ML Project Report

Kappagantula Lakshmi Abhigna - IMT2019040
Vaibhavi Lokegaonkar - IMT2019090

December 25, 2021

1 Problem Statement

We were provided with an imaginary entity - CodeProject; a community of software developers. Where members from all over the world come together to share code, tutorials, and knowledge for free to help their fellow software developers.

A scenario was given - CodeProject's forums are being overrun with spam questions from beginners. The amount of spam on the website is more than can be manually filtered through.

We were to create a model, which, when given various details about a question and the account can predict whether the question should be deleted, and also predict the reason for deletion (from a small set of standard reasons i.e multiclass classification).

The classes for the target are:

- 0: "valid",
- 1: "kehna kya chahte ho"
- 2: "irrelevant"
- 3: "low quality"
- 4: "not reproducible"

2 Data Preprocessing

We performed the data processing in following three stages:

2.1 Preprocessing to generate text corpus

2.1.1 Understanding the Data

- PostDateTime: When the person has posted their question.
- WhenAccountMade: When the account was made
- Heading: The heading of the question
- MainText: The actual question (we realized that this included text + code)
- PrimarySubject, SecondarySubject, TertiarySubject and OtherSubject: Tags for each question (which subjects it pertains to)
- Target: The classification of the question as given in the actual problem statement

2.1.2 Word Clouds

Note: The data was really large, allowing us to generate word clouds for limited number of rows.

2.1.3 Basic Observations

- 'Karma' and 'NumAnswers': The more the number of questions answered and total karma earned, the higher the probability that the person is an experienced user and will ask valid questions.
- 'PostDateTime' and 'WhenAccountMade': They will help calculate the account age. The higher the account age, the longer the user has been on the platform.
- Text columns: We merged all the text columns to have the complete question along with the heading, text and tags.

To deal with the non-numeric columns, we performed operations on the existing data. The newly generated columns are discussed in the next few subsections.

2.1.4 Dealing with 'PostDateTime' and 'AccountDateTime'

- To uniformize the formats of the dates, All of the dates and times were converted into the format: **MM/DD/YY hh:mm:ss**
- Time was missing in some places. Wherever that took place, we added 00:00:00

- Finally, to make everything numeric, we converted all the dates and times to dateTime objects. We then took the difference of the date time columns and dropped them to obtain 'accountAge'.

2.1.5 Text Columns

- We combined all of the text columns to create 'PostText'.
- 'Heading', 'MainText', 'PrimarySubject', 'SecondarySubject', 'TertiarySubject', 'OtherSubject' were dropped.
- 'Qid' was also dropped.

After doing the above operations, we were left with:

	Karma	NumAnswers	Target	accountAge	PostText
0	1	0	0	12	Populate dropdown list using ajax In grails we...
1	230	1	0	317	How to query database where date and time fiel...
2	1	0	0	83	Character Casing in Silverlight 4 TextBox I am...
3	5273	381	0	406	Stop exception from being thrown for non-exist...
4	8	0	0	14	search for campaigns with multiple values for ...

Finally, we dealt with the nans (replaced them with '~') and checked for duplicate rows.

2.2 Preprocessing of the text corpus

2.2.1 Operations on Text Corpus

On the text corpus, we performed the following:

- Converted to lowercase
- Removed special characters
- Removed escape sequences
- Removed words of length less than or equal to 2 (most of such words were inconsequential to our cause)
- Removed stop words (used nltk first which took too long to run. We later switched to gensim)

2.2.2 Stemming

In stemming we chop off the ends of words. For example: troubling and trouble are both given as troubl after the stemming operation. We used Stemming on the text column in our data.

Note: One question might arise - why didn't we use lemmatization?
Lemmatization cuts the words so that things are grammatically correct with more meaning. It seems good. But our dataset was extremely large. Since doing lemmatization involves a timewise costly algorithm for a dataset that big, we didn't use it.

2.3 Conversion to Sparse matrices

- After creating the corpus, our first idea was to try to convert the csr format into a dense format. We couldn't do that due to the sheer size of the data. Our notebook crashed because of a ram shortage.
- We solved the problem by retaining the sparse format of the Document Term Matrix as obtained from the Count Vectorizer.

- We then converted the rest of the columns - 'Karma', 'NumAnswers' and 'accountAge' into a csr. We then merged that csr with the DTM to obtain our final csr which got us ready to start trying the initial models (we fed the csr matrices directly to the models).

We additionally did the following stage as a part of pre-processing towards the end of project.

2.4 Random Sampling

Our given dataset had severely imbalanced classes. For instance, there are 5 classes '0', '1', '2', '3', '4' , but the mean of the labels is '0.4'. It was highly skewed.

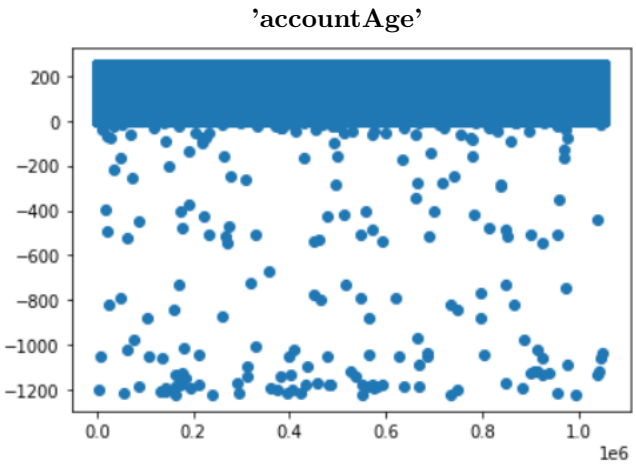
To solve this problem, we used the stratify parameter during train test split. Also, we performed random sampling

Random Sampling:
We did random oversampling of minority classes and undersampling for the majority class combined. Below mentioned were the proportions of the classes in the sampled set:

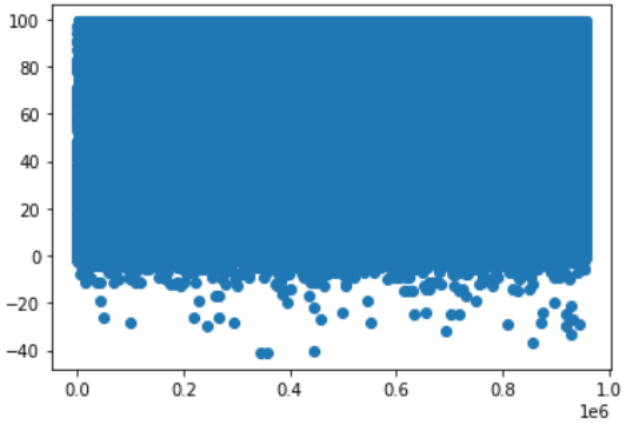
- 0: 30
- 1: 25
- 2: 20
- 3: 20
- 4: 5

These numbers align with the relative proportions of the original dataset as the application using the model would get similar kind of data.

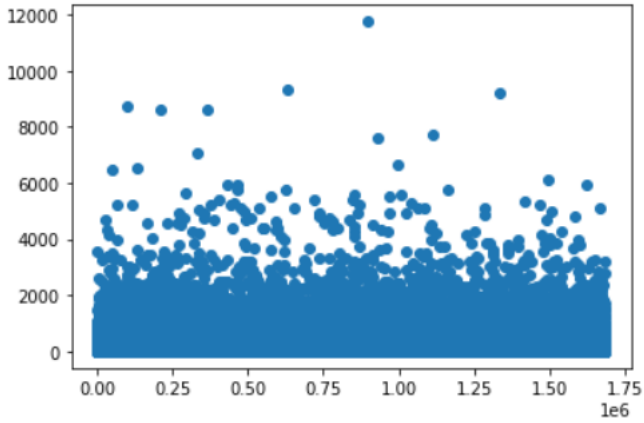
3 Exploratory Data Analysis



'Karma'



'NumAnswers'



As seen, 'accountAge' and 'Karma' have negative values. We discovered that this was a problem while trying to use the model Naive Bayes.

To deal with the negative values of 'Karma', we crafted the following function:

```
def make_col_positive(df, col):
    df[col] = df[col] + abs(df[col].min())
    return df
```

Basically, 'Karma' is just the total number of points accumulated by a user over a period of time through upvotes and downvotes. So, by adding all the values of 'Karma' with a particular value, no change is brought about in the impact of it. That is what we did.

In case of the negative values of 'accountAge', we understood why some values might be skewed (we had filled missing times with 00:00:00), but the large values left us a little baffled. It might be a long shot, but we assumed that for some of the values, there was a mistake in taking the data in (for eg: values for 'PostDateTime' and 'AccountDateTime' might've been swapped in some cases).

Since negative values were problematic only in some models, we dealt with those by taking the absolute of those negatives (here, only 'accountAge').

4 Models

We tried the following models (parameters included):

Sr No	Model Name	Parameters
	Complete Fit Models	
	(Problems: Large running times, unsatisfactory Macro F1-Score scores)	
1	MultinomialNB	*default*
2	ComplementNB	*default*
3	LogisticRegression (Attempt 1)	*multi_class = 'ovr', solver = 'liblinear', class_weight = 'balanced', max_iter = 150*
4	LogisticRegression (Attempt 2)	*multi_class = 'ovr', solver = 'liblinear', class_weight = 'balanced', max_iter = 200*
	Partial Fit Models (BATCH SIZE = 70,000)	
5	SGDClassifier()	*loss = 'log'*
6	SGDClassifier()	*loss = 'modified_huber', penalty = 'elasticnet', warm_start = True*
7	SGDClassifier()	*loss = 'squared_hinge', penalty = 'elasticnet', warm_start = True*
8	SGDClassifier()	*loss = 'modified_huber'*
9	SGDClassifier()	*loss = 'log', warm_start = True*
10	SGDClassifier()	*loss = 'perceptron'*
11	SGDClassifier()	*loss = 'perceptron', warm_start = True*
12	PassiveAggressiveClassifier()	*default*
16	Perceptron()	*default*
17	MLPClassifier()	*max_iter = 100, warm_start = True, solver = 'adam', hidden_layer_sizes = (2,)*
	Ensemble Models (with random sampling)	
18	XGBoost()	*(use_label_encoder = False, booster = 'dart', one_drop = 1, rate_drop = 0, max_depth = 6, tree_method = 'approx', objective = 'multi:softmax', num_class = 5)*
19	XGBoost()	*use_label_encoder = False, booster = 'dart', one_drop = 1,rate_drop = 0, max_depth = 6, tree_method = 'approx', objective = 'multi:softmax', num_class = 5*
20	XGBoost()	*use_label_encoder=False, eval_metric ='merror'*
21	XGBoost()	use_label_encoder=False, booster = 'dart', one_drop = 1, rate_drop = 0, eta = 0.01, gamma = 0.2, max_depth = 4,
22	XGBoost()	*use_label_encoder=False, booster = 'dart', one_drop = 1, rate_drop = 0, eta = 0.01, gamma = 0.2, max_depth = 4, tree_method = 'approx', scale_pos_weight = 2, objective ='multi:softmax', num_class = 5)*
23	BaggingClassifier()	*base_estimator = sgd_cls, n_estimators = 5*
24	RandomForestClassifier()	*default*

Even though we have tried all of the models above, in the multiple notebooks we submitted, not all possible variations of the models might be visible. Some models gave us unsatisfactory macro f1 score. So, we had deleted those from our code without writing.

5 Conclusion

The model that gave us the best accuracy was XGBoost with the following parameters:

xgb = XGBClassifier(eval_metric='merror')

xgb.fit(xtrain[: 4 * BATCH_SIZE], ytrain[: 4*BATCH_SIZE])

We took BATCH_SIZE = 70000. So we trained on 280000 rows.

Macro F1 score obtained with given test set: 0.22964

6 Future Scope

If we had the time, we could have tried more preprocessing methods like contraction. Some models we missed were tweaking more parameters in XGBoost and another model like LGBM.