Quora Insincere Questions Classification

CIS 668/ IST 664 - Natural Language Processing

Shwet Jain
Digvijay Sonawane
Himanshu Patel
Nakul Sahayata

Contents

Introduction

Quora is a question and reply website wherein a man or woman can ask a query and get options from different users who think they have a solution for it. The solutions can be factual or opinion. It is also a mega web page and receives around one hundred million visitors in line with month and about 25000 questions are requested and 5000 solutions are written on a day by means of day basis. When we have such a lot of questions and answers being delivered at the platform there may also be a prefer for moderation. One of the key obligations of moderation is to clear out the questions which are irrelevant, these centered about the false premises or may also have a rhetorical query however than search for useful solutions.

Objective

The fundamental goal of this project is to predict if a question requested on the Quora platform is a legitimate query or not. Some of the features/characteristics that predict whether a given query asked in Quora is sincere by way of using NLP are given below.

- Based on false statistics or if it incorporates assumptions that are absurd.
- Makes attacks or insults which are disparagingly targeting a team of people, or a unique person.
- If it is now not grounded to reality; if it is inflammatory
- Make use of sexual content material for shock price and now not to get an answer which is genuine in nature.
- Has a non-neutral tone, has an exaggerated tone to underscore a point about a team that seeks affirmation of a stereotype.
- Based on an outlandish premise about a crew of people

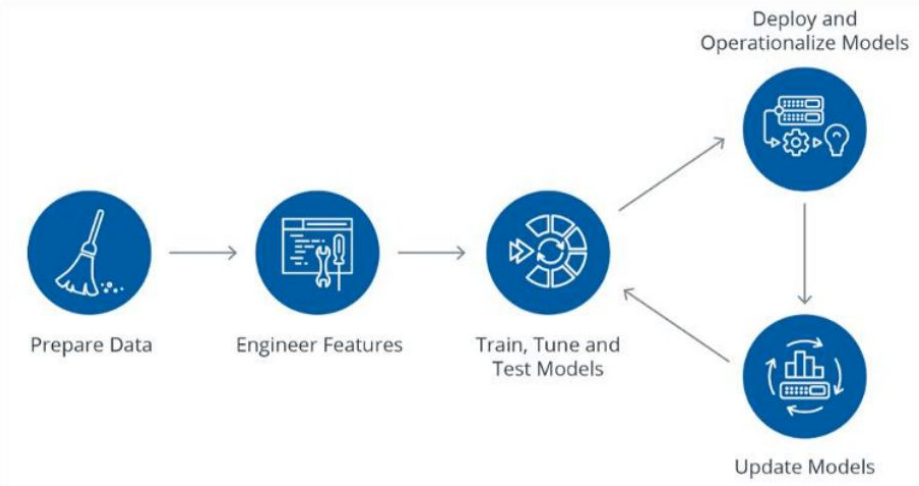
Representative Technique and Dataset in our Project

- In Natural Language Processing, Text classification is a crucial assignment that transforms a sequence of a given textual content of indefinite size into a class text. This is one of the most extensively used Natural Language Processing duties in more than a few enterprise problems. The most important intention of the Text Classification is to classify the text of the archives mechanically into one or greater categories which are already defined.
- Few examples of Text Classification are:
    - Understanding the sentiment of the people from social media and classifying it as nice or negative.
    - Auto-tagging consumer queries.
    - In the Process of detecting unsolicited mail and non-spam emails primarily based on the situation and body of the email.
    - Mapping/categorizing the new subjects into already defined topics.

To put into effect this project, we used a dataset that consists of about 1.7M documents from Kaggle. The facts have already been divided into two sets namely check and teach sets. The instruct set has about 1.3 M files and the Test set has about 380K records. The dataset has the following three fields:

1. **Qid** - this is a Unique identifier which differentiates one question from another.
2. **Question_text** -This contains the Quora question text.
3. **Target** - Here a question which is labeled “insincere” has value 1, else it is named 0.

Text Classification Approach



Dataset Preparation

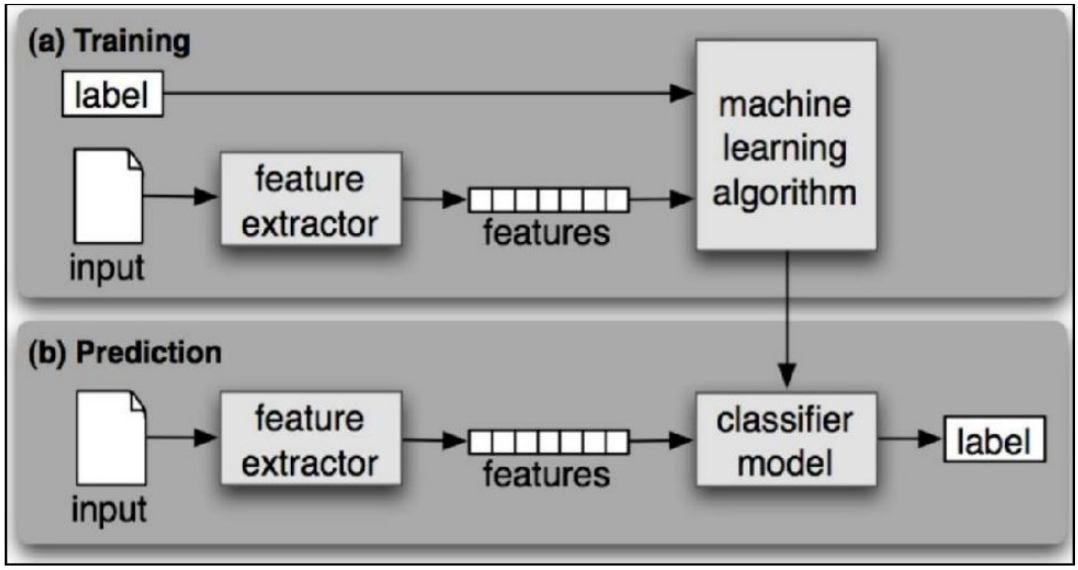
This is the first step in the course of which the dataset that is accumulated is loaded and simple preprocessing steps like information cleansing and guidance are performed. The information set is then divided into training, validation and trying out sets.

Feature Engineering

The facts that are gathered are in the form of text. But the computer can't understand the words or sentences, so it is indispensable to convert it into a structure the computer can understand. So, facets are extracted from the dataset and from which the machine will learn, and also new elements are created.

Model Training

This is the final step during which various machine learning models are built and trained and tested on the labeled dataset.



As proven in the image the dataset we have collected which has labels for the target values is given as an input to the feature extractor. This gave us the necessary features and characteristics of the dataset. These features are then given to a number of machine learning models that are discussed in the later sections. These models used the K fold training technique for the duration of which the training set is divided into k equal segments and for each epoch, one phase is regarded as the trying out set. Also for the duration of the training, it will try to work on the validation units to check if there is any overfitting. This step will supply us various parameters like F1-score, precision, and recall of the coaching set. Once the training of the model is executed the testing set that is firstly received through dividing the dataset is used to

check if the model used to be capable to classify the sincere questions. We can check the prediction accuracy of the F1 rating of information during this step.

Data Analyzation

The records from the data set is analyzed by dividing the dataset into training and testing datasets as follows. After having two documents as train.csv and test.csv, we did our sentiment analysis using these two files.

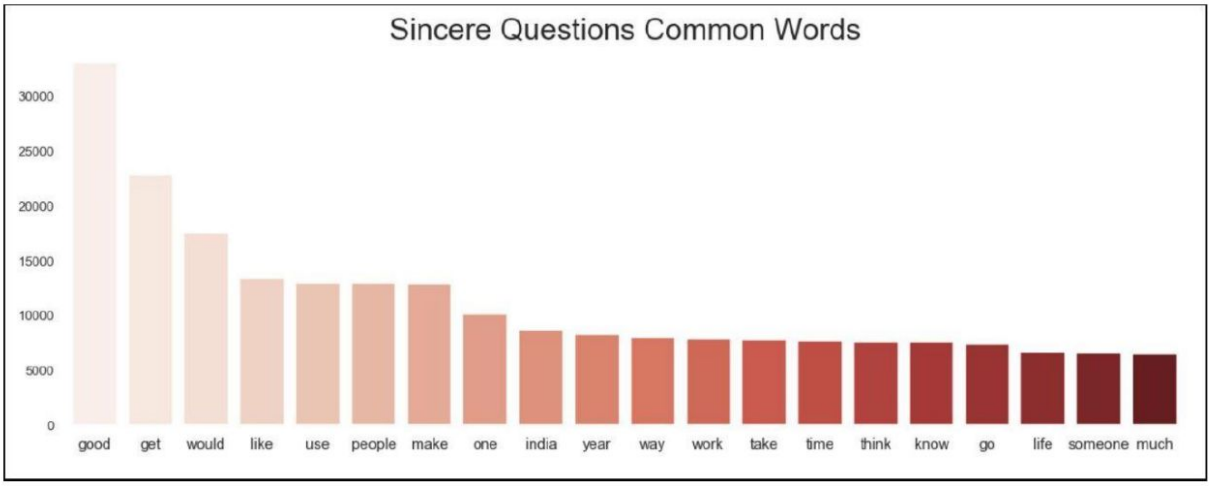
```
[10]  1 train_df = pd.read_csv("drive/My Drive/train.csv")
      2 given_test_df = pd.read_csv("drive/My Drive/test.csv")
      3 print("Train shape : ",train_df.shape)
      4 print("Test shape : ",given_test_df.shape) #useless

      Train shape :  (1306122, 3)
      Test shape :  (375806, 2)
```

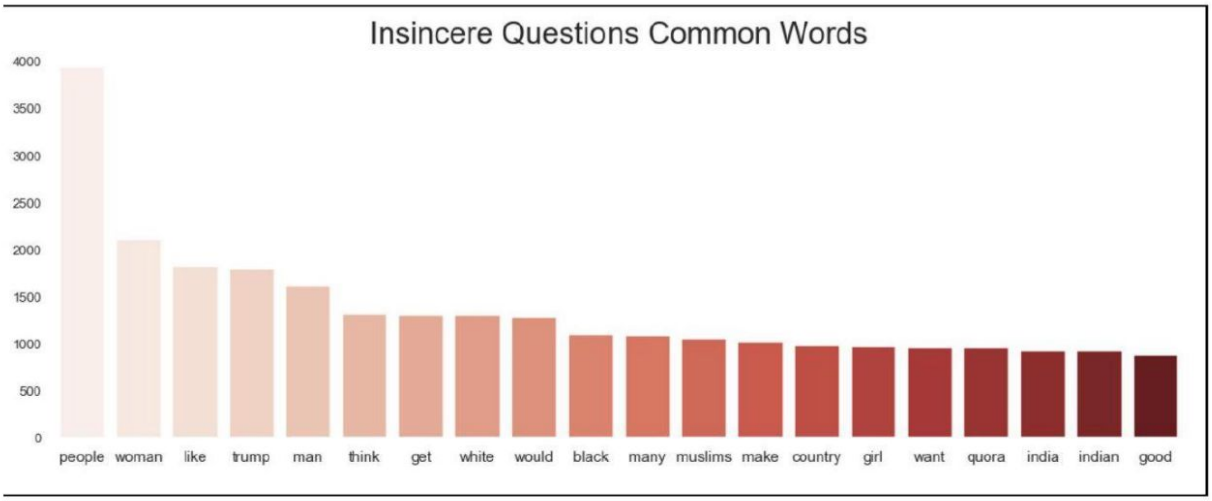
The libraries we used in our project to do the sentiment analysis are:

- **Pandas** is an open-source software library that is written for the python programming language and is mainly used for data manipulation and analysis. In this project we have used the pandas to create the data frames and also for many data analysis tasks.
- **Scikit-learn** is also an open source software library that supports various machine learning concepts like clustering, regression, support vector machines etc. It is used in this project to implement the machine learning algorithms that are performed on the test data to obtain the classification.
- **XGBoost** is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. Here, we used it to implement the boosting trees as a part of our machine learning approach.

- **TextBlob** is a library for processing textual data and it provides a simple API for diving into common natural language processing tasks such as PoS tagging, sentiment analysis, classification and more.

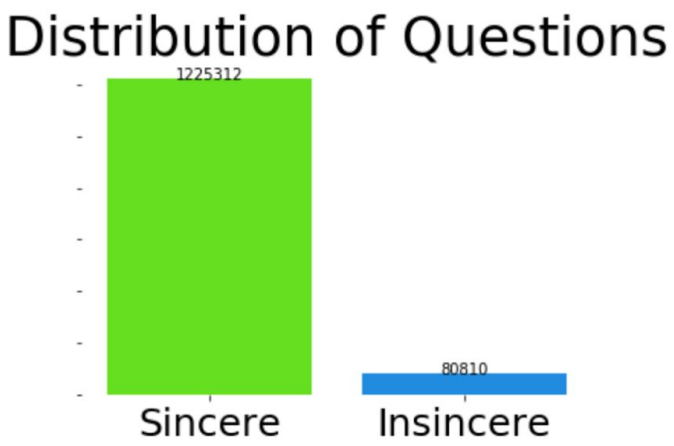
Comparison between Sincere and Insincere questions in dataset:



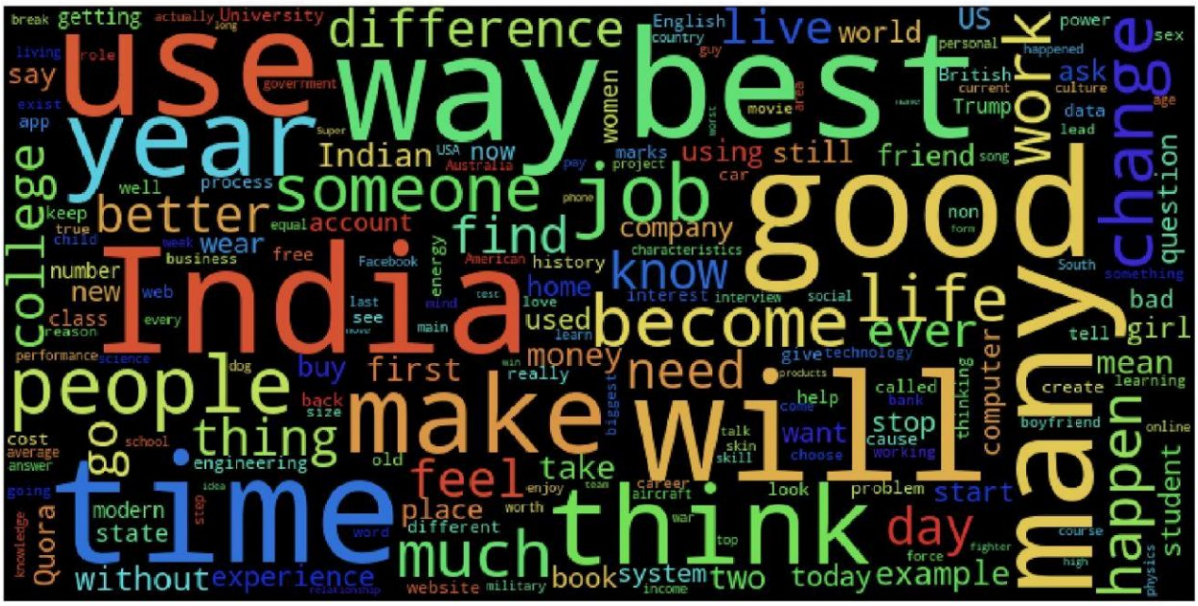


Of the many times occurring words in the test data set that are classified as sincere. It can be considered that ‘good’ takes place in most of the questions with a count of above 3000 and ‘much’ occurs someplace round five hundred times.

The above layout shows the distribution of most often occurring words in the test data set that are classified as insincere. It can be considered that ‘people’ take place in most of the questions with a count of above 3500 and ‘good’ happens someplace round one thousand times.

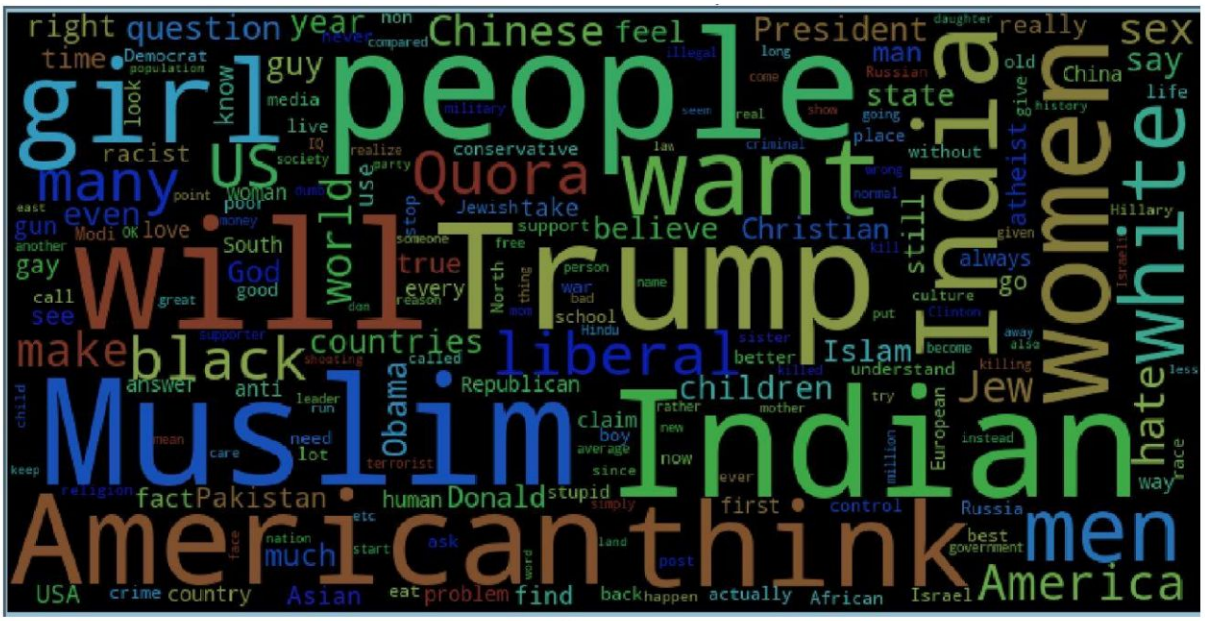


The above distribution shows that the questions are quite undistributed with the majority of questions are Sincere (i.e labeled as 0)and very few questions are Insincere (i.e labeled as 1) in the test data set .



Word Count of sincere questions

The above figure describes a word cloud consisting of texts that show up most often in the sincere questions. The more frequent the word in that category, the bigger it appears in the world cloud and vice versa.



Word Count of insincere questions

The above figure describes a word cloud in which it can be viewed that the frequency of sure comparable words is greater in each the categories, i.e. India and Indian take place quite frequently in each sincere and insincere question existing in the test data set. Also, there are some words in the insincere sentences word cloud such as American, Muslim, Trump and many others which do now not make sense on its own.

	word	freq	percentage
0	tips someone starting	716	12.402564
1	useful tips someone	713	12.350598
2	someone starting work	713	12.350598
3	advice give someone	640	11.086090
4	give someone moving	519	8.990126
5	good hotels short-term	519	8.990126
6	hotels short-term business	519	8.990126
7	short-term business travelers	519	8.990126
8	good bad neighborhoods	515	8.920838
9	best known for?	400	6.928807

Fig: most common sincere n grams

	word	freq	percentage
0	will donald trump	43	12.250712
1	black lives matter	42	11.965812
2	long will take	38	10.826211
3	kim jong un	36	10.256410
4	12 year old	35	9.971510
5	14 year old	33	9.401709
6	people still believe	33	9.401709
7	united states america	31	8.831909
8	think donald trump	30	8.547009
9	ask stupid questions	30	8.547009

Fig: most common insincere n grams

Data Cleaning

The following steps are involved in data cleaning.

- Remove Characters: For getting rid of any specific characters, punctuations, we have used the regex function. Using this function, putting off distinctive characters was very simplified.
- Tokenization: Tokenization is the way toward setting apart content into a lot of significant pieces. These portions are known as tokens. For instance, we can isolate a lump of content into words, or we can partition it into sentences. Contingent upon the job that needs to be done, we can signify our very own conditions to separate the information content into important tokens.
- Missing values: When the missing features were taken care of and the sentences have been gotten in a professional arrangement in the wake of making use of everyday articulations and lower packaging them, tokenization used to be utilized to get words as the tokens in a rundown group.
- Padding: Strings can be padded with spaces or different characters to a high-quality width. For example, some numerical, abbreviated, or textual data are often represented with prepending blank areas to make certain the data is unclustered and unambiguous for classification.

Feature Engineering

The next step in the project is the feature engineering step. In this step, raw text records were transformed into feature vectors and with the use of the current dataset, new features will be created.

We will be implementing the following distinct ideas in order to collect applicable features from our dataset.

TF-IDF Vectors as features

● **N-Gram level: N-grams are the combination of N terms together. This Matrix representing TF-IDF scores of N-grams (represents the relative importance of a term in the document and the entire corpus)**

$$\text{TF(word, text)} = \frac{\text{number of times the word occurs in the text}}{\text{number of words in the text}}$$

$$\text{IDF(word)} = \textbf{log}\left[\frac{\text{number of texts}}{\text{number of texts where the word occurs}}\right]$$

$$\text{TF-IDF(word, text)} = \text{TF(word, text)} \times \text{IDF(word)}$$

Model Building - Classification models

There are many machine learning classification models including Random Forest, Logistic Regression, XGBoost, Naive Bayes, SVM. These models are useful when we need to draw some conclusions from data. Here we have listed the accuracy of some classifiers that we implemented applying features that we built in the previous step.

Logistic Regression

Logistic Regression models are the machine learning classification model, especially binary classification, models. They are used when there are two categories to choose from for example if the email is spam email or not spam email.
In our Quora insincere questions observation, we have used Logistic regression as the base model because either the asked question is sincere/logical or it is insincere.

Here we will see how Logistic Regression calculates the Probability
If we want to know the question is sincere or not we will first calculate its weighted sum of weights. This weighted sum will go to the sigmoid function as input.

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

With this function we can calculate the value of y here y is dependent variable and x1,x2 .. xn are explanatory variables

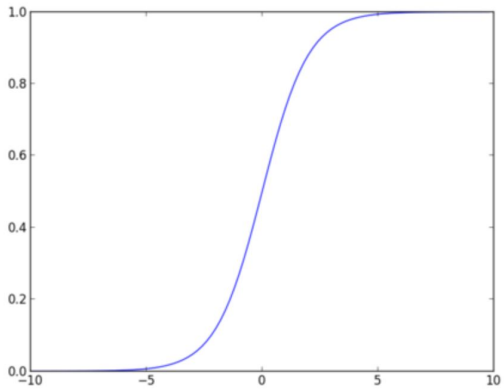
Sigmoid Function

$$p = 1/1 + e^{-y}$$

- The dependent variables follow Bernoulli distribution

- Sigmoid function makes ‘S’ type of graph and it can map any given value between 0 and 1. If value goes towards Positive Infinity then Y will become 1 that as an outcome is YES.

  If value goes towards Negative infinity then Y will become 0 as an outcome is YES




```
# split training data to train and test data
train_data, test_data = train_test_split(train_df, test_size=float(1.0/8), random_state=2018)

# storing question_text and target columns in different lists
train_text = train_data['question_text']
valid_text = val_data['question_text']
test_text = test_data['question_text']
train_target = train_data['target']
valid_target = val_data['target']
test_target = test_data['target']
all_text = train_text.append(train_text)

# applying TFIDF and count vectorization to question text and target values for each training and t
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(all_text)

train_text_features_tf = tfidf_vectorizer.ktransform(train_text)
test_text_features_tf = tfidf_vectorizer.transform(test_text)

# using kfold technique for fitting and predicting using Logistic Regression
kfold = KFold(n_splits=5, shuffle=True, random_state=2018)
test_preds = 0
oof_preds = np.zeros([train_data.shape[0],])

classifier1 = LogisticRegression()
```

With using the above code we’re getting the following accuracy.

```
pred_train = np.where(pred_train > 0.25, 1, 0)
print("Training F1 score: ",f1_score(train_target, pred_train))

# training data accuracy score
print("Training Accuracy: ",accuracy_score(train_target, pred_train))

pred_test=classifier1.predict_proba(test_text_features_tf)[:,1]
pred_test = np.where(pred_test > 0.25, 1, 0)
print("Testing F1 score: ",f1_score(test_target, pred_test))

# training data accuracy score
print("Testing Accuracy: ",accuracy_score(test_target, pred_test))
```

```
fitting.......
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\sklearn\linea
r_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
Training F1 score:  0.6450546290971824
Training Accuracy:  0.9524717024717024
Testing F1 score:  0.624108995759271
Testing Accuracy:  0.9489667168914532
```

Naive Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

There is not one algorithm for training such classifiers, but a family of algorithms supported a standard principle: all naive Bayes classifiers assume that the worth of a specific feature is independent of the worth of the other feature, given the category variable. For instance , a fruit could also be considered to be an apple if it's red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of those features to contribute independently to the probability that this fruit is an apple, no matter any possible correlations between the colour , roundness, and diameter features.

The main goal of Naive Bayes Classifier is to work out the possibilities of feature occurring in each class and to return the category with the very best probability. this will be through with the assistance of the Bayes rule.

The formula for Bayes rule is given as: $P(A \setminus B) = \frac{P(B|A)\,P(A)}{P(B)}$

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$\begin{aligned} P(c_i|x_0,\ldots,x_n) &\propto P(x_0,\ldots,x_n|c_i)P(c_i) \\ &\propto P(c_i)\prod_{j=1}^{n} P(x_j|c_i) \end{aligned}$$

In our project, we've used Multinomial Naive Bayes classifier for text classification. We used this type of Naive Bayes classifier because this multinomial classifier is best fitted to text classification.

In [54]:

```
count_vectorizer1 = CountVectorizer()
count_vectorizer1.fit(all_text)

train_text_features_cv = count_vectorizer1.transform(train_text)
test_text_features_cv = count_vectorizer1.transform(test_text)

# using kfold technique for fitting and predicting using Naive Bayes
kfold = KFold(n_splits=5, shuffle=True, random_state=2018)
test_preds = 0
oof_preds = np.zeros([train_data.shape[0],])



classifier1 = MultinomialNB()
classifier1.fit(train_text_features_cv,train_target)
pred_train=classifier1.predict_proba(train_text_features_cv)[:,1]
# training data F1 score
pred_train = np.where(pred_train > 0.25, 1, 0)
print("Training F1 score: ",f1_score(train_target, pred_train))

# training data accuracy score
print("Training Accuracy: ",accuracy_score(train_target, pred_train))

pred_test=classifier1.predict_proba(test_text_features_cv)[:,1]
pred_test = np.where(pred_test > 0.25, 1, 0)
print("Testing F1 score: ",f1_score(test_target, pred_test))

# training data accuracy score
print("Testing Accuracy: ",accuracy_score(test_target, pred_test))
```

Code snippet:

With using the above code we're getting the following accuracy.

Training F1 score: 0.5487336305239224

Training Accuracy: 0.9204589204589204

Testing F1 score: 0.5434520858417735

Testing Accuracy: 0.9210368355934487

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a mess of decision trees at training time and outputting the category that's the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The Random Forest method offers more randomness and variation by applying the bagging method to the feature space. That is, rather than seeking greedily for the simplest predictors to create branches, it randomly samples elements of the predictor space, thus adding more heterogeneity and reducing the variance of the trees at the value of equal or higher bias. This process is also referred to as “feature bagging” and it's this powerful method that results in a more robust model.

Code snippet :

In [38]:
```
# using kfold technique for fitting and predicting using Random Forest Classifier
from sklearn.ensemble.forest import RandomForestClassifier

kfold = KFold(n_splits=5, shuffle=True, random_state=2018)
test_preds = 0
oof_preds = np.zeros([train_df.shape[0],])

classifier5 = RandomForestClassifier()
print('fitting.......')
classifier5.fit(train_text_features_tf,train_target)
pred_train=classifier5.predict_proba(train_text_features_tf)[:,1]
# training data F1 score
pred_train = np.where(pred_train > 0.25, 1, 0)
print("Training F1 score: ",f1_score(train_target, pred_train))

# training data accuracy score
print("Training Accuracy: ",accuracy_score(train_target, pred_train))

pred_test=classifier5.predict_proba(test_text_features_tf)[:,1]
pred_test = np.where(pred_test > 0.25, 1, 0)
print("Testing F1 score: ",f1_score(test_target, pred_test))

# training data accuracy score
print("Testing Accuracy: ",accuracy_score(test_target, pred_test))
```

With using the above code we're getting the following accuracy.

Training F1 score: 0.9973402566617047

Training Accuracy: 0.9996701246701246

Testing F1 score: 0.5956762921872569

Testing Accuracy: 0.952231328016856

SVM

SVM Stands for Support Vector Machines; it is both classification and regression model but mostly used for classification purposes. It is a representation of various classes in a hyperplane in multidimensional space.

SVM is implemented with a kernel that reconstructs an input data space into the expected form. Low dimension input space is converted to a higher-dimensional space using kernel trick. It adds more and more dimensions and separates the problems which are not-separable. There are some types of Kernels used by SVM like Linear Kernel, Polynomial Kernel, RBF Kernel. SVM is commonly used in aspect based recognition, handwritten digit recognition, and image recognition challenges.

Here in our implementation, we have implemented LinearSVC; it is another implementation of Support Vector Classification for the case of a linear kernel.

```
In [48]: train_vectorized = vectorizer.transform(train_df.question_text.values)
         train_vectorized

Out[48]: <1306122x234530 sparse matrix of type '<class 'numpy.float64'>'
                with 10058097 stored elements in Compressed Sparse Row format>

In [49]: test_vectorized = vectorizer.transform(test_df.question_text.values)

In [50]: X_train, X_val, y_train, y_val=train_test_split(train_vectorized,train_df.target.values,test_size=0

In [51]: from sklearn.svm import LinearSVC
         svc = LinearSVC(dual=True,C=5,penalty='l2',max_iter=1000,tol=0.01)
         svc.fit(X_train, y_train)

Out[51]: LinearSVC(C=5, class_weight=None, dual=True, fit_intercept=True,
                   intercept_scaling=1, loss='squared_hinge', max_iter=1000,
                   multi_class='ovr', penalty='l2', random_state=None, tol=0.01,
                   verbose=0)

In [52]: svc_preds=svc.predict(X_train)

In [53]: from sklearn.metrics import f1_score, precision_score,recall_score
         print("F1 Score " , f1_score(y_train,svc_preds))
         print("Precision Score " ,precision_score(y_train,svc_preds))
         print("Recall Score " ,recall_score(y_train,svc_preds))
```

With using the above code we're getting the following accuracy.

```
In [53]: from sklearn.metrics import f1_score, precision_score,recall_score
         print("F1 Score " , f1_score(y_train,svc_preds))
         print("Precision Score " ,precision_score(y_train,svc_preds))
         print("Recall Score " ,recall_score(y_train,svc_preds))

         F1 Score  0.8192840221885438
         Precision Score  0.8810703039701252
         Recall Score  0.7655955671052812
```

XGBoost

XGBoost is an implementation of gradient boosted decision trees providing high-performance and speed.

When we use a regular machine learning model like Linear regression or other models they train on a single dataset and apply that for prediction whereas in boosting various models are linked to perform a final one. Boosting trains model one after another as each new model is trained to improve the errors made by the former one.

For using XGBoost we have to convert our data into a format that XGBoost can handle which is DMatrix format.

As shown in this below figure we have used xgb.Dmatrix this will convert it into DMatrix format. We have also defined model_params which will be used in a gradient boosting ensemble.

Some of the parameters used are maxDepth which shows maximum depth of the decision tree, Objective the function which will be used, eta which is the learning rate of the algorithm.

```
In [24]:
def build_xgb(train_X, train_y, valid_X, valid_y=None, subsample=0.75):

    xgtrain = xgb.DMatrix(train_X, label=train_y)
    if valid_y is not None:
        xgvalid = xgb.DMatrix(valid_X, label=valid_y)
    else:
        xgvalid = None

    model_params = {}
    # binary 0 or 1
    model_params['objective'] = 'binary:logistic'
    # eta is the learning_rate, [default=0.3]
    model_params['eta'] = 0.3
    # depth of the tree, deeper more complex.
    model_params['max_depth'] = 6
    # 0 [default] print running messages, 1 means silent mode
    model_params['silent'] = 1
    model_params['eval_metric'] = 'auc'
    # will give up further partitioning [default=1]
    model_params['min_child_weight'] = 1
    # subsample ratio for the training instance
    model_params['subsample'] = subsample
    # subsample ratio of columns when constructing each tree
    model_params['colsample_bytree'] = subsample
    # random seed
    model_params['seed'] = 2018
    # imbalance data ratio
    #model_params['scale_pos_weight'] =

    # convert params to list
```

For training, the XGBoost model xgb.train() method is used. Here we have trained it with the train_xgboost function so when we need to train it this method will be called.

In [25]:

```
def train_xgboost(xgtrain, xgvalid, model_params, num_rounds=500, patience=20):

    if xgvalid is not None:
        # watchlist what information should be printed. specify validation monitoring
        watchlist = [ (xgtrain, 'train'), (xgvalid, 'test') ]
        #early_stopping_rounds = stop if performance does not improve for k rounds
        model = xgb.train(model_params, xgtrain, num_rounds, watchlist, early_stopping_rounds=patie
    else:
        model = xgb.train(model_params, xgtrain, num_rounds)

    return model
```

With using the above code we're getting the following accuracy.

```
    print('F1 score: {} for threshold: {}'.format(score, threshold))

    scores_list.sort(key=lambda x:x[1] , reverse=True)
    best_threshold = scores_list[0][0]
    print('best threshold to generate predictions: ', best_threshold)
    print('best score: ', scores_list[0][1])
```

```
F1 score: 0.6209603699521428 for threshold: 0.2
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.6203346203346204 for threshold: 0.3
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.6189679737347458 for threshold: 0.31
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.6160849772382399 for threshold: 0.33
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.5994295028524858 for threshold: 0.4
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.5784676730276274 for threshold: 0.45
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
F1 score: 0.5581947743467933 for threshold: 0.5
best threshold to generate predictions:  0.2
best score:  0.6209603699521428
```

Results

Classifier	Train Accuracy	Test Accuracy
Logistic Regression	0.952	0.948
Naive Bayes	0.92	0.921
Random Forest	0.99	0.95
SVM	0.88	0.819