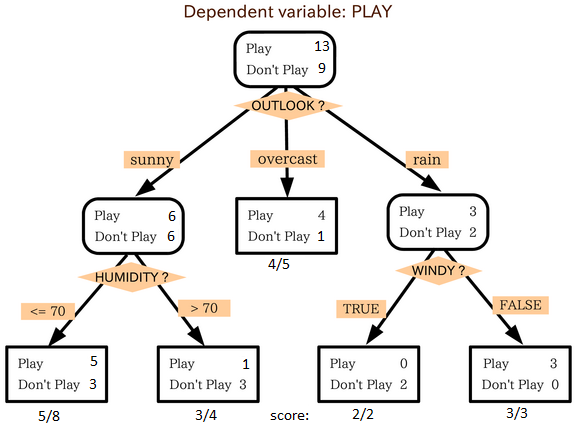
|  |
| --- |
| #Import Library  from sklearn.linear\_model import LogisticRegression  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create logistic regression object  model = LogisticRegression()  # Train the model using the training sets and check score  model.fit(X, y)  model.score(X, y)  #Equation coefficient and Intercept  print('Coefficient: \n', model.coef\_)  print('Intercept: \n', model.intercept\_)  #Predict Output  predicted= model.predict(x\_test) |

There are many different steps that could be tried in order to improve the model:

* including interaction terms
* removing features
* [regularization techniques](https://www.analyticsvidhya.com/blog/2015/02/avoid-over-fitting-regularization/)
* using a non-linear model

# Decision Tree

This is one of my favorite algorithm and I use it quite frequently. It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible. For more details, you can read: [Decision Tree Simplified](https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/).

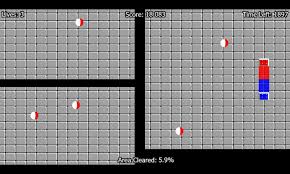
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/IkBzK.png)

source: [statsexchange](http://stats.stackexchange.com)

In the image above, you can see that population is classified into four different groups based on multiple attributes to identify ‘if they will play or not’. To split the population into different heterogeneous groups, it uses various techniques like

* Gini, Information Gain,
* Chi-square,
* entropy.

The best way to understand how decision tree works, is to play Jezzball – a classic game from Microsoft (image below). Essentially, you have a room with moving walls and you need to create walls such that maximum area gets cleared off with out the balls.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/download.jpg)

So, every time you split the room with a wall, you are trying to create 2 different populations with in the same room. Decision trees work in very similar fashion by dividing a population in as different groups as possible.

More: [Simplified Version of Decision Tree Algorithms](https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/)

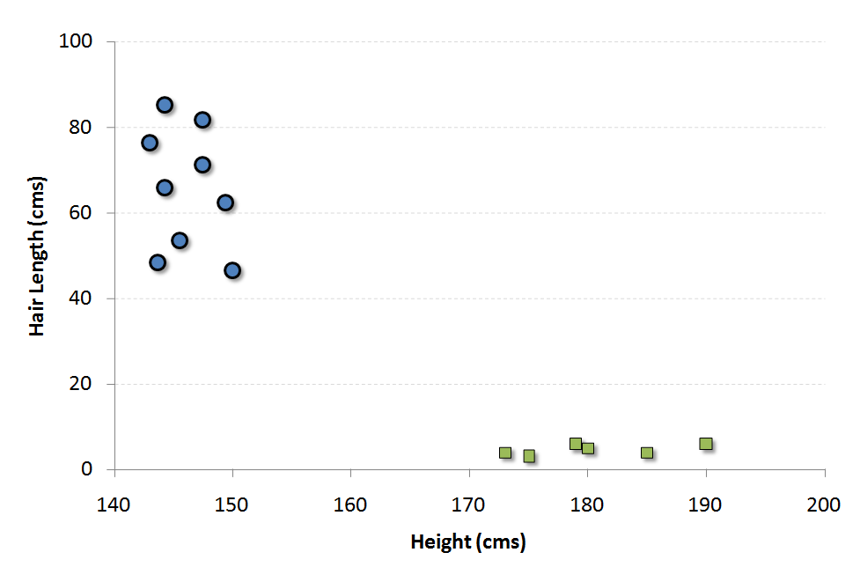
Python Code

|  |
| --- |
| #Import Library  #Import other necessary libraries like pandas, numpy...  from sklearn import tree  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create tree object  model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algorithm as gini or entropy (information gain) by default it is gini  # model = tree.DecisionTreeRegressor() for regression  # Train the model using the training sets and check score  model.fit(X, y)  model.score(X, y)  #Predict Output  predicted= model.predict(x\_test) |

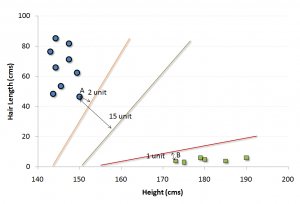
# SVM (Support Vector Machine)

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

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

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/SVM1.png)

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

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/SVM2.png)

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

More: [Simplified Version of Support Vector Machine](https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/)

**Think of this algorithm as playing JezzBall in n-dimensional space. The tweaks in the game are:**

* You can draw lines / planes at any angles (rather than just horizontal or vertical as in classic game)
* The objective of the game is to segregate balls of different colors in different rooms.
* And the balls are not moving.

#### Python Code

|  |
| --- |
| #Import Library  from sklearn import svm  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create SVM classification object  model = svm.svc() # there is various option associated with it, this is simple for classification. You can refer [link](http://scikit-learn.org/stable/modules/svm.html), for mo# re detail.  # Train the model using the training sets and check score  model.fit(X, y)  model.score(X, y)  #Predict Output  predicted= model.predict(x\_test) |

## Naive Bayes

It is a classification technique based on [Bayes’ theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:  
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Bayes_rule.png)

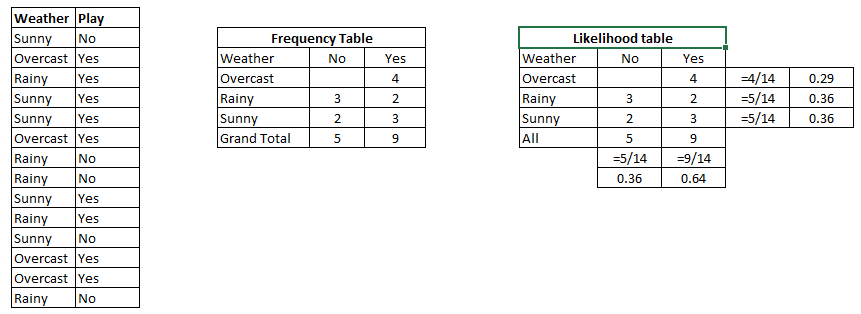
Here,

* *P*(*c|x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
* *P*(*c*) is the prior probability of *class*.
* *P*(*x|c*) is the likelihood which is the probability of *predictor* given *class*.
* *P*(*x*) is the prior probability of *predictor*.

**Example:**Let’s understand it using an example. Below I have a training data set of weather and corresponding target variable ‘Play’. Now, we need to classify whether players will play or not based on weather condition. Let’s follow the below steps to perform it.

Step 1: Convert the data set to frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Bayes_41.png)

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Problem:** Players will pay if weather is sunny, is this statement is correct?

We can solve it using above discussed method, so P(Yes | Sunny) = P( Sunny | Yes) \* P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P( Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 \* 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

#### Python Code:

|  |
| --- |
| #Import Library  from sklearn.naive\_bayes import GaussianNB  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create SVM classification object model = GaussianNB() # there is other distribution for multinomial classes like Bernoulli Naive Bayes, [Refer link](http://scikit-learn.org/stable/modules/naive_bayes.html" \t "_blank)  # Train the model using the training sets and check score  model.fit(X, y)  #Predict Output  predicted= model.predict(x\_test) |

# kNN (k- Nearest Neighbors)

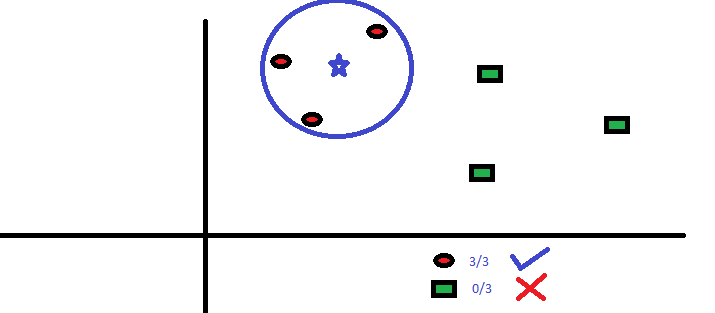
It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned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

**These distance functions can be**

* Euclidean,
* Manhattan,
* Minkowski
* Hamming distance.

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

More: Introduction to k-nearest neighbors : Simplified.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/KNN.png)

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

**Things to consider before selecting kNN:**

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

#### Python Code

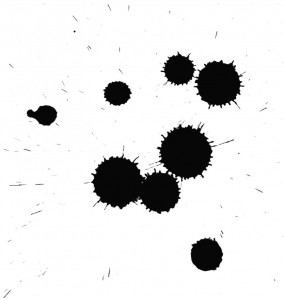
|  |
| --- |
| #Import Library  from sklearn.neighbors import KNeighborsClassifier  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create KNeighbors classifier object model  KNeighborsClassifier(n\_neighbors=6) # default value for n\_neighbors is 5  # Train the model using the training sets and check score  model.fit(X, y)  #Predict Output  predicted= model.predict(x\_test) |

## K-Means

It is a type of unsupervised algorithm which  solves the clustering problem. Its procedure follows a simple and easy  way to classify a given data set through a certain number of  clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Centroid is the features of the cluster

Remember figuring out shapes from ink blots? k means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/splatter_ink_blot_texture_by_maki_tak-d5p6zph.jpg)

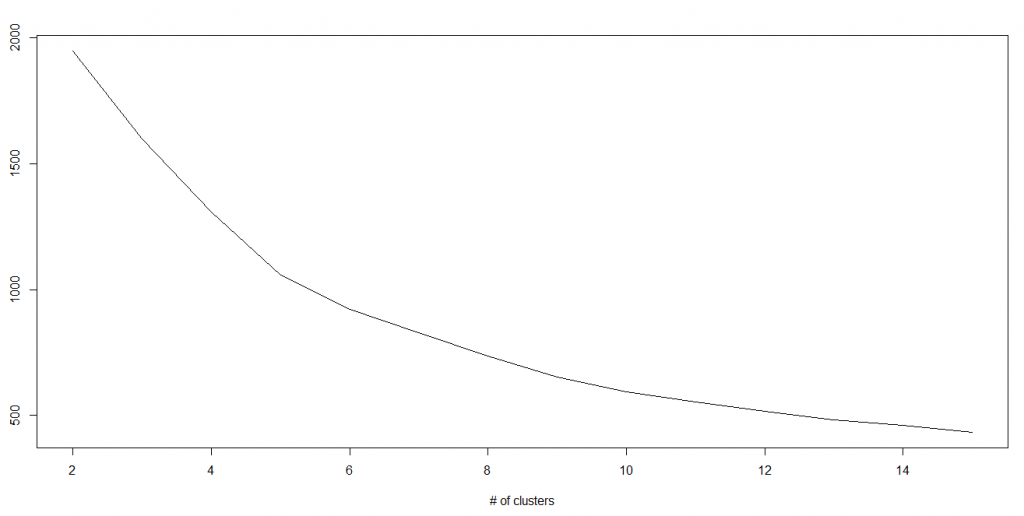
**How K-means forms cluster:**

1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.
3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

**How to determine value of K:**

In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Kmenas.png)

#### Python Code

|  |
| --- |
| #Import Library  from sklearn.cluster import KMeans  #Assumed you have, X (attributes) for training data set and x\_test(attributes) of test\_dataset  # Create KNeighbors classifier object model  k\_means = KMeans(n\_clusters=3, random\_state=0)  # Train the model using the training sets and check score  model.fit(X)  #Predict Output  predicted= model.predict(x\_test) |

# **K-means Clustering from datascience site:**

<https://www.datascience.com/blog/k-means-clustering>

**Prerequisites**

* Experience with the specific topic: Novice
* Professional experience: No industry experience

Knowledge of machine learning is not required, but the reader should be familiar with basic data analysis (e.g., descriptive analysis) and the programming language Python. To follow along, download the sample dataset [here](https://raw.githubusercontent.com/datascienceinc/learn-data-science/master/Introduction-to-K-means-Clustering/Data/data_1024.csv).

**Introduction to K-means Clustering**

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

1. The centroids of the K clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K" section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the K-means clustering algorithm covers:

* Common business cases where K-means is used
* The steps involved in running the algorithm
* A Python example using delivery fleet data

### 

**Business Uses**

The K-means clustering algorithm is used to find groups which have not been explicitly labeled in the data. This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the correct group.

This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:

* Behavioral segmentation:
  + Segment by purchase history
  + Segment by activities on application, website, or platform
  + Define personas based on interests
  + Create profiles based on activity monitoring
* Inventory categorization:
  + Group inventory by sales activity
  + Group inventory by manufacturing metrics
* Sorting sensor measurements:
  + Detect activity types in motion sensors
  + Group images
  + Separate audio
  + Identify groups in health monitoring
* Detecting bots or anomalies:
  + Separate valid activity groups from bots
  + Group valid activity to clean up outlier detection

In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

### **Algorithm**

The Κ-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters Κ and the data set. The data set is a collection of features for each data point. The algorithms starts with initial estimates for the Κ centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assigment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if ci is the collection of centroids in set C, then each data point x is assigned to a cluster based on

where dist( · ) is the standard (L2) Euclidean distance. Let the set of data point assignments for each ith cluster centroid be Si.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

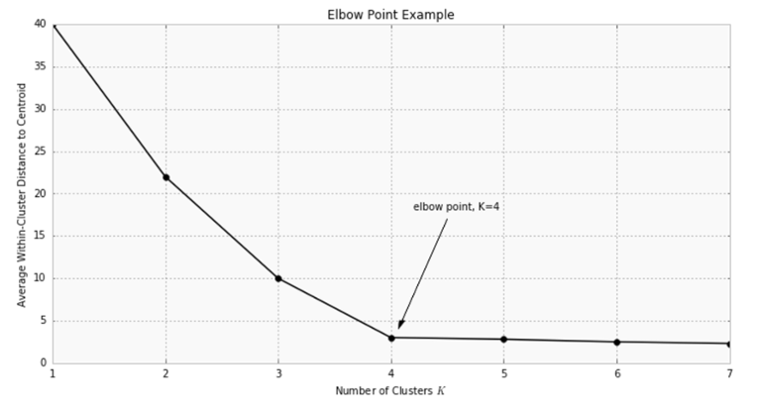
This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

### **Choosing** K

The algorithm described above finds the clusters and data set labels for a particular pre-chosen K. To find the number of clusters in the data, the user needs to run the K-means clustering algorithm for a range of K values and compare the results. In general, there is no method for determining exact value of K, but an accurate estimate can be obtained using the following techniques.

One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K.

A number of other techniques exist for validating K, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm. In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each K.

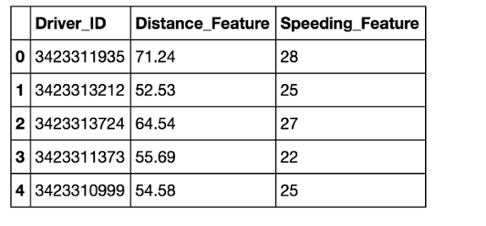


### Example: Applying K-Means Clustering to Delivery Fleet Data

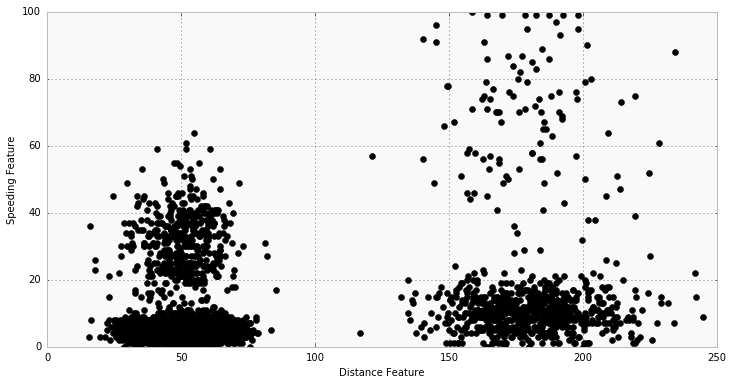
As an example, we'll show how the K-means algorithm works with a [sample dataset of delivery fleet driver data](https://raw.githubusercontent.com/datascienceinc/learn-data-science/master/Introduction-to-K-means-Clustering/Data/data_1024.csv). For the sake of simplicity, we'll only be looking at two driver features: mean distance driven per day and the mean percentage of time a driver was >5 mph over the speed limit. In general, this algorithm can be used for any number of features, so long as the number of data samples is much greater than the number of features.

### **Step 1: Clean and Transform Your Data**

For this example, we've already cleaned and completed some simple data transformations. A sample of the data as a pandas DataFrame is shown below.



The chart below shows the dataset for 4,000 drivers, with the distance feature on the x-axis and speeding feature on the y-axis.



### **Step 2: Choose K and Run the Algorithm**

Start by choosing K=2. For this example, use the Python packages [scikit-learn](http://scikit-learn.org/stable/) and [NumPy](http://www.numpy.org/) for computations as shown below:

import numpy as np

from sklearn.cluster import KMeans

### For the purposes of this example, we store feature data from our

### dataframe `df`, in the `f1` and `f2` arrays. We combine this into

### a feature matrix `X` before entering it into the algorithm.

f1 = df['Distance\_Feature'].values

f2 = df['Speeding\_Feature'].values

X=np.matrix(zip(f1,f2))

kmeans = KMeans(n\_clusters=2).fit(X)

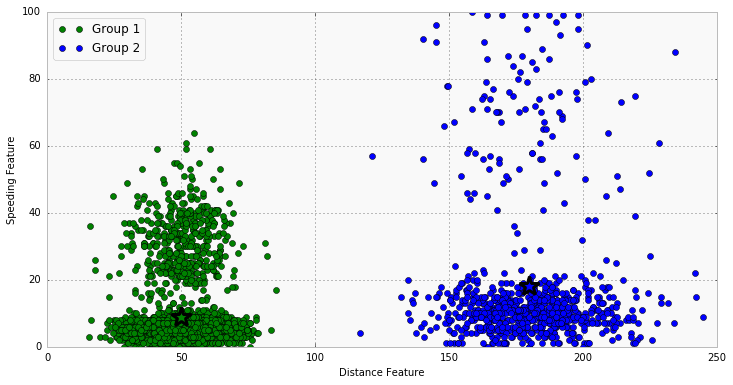
The cluster labels are returned in kmeans.labels\_.

### **Step 3: Review the Results**

The chart below shows the results. Visually, you can see that the K-means algorithm splits the two groups based on the distance feature. Each cluster centroid is marked with a star.

* Group 1 Centroid = (50, 5.2)
* Group 2 Centroid = (180.3, 10.5)

Using domain knowledge of the dataset, we can infer that Group 1 is urban drivers and Group 2 is rural drivers.

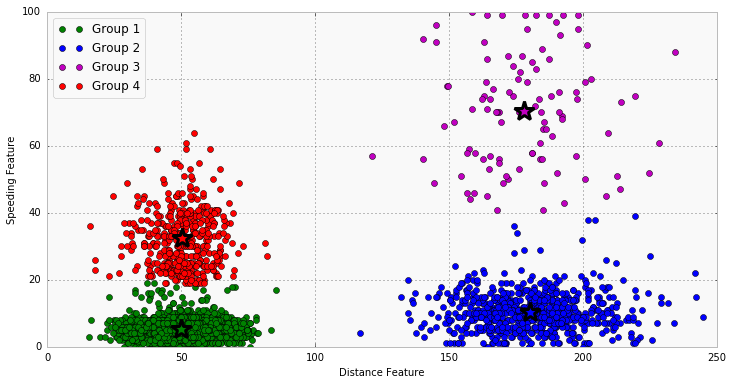


### **Step 4: Iterate Over Several Values of K**

Test how the results look for K=4. To do this, all you need to change is the target number of clusters in the KMeans() function.

kmeans = KMeans(n\_clusters=4).fit(X)

The chart below shows the resulting clusters. We see that four distinct groups have been identified by the algorithm; now speeding drivers have been separated from those who follow speed limits, in addition to the rural vs. urban divide. The threshold for speeding is lower with the urban driver group than for the rural drivers, likely due to urban drivers spending more time in intersections and stop-and-go traffic.



### **Additional Notes and Alternatives**

### **Feature Engineering**

Feature engineering is the process of using domain knowledge to choose which data metrics to input as features into a machine learning algorithm. Feature engineering plays a key role in K-means clustering; using meaningful features that capture the variability of the data is essential for the algorithm to find all of the naturally-occurring groups.

Categorical data (i.e., category labels such as gender, country, browser type) needs to be encoded or separated in a way that can still work with the algorithm.

Feature transformations, particularly to represent rates rather than measurements, can help to normalize the data. For example, in the delivery fleet example above, if total distance driven had been used rather than mean distance per day, then drivers would have been grouped by how long they had been driving for the company rather than rural vs. urban.

**Alternatives**

A number of [alternative clustering algorithms](https://www.datascience.com/blog/k-means-alternatives) exist including DBScan, spectral clustering, and modeling with Gaussian mixtures. A dimensionality reduction technique, such as principal component analysis, can be used to separate groups of patterns in data. You can read more about alternatives to K-means [in this post](https://www.datascience.com/blog/k-means-alternatives).

One possible outcome is that there are no organic clusters in the data; instead, all of the data fall along the continuous feature ranges within one single group. In this case, you may need to revisit the data features to see if different measurements need to be included or a feature transformation would better represent the variability in the data. In addition, you may want to impose categories or labels based on domain knowledge and modify your analysis approach.

For more information on K-means clustering, visit the [scikit learn site](http://scikit-learn.org/stable/modules/clustering.html#k-means).

# Random Forest

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

Each tree is planted & grown as follows:

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

For more details on this algorithm, comparing with decision tree and tuning model parameters, I would suggest you to read these articles:

1. [Introduction to Random forest – Simplified](https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/)
2. [Comparing a CART model to Random Forest (Part 1)](https://www.analyticsvidhya.com/blog/2014/06/comparing-cart-random-forest-1/)
3. [Comparing a Random Forest to a CART model (Part 2)](https://www.analyticsvidhya.com/blog/2014/06/comparing-random-forest-simple-cart-model/)
4. [Tuning the parameters of your Random Forest model](https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/)

**Python**

|  |
| --- |
| #Import Library  from sklearn.ensemble import RandomForestClassifier  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create Random Forest object  model= RandomForestClassifier()  # Train the model using the training sets and check score  model.fit(X, y)  #Predict Output  predicted= model.predict(x\_test) |

# Dimensionality Reduction Algorithms

In the last 4-5 years, there has been an exponential increase in data capturing at every possible stages. Corporates/ Government Agencies/ Research organisations are not only coming with new sources but also they are capturing data in great detail.

For example: E-commerce companies are capturing more details about customer like their demographics, web crawling history, what they like or dislike, purchase history, feedback and many others to give them personalized attention more than your nearest grocery shopkeeper.

As a data scientist, the data we are offered also consist of many features, this sounds good for building good robust model but there is a challenge. How’d you identify highly significant variable(s) out 1000 or 2000? In such cases, dimensionality reduction algorithm helps us along with various other algorithms like Decision Tree, Random Forest, PCA, Factor Analysis, Identify based on correlation matrix, missing value ratio and others.

To know more about this algorithms, you can read “[Beginners Guide To Learn Dimension Reduction Techniques](https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/)“.

#### Python  Code

|  |
| --- |
| #Import Library  from sklearn import decomposition  #Assumed you have training and test data set as train and test  # Create PCA obeject pca= decomposition.PCA(n\_components=k) #default value of k =min(n\_sample, n\_features)  # For Factor analysis  #fa= decomposition.FactorAnalysis()  # Reduced the dimension of training dataset using PCA  train\_reduced = pca.fit\_transform(train)  #Reduced the dimension of test dataset  test\_reduced = pca.transform(test)  #For more detail on this, please refer  [this link](http://scikit-learn.org/stable/modules/decomposition.html" \l "decompositions" \t "_blank). |

# Gradient Boosting Algorithms

## 10.1. GBM

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

More: [Know about Boosting algorithms in detail](https://www.analyticsvidhya.com/blog/2015/05/boosting-algorithms-simplified/)

#### Python Code

|  |
| --- |
| #Import Library  from sklearn.ensemble import GradientBoostingClassifier  #Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset  # Create Gradient Boosting Classifier object  model= GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1, random\_state=0)  # Train the model using the training sets and check score  model.fit(X, y)  #Predict Output  predicted= model.predict(x\_test) |

GradientBoostingClassifier and Random Forest are two different boosting tree classifier and often people ask about the [difference between these two algorithms](http://discuss.analyticsvidhya.com/t/what-is-the-fundamental-difference-between-randomforest-and-gradient-boosting-algorithms/2341).

### 10.2. XGBoost

Another classic gradient boosting algorithm that’s known to be the decisive choice between winning and losing in some Kaggle competitions.

The XGBoost has an immensely high predictive power which makes it the best choice for accuracy in events as it possesses both linear model and the tree learning algorithm, making the algorithm almost 10x faster than existing gradient booster techniques.

The support includes various objective functions, including regression, classification and ranking.

One of the most interesting things about the XGBoost is that it is also called a regularized boosting technique. This helps to reduce overfit modelling and has a massive support for a range of languages such as Scala, Java, R, Python, Julia and C++.

Supports distributed and widespread training on many machines that encompass GCE, AWS, Azure and Yarn clusters. XGBoost can also be integrated with Spark, Flink and other cloud dataflow systems with a built in cross validation at each iteration of the boosting process.

To learn more about XGBoost and parameter tuning, visit <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>.

Python Code:

|  |
| --- |
| from xgboost import XGBClassifier  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score  X = dataset[:,0:10]  Y = dataset[:,10:]  seed = 1  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=seed)  model = XGBClassifier()  model.fit(X\_train, y\_train)  #Make predictions for test data  y\_pred = model.predict(X\_test) |

### 10.3. LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

* Faster training speed and higher efficiency
* Lower memory usage
* Better accuracy
* Parallel and GPU learning supported
* Capable of handling large-scale data

The framework is a fast and high-performance gradient boosting one based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. It was developed under the Distributed Machine Learning Toolkit Project of Microsoft.

Since the LightGBM is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

Also, it is surprisingly very fast, hence the word ‘Light’.

Refer to the article to know more about LightGBM: <https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/>

Python Code:

|  |
| --- |
| data = np.random.rand(500, 10) # 500 entities, each contains 10 features  label = np.random.randint(2, size=500) # binary target  train\_data = lgb.Dataset(data, label=label)  test\_data = train\_data.create\_valid('test.svm')  param = {'num\_leaves':31, 'num\_trees':100, 'objective':'binary'}  param['metric'] = 'auc'  num\_round = 10  bst = lgb.train(param, train\_data, num\_round, valid\_sets=[test\_data])  bst.save\_model('model.txt')  # 7 entities, each contains 10 features  data = np.random.rand(7, 10)  ypred = bst.predict(data) |

### 10.4. Catboost

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google’s TensorFlow and Apple’s Core ML.

The best part about CatBoost is that it does not require extensive data training like other ML models, and can work on a variety of data formats; not undermining how robust it can be.

Make sure you handle missing data well before you proceed with the implementation.

Catboost can automatically deal with categorical variables without showing the type conversion error, which helps you to focus on tuning your model better rather than sorting out trivial errors.

Learn more about Catboost from this article: <https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/>

Python Code:

|  |
| --- |
| import pandas as pd  import numpy as np  from catboost import CatBoostRegressor  #Read training and testing files  train = pd.read\_csv("train.csv")  test = pd.read\_csv("test.csv")  #Imputing missing values for both train and test  train.fillna(-999, inplace=True)  test.fillna(-999,inplace=True)  #Creating a training set for modeling and validation set to check model performance  X = train.drop(['Item\_Outlet\_Sales'], axis=1)  y = train.Item\_Outlet\_Sales  from sklearn.model\_selection import train\_test\_split  X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, train\_size=0.7, random\_state=1234)  categorical\_features\_indices = np.where(X.dtypes != np.float)[0]  #importing library and building model  from catboost import CatBoostRegressormodel=CatBoostRegressor(iterations=50, depth=3, learning\_rate=0.1, loss\_function='RMSE')  model.fit(X\_train, y\_train,cat\_features=categorical\_features\_indices,eval\_set=(X\_validation, y\_validation),plot=True)  submission = pd.DataFrame()  submission['Item\_Identifier'] = test['Item\_Identifier']  submission['Outlet\_Identifier'] = test['Outlet\_Identifier']  submission['Item\_Outlet\_Sales'] = model.predict(test) |

### End Notes

By now, I am sure, you would have an idea of commonly used machine learning algorithms. My sole intention behind writing this article and providing the codes in R and Python is to get you started right away. If you are keen to master machine learning, start right away. Take up problems, develop a physical understanding of the process, apply these codes and see the fun!