Quora Question Pairs

## Problem Statement

Identify which questions on Quora are duplicates of questions that have already been asked.

**Approach**

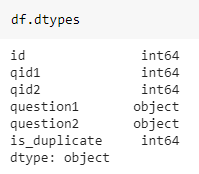
## Exploratory Data Analysis (EDA)

Exploratory data analysis means to understand each attribute and find the most important characters in the dataset.

The dataset shape:



The data type:



This step will include :

**Checking the missing values**

there is three Null values in the dataset

two values in question 2 and one value in question 1

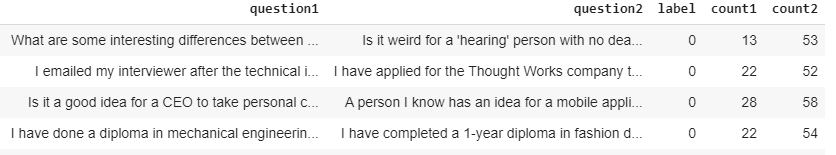
**Renaming the ‘is-duplicate’** attribute to ‘label’

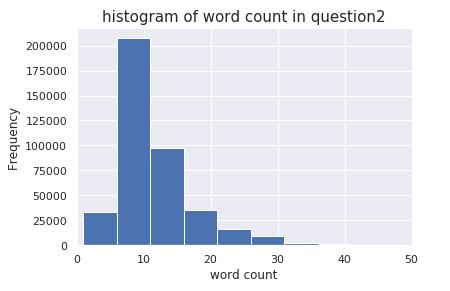
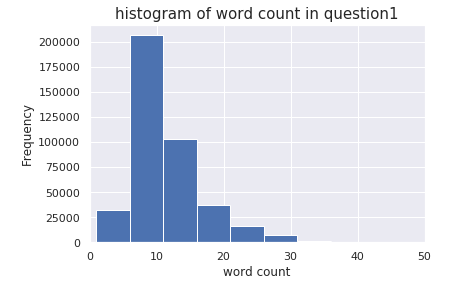
**Counting the words in each questions**

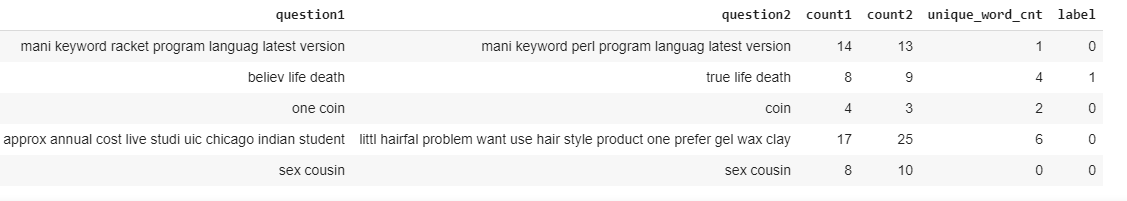
**Checking the distribution** **of the word count**

*count1* for question1 and *count2* for question2

both question 1 and question 2 has right (or positive) skewed distribution,

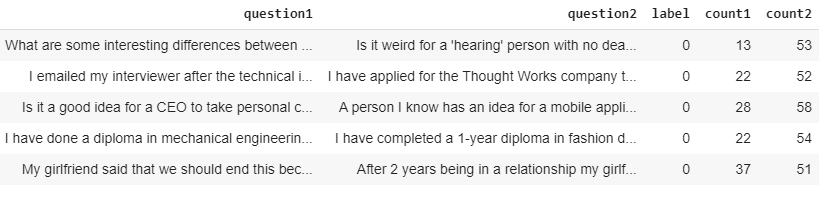






**Checking questions that have more than 50 word count**:

In question1 there is 0.18% of the questions more than 50 words and in question2 there is 0.04%. The ratio of the long questions are very small.



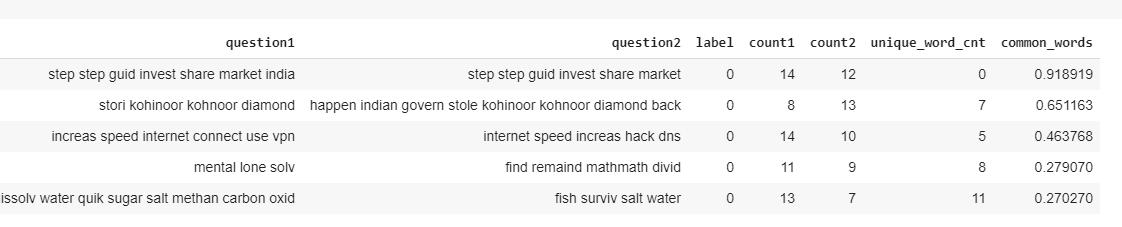
**Count the number of unique words in the pair of questions**

by unique words we mean the number of non-duplicate words in question1 and question2.

I obtained this by subtracting the intersect between question1 and question2 from the union between question 1 and question 2 then count the words that is left.

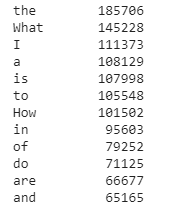
**Checking the ratio of common words between two questions**

I obtained this by set question 1 and finding the intersect with question 2 then count the number of words.



**Checking the words with high frequency**

in question1 and question2



Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Word cloud by default remove stop words.

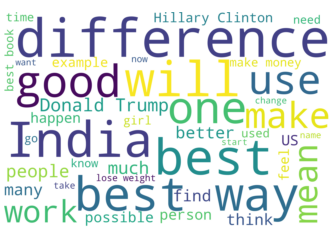


Figure word cloud for question 1

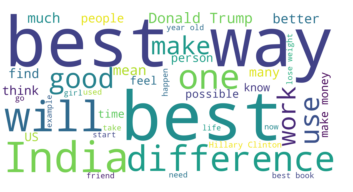


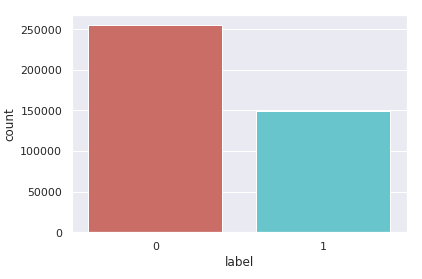
Figure 2 Word cloud for question2

**Deciding on The dependent variable** in our dataset is ‘label’

**Checking for imbalance in the dependent variable ‘label’**

Imbalanced data is a classification problem when the number of observations in each class is not equal.

The dataset has 63% non-duplicate questions with label (0) and 37% duplicate questions with label (1)



## Text Preprocessing

Text preprocessing is to convert the text into a form that computers can understand and be used with machine learning algorithms.

The techniques that I used with the text data:

**Convert to lower case:**

will prevent having multiple copies of the same and helps with having consistent data.

**Remove the top 50 common words**

as most of these words are stop words and will not affect the meaning of the questions.

**Replace shortcuts**

‘ve 🡪 have

n’t 🡪 not

‘s 🡪 is

‘m 🡪 am

‘re 🡪 are

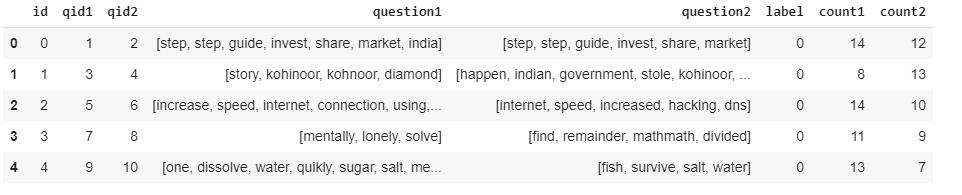
‘d 🡪 would

‘ll 🡪 will

**Remove puctuations** or any other symbols in order to not consider puctuations as words like question marks, comas, dots...etc.

**Remove stop words**

After tokenizing the questions. Stop words are commonly used words (such as like, there, the) and we do not want these words to take a large space in the memory or slow down the processing time. For this step I used  NLTK(Natural Language Toolkit) in python which has a list of predefined stop words with a length of 174 words.



**Remove Numbers** to keep all the dataset as string text.

**Remove invalid questions**

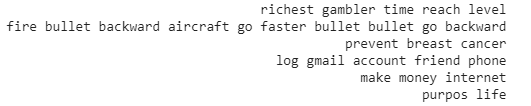
by removing questions that do not have any vowels there is *0.04%* invalid questions in the dataset which is a very small fraction of the dataset I decided to remove them.

**Drop questions with more than 50 words**:

I decided to remove these questions because they are too long and will affect the processing time.

**Text normalization**

is the process of transforming the text into a standard form. For this step I used text stemming which convert all the words to their word stem by removing the words suffices and converting all the words to their roots example of stemming is student, studying and study will be all converted to stud. I used snowball stemmer in python.



### **TF-IDF**

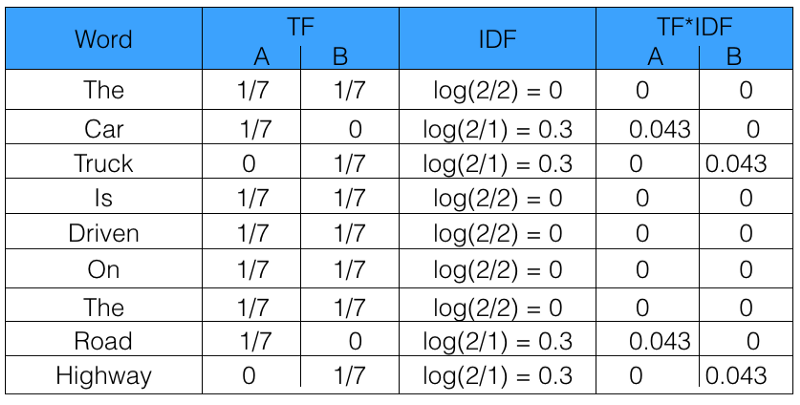
stands for term frequency-inverse document frequency is a statistical measure used to evaluate how important a word is to a document in a collection of documents.

**TF: Term Frequency**, which measures how frequently a term occurs in a document

**IDF: Inverse Document Frequency**, which measures how important a term is based on the number of appearance of the term, as some terms might appear a lot of times (such as an, of, and which)

We want to give these words less weight than the rare appearance words While computing TF, all terms are considered equally important.

The formula that is used to compute the tf-idf for a term t of a document d in a document set is



Example of tfidf-matrix

*Pic source:* <https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/>

I used ***tfidfVectorizer*** from sklearn python library toConvert the questions in the dataset to a matrix of TF-IDF features. I converted both question1 and question2 to lists then saved them in one list called questions (in order to have the same length) after that I applied ***tfidfVectorizer*** The output matrix is a sparse matrix (most of the elements are zeros)

**TfidfVectorizer Parameters:**

* The input is a list of strings with all the questions
* **max\_df**
* When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold 0.5.
* **min\_df**

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold (10)

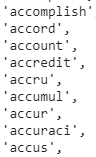
* **use\_idf**

Enable inverse-document-frequency reweighting.

* **smooth\_idf *= 1*** constant “1” is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions.
* **sublinear\_tf *= 1***

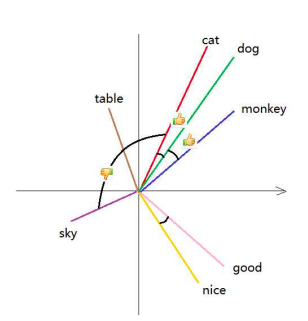
Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

Some of the features names:

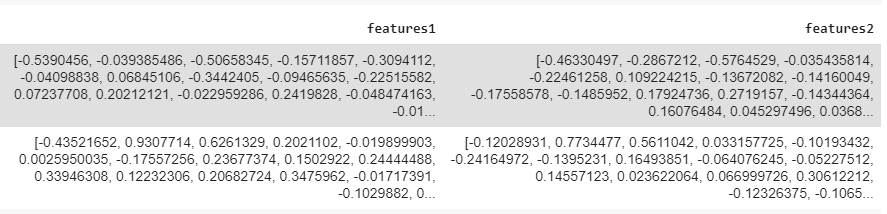


### **Word Embedding**

Is representing words as vectors of real numbers where the words that have the same meanings have similar vectors representation. each word will be represented by an array (word vector).



For this step I used **en\_core\_web\_md** package that come with built-in word vectors that was pre-trained on GloVe vectors this package can be found in **SpaCy** library. **en\_core\_web\_md** contains 20000 pretrained word vectors. The rows for the 20k most frequent words in the vocab. All other words are mapped to their nearest neighbor within those frequent words. Words that are not in the vocabulary will be presented as zeros. After obtaining word vectors with *.vector* we will average the word vectors to get the sentence (the document) vectors. Now each vector will represent sentence(document) vector.



**Difference between TF-IDF and word embeddings:**

|  |  |
| --- | --- |
| **TF-IDF** | **Word Embeddings** |
| Creates one number per word | Creates one vector per word |
| Good for classification | Good for identifying contextual content |

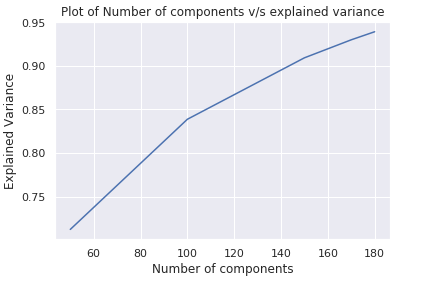
## Dimensionality Reduction

### Principal Component Analysis (PCA)

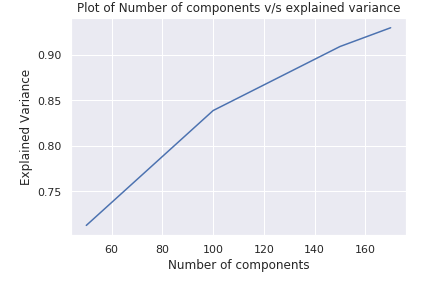
Unsupervise linear dimensionality reduction technique that takes large data dimension and returns lower data dimension by using clustering. it keeps the components with high variance which means we won't lose valuable features. **principal component** is the output of converting correlated variables to uncorrelated variables. Advantage of applying PCA is to speed up the machine learning algorithm as well as reducing the noise in the dataset

Using PCA package from *sklearn.decomposition* library to reduce the dimensionality of the features in features 1 and features 2 which represents the documents vectors.

One of the important parameter is the number of components as we are trying to choose a number of components that will reduce dimension but without losing information we can achieve that by iterating through PCA and changing the number of components number until we get to the optimal number of components with the highest explained variance.



The optimal number of components is 180 components with 94% explained variance for question1. Similarly for question2.



I was able to reduce the dimension while keeping 94% of the explained variance.





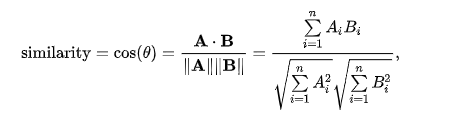
## Feature Extraction

### **Cosine Similarity**

is a metric to measure how similar documents are regardless of their length. it measures the cosine of the angle between two vectors by looking at their directions (or orientations) regardless of their length.

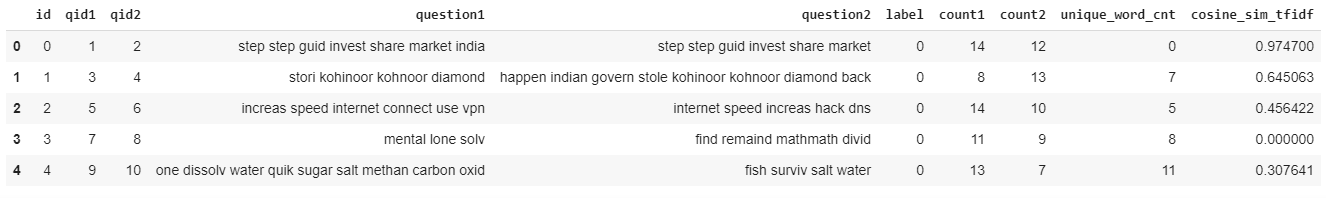
two vectors with the same orientation at 0° have a cosine similarity of 1, two vectors oriented at 90° have a similarity of 0.

The cosine similarity equation is the ratio of the dot product of the vectors and the product of the magnitude for the vectors :



Using cosine similarity package from ***sklearn.metrics.pairwise*** we can obtain the cosine similarity between the questions by applying it on tf-idf matrix. To do so I split tf-idf matrix into two matrices then I started for loop to go over the documents one by one. So the tf-idf for each document from question 1 will be compared to the tfidf from question 2.

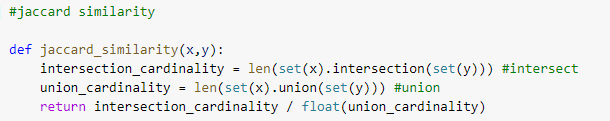
Then I added the list of cosine similarity output to the data frame as ‘***’cosine\_sim\_tfidf’*** attribute.



### **Jaccard index/Similarity coefficient**

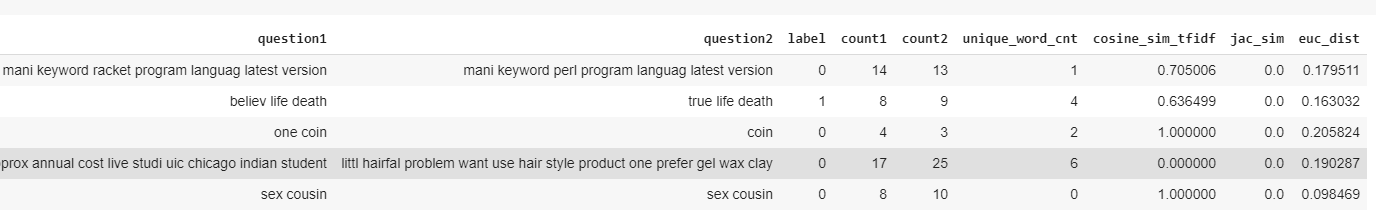
It compares members for two sets to see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data. The formula is given below where x is the first set and y is the second set:

Building jaccard similarity function from the above formula:



Take the intersect first then the union then calculate the fraction between these two after we get the jaccard similarity we normalize the data between the range (0,1)

After that we add the normalized data into the data frame as ‘***jac\_sim’*** attribute.



### **Euclidean distance**

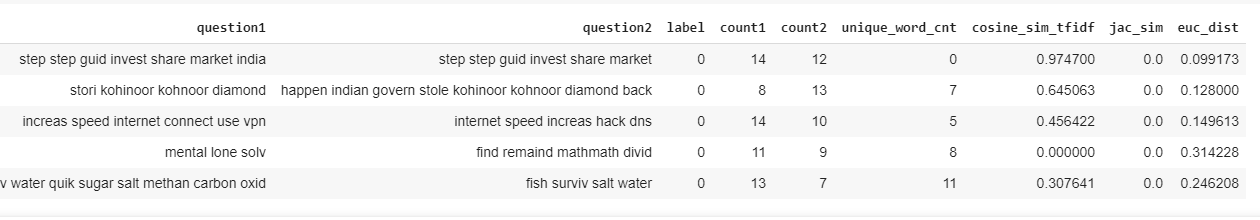
Is the line distance between two points. Euclidean distances formula is given below:



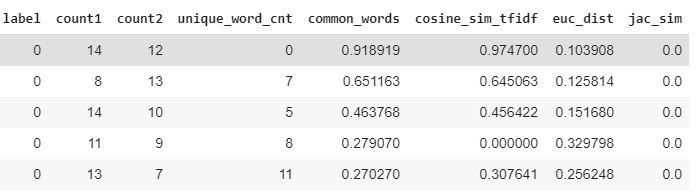
Using ***pairwise.euclidean\_distances*** from ***sklearn*** library we can compute the distance matrix between each pair of document vectors.

For efficiency reasons, the Euclidean distance between a pair of row vector x and y is computed as:

After calculating the Euclidean distance between each document. I added the distance into the data frame as ***‘euc\_dist’*** attribute. The smaller the distance the closer the document to each other which means they are more similar.



I dropped id, qid1, qid2 as they are unique identification number for the questions and pair of questions so they will not affect the model. In addition to dropping question1 and question2 because they are string values and we can’t feed out classification model text data.



I did not normalize the data. **Normalization** which means rescaling the data into the range[0,1]. Because Decision Trees do not require normalization of their inputs; and since XGBoost is essentially an ensemble algorithm comprised of Decision Trees, it does not require normalization for the inputs either.

## Classification

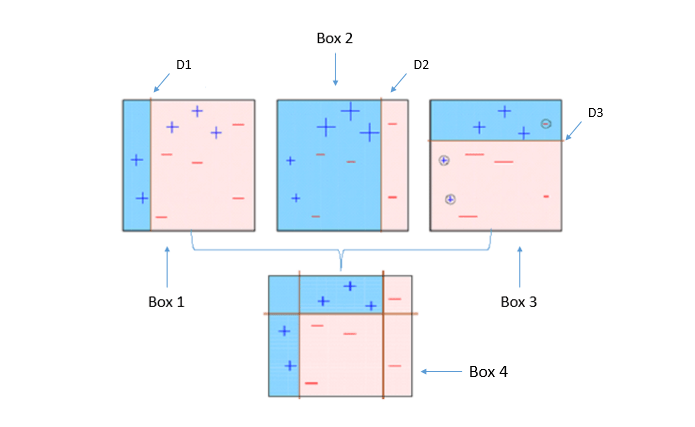
### **Modeling**

Staring with defining some methodologies first:

**Ensemble learning** use multiple algorithms to obtain better predictive performance.

**Tree ensembles** use multiple decision trees to obtain better predictive performance where each tree learn from the previous one and give more weight to the misclassified labels.

**Boosting** is a sequential techniqueworks on the concept of ensemble by learning from previous mistakes where it gives more weight to weak learners compared to strong learners. The example below shows how boosting works.



**Pic Source:** <https://www.datacamp.com/community/tutorials/xgboost-in-python>

**Box 1:** the first classifier creates a line that separate + from – although it classifies two + correct it fails to classify three + to the right.

**Box 2:** the second classifier learns from the first one mistakes and give more weight to the misclassified ones. The classifier will catch the three + that missed in the first one but will include three – classified incorrectly as +.

**Box 3:** the third classifier again will learn from the previous one mistakes and give more weight to the misclassified ones. This classifier will classify three + correctly as well as four – correctly but will fail to classify two + on the left.

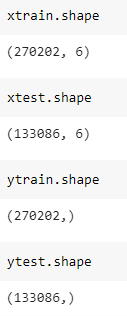
**Box 4:** The last classifier will give more weight to the misclassified signs in all the boxes (box 1 , box 2 and box 3). This will capture all the + and – signs correctly by learning from all the previous classifiers.

### **Train/Test Split:**

We split the dataset into two sets. Train set that has known label either duplicate or not. We will use the dataset in the train set to train the model and later on will test our model with the data that the model has never seen before in our test set.

***Train set 67%***

***Test set 33%***



Some of the challenges that will face from train/ test splits are:

**Underfitting** when the model fails to represent the trend in the training data

**Overfitting** when the model scores high on the training data but fails to represent the trend in the testing data as a result the model cannot be generalized.

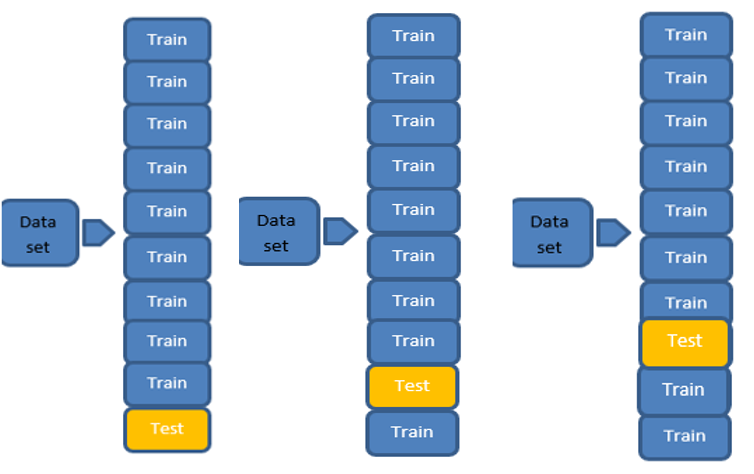
***To overcome these challenges I will apply 10 fold cross validation to the XGBoost classifier.***

### **10 fold Cross Validation**

At the begging of the experiment we will hold out a test dataset that will be used to evaluate the model and separate train set that will be handled as follows:

1. The training set is randomly partitioned into ten folds.
2. Ten models are built out of these ten folds. Each model is built using nine of the folds as a training set, and one fold as a testing set.
3. The ten models are averaged to create a single model balanced for randomness.

Lastly the averaged model will be tested using the test data that was hold out in the beginning of the experiment.



This process will be repeated on 10 folds

### **XGBoost**

stands for “Extreme Gradient Boosting” is a supervised learning technique. It is an ensemble of decision trees algorithm where new trees fix errors of those trees that are already part of the model. Trees are added until no further improvements can be made to the model.. XGBoost is an effective machine learning model, even on datasets where the class distribution is skewed.

I used XGBoost classifier from sklearn library to build the model with the default parameters.

### **XGBoost Hyperparameter Tuning:**

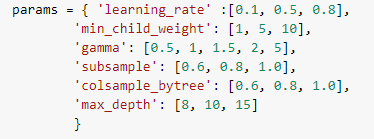
It means choose the parameter that fit the data the best or our model the best.  In tree-based models, hyper-parameters are the depth of the tree, the number of trees to grow, the minimum number of samples on a leaf, the fraction of observations used to build a tree, and others.

In order to tune the hyperparameter I used Randomized search CV where we define parameters earlier with range of values the randomized search CV selects random combination of hyperparameters.

**Hyperparameter tuning:**

* **Learning Rate** the shrinkage we apply at every step If we make 1 step at eta = 1, the step weight is 1. If we make 1 step at eta = 0.25, the step weight is 0.25. it is used to prevent overfitting
* **Max Depth** how deeply the tree can grow after each step
* **Min\_child\_weight** Defines the minimum sum of weights of all observations required to create a new node in a tree
* **Gamma** A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
* **Colsample\_bytree**  Denotes the fraction of columns to be randomly samples for each tree.
* **Subsample** the ratio of the training instances. sum(negative instances) / sum(positive instances) I changed scale\_pos\_weight which refers to the **data imbalance** in our case 60% is non duplicate so I changed scale\_pos\_weight to 6 which means for every observation= 0 there is 6 observations = 1.

**I applied random grid search on these parameters:**



The random search was done using 10-fold cross validation and the scoring was conducted with roc-auc. I applied the random search on xtrain, ytrain. After that the best estimator was returned from the search result for the parameters values. By using these best tuned parameter we train the model then evaluate the training accuracy using 10 fold cross fold validation. the accuracy result was averaged from all ten models.

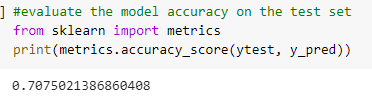
### **Evaluation**

For the evaluation I used the following measurements:

**Accuracy**

is the fraction of right prediction. Calculated by the number of correct prediction divided by the total number of predictions

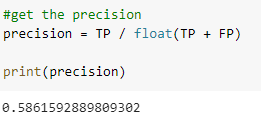
The model accuracy is 70%



**Precision**

(also called positive predictive value) is the number of items correctly identified as positive out of total items identified as positive.

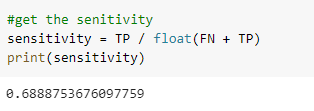
The precision score is 58.6%



**Recall**

(also known as sensitivity) It is the number of items correctly identified as positive out of total true positives.

The model precision is 68.8%



**Specificity**

(also called the true negative rate) is the number of items correctly identified as negative out of total negatives.

**ROC\_AUC**

ROC (*Receiver Operating Characteristic*) it plots two parameters True positive rate on Y-axis and False positive rate on X-axis

AUC (Area Under The Curve) It measures the entire 2Dimensional Area covering X-axis & Y-axis under the ROC curve from (0,0) to (1,1).

**GitHub link:**

<https://github.com/SaraKmair/Quora-Question-Pairs>