MDSC-302(P)

Mini-project

Sentiment Analysis of Amazon Customer Reviews

Krishnakanth G

20233

Contents

Introduction....2
Problem statement....2
Scope of the solution....3
Challenges....3
Data Description....3
Data Pre-processing....4
Design of solution....7
Modelling....8
Naive bayes....8
Results and Analysis....10
Future works....10
Conclusion....11
References....11

Sentiment Analysis of Amazon Customer Reviews

Introduction

Getting customer feedback is a very important part of any business model. Only when we know how our customers feels about the product or services, we can serve our customers better and expand our business. In this project we use an amazon customer reviews data set to classify the reviews into positive or negative using sentiment analysis.

Problem statement

Problem statement: Sentiment analysis of Amazon kindle store reviews.

In this project we are going to pre-process the text and create a model to find out the sentiment that's contained in the review.

The tasks that must be done in this project are:

- Labelling the data
- Pre-processing the data
    - Basic cleaning
    - Negation handling
    - Word tokenization
    - Word lemmatization
    - N-gram models
- Modelling
    - Naive bayes
    - Logistic regression
    - Linear SVC

This project is done together with my class mate M R Santhan. We shared the above tasks and finished the project.

Scope of the solution

- First, we need to understand the given problem.
- After understanding the problem, we have to collect correct data related to the problem.
- Once we have data in our hand, we need label the reviews and preprocess the data.
- After getting enough understanding of the data then extract the required feature from it.
- Then build a machine learning pipeline which includes count vectorizer, IDF and model.
- Evaluate the model using test data.
- Conclude the results based on the model selected.

Challenges

The main challenge that we faced is that each model took around 30 minutes to run. As we know that modelling is done in an iterative approach. First, we create a model and then we go on tuning the model until we get a good accuracy. So, we spend lot of time to wait for a model run. This can be avoided by connecting to GPU but due to lack of free GPU we used our own CPU for processing.

Data Description

The data comes from the website "Amazon product data" managed by Dr. Julian McAuley from UCSD. We choose the smaller subset of the customer review data from the Kindle store of Amazon.com (link to download the dataset). The data is in the JSON format, which contains 982,619 reviews and metadata spanning May 1996 - July 2014.

The data from the dataset had following variables

#	Column	Type	Description
1	Asin	String	ID of the product
2	Helpful	Array	helpfulness rating of the review, e.g., 2/3

3	Overall	Float	The rating of the customer about the product.
4	Reviewtext	String	The original review of the customer about the product
5	reviewTime	Datetime	The date and time at which the customer provided the review
6	reviewerID	String	A unique ID given to each reviewer.
7	reviewerName	String	Name of the reviewer.
8	summary	String	Summary of the review in few words
9	unixReviewTime	Integer	Unix time is a system for describing a point in time. It is the number of seconds that have elapsed since the Unix epoch

Reviews with overall rating of 1, 2, or 3 are labelled as negative (1), and reviews with overall rating of 4 or 5 are labelled as positive (0). Thus, number of positive and negative reviews are as follows in the original dataset:

- positive: 829,277 reviews (84.4%)
- negative: 153,342 reviews (15.6%)

There is lot of data and we are trying to take a balanced sample of the data.

Data Pre-processing

In pre-processing we labelled the data based on costumer rating. If the rating is above 4, then we label it as 0 i.e., positive class and ant record having a rating less than 4 is labelled as 1 i.e., negative class.

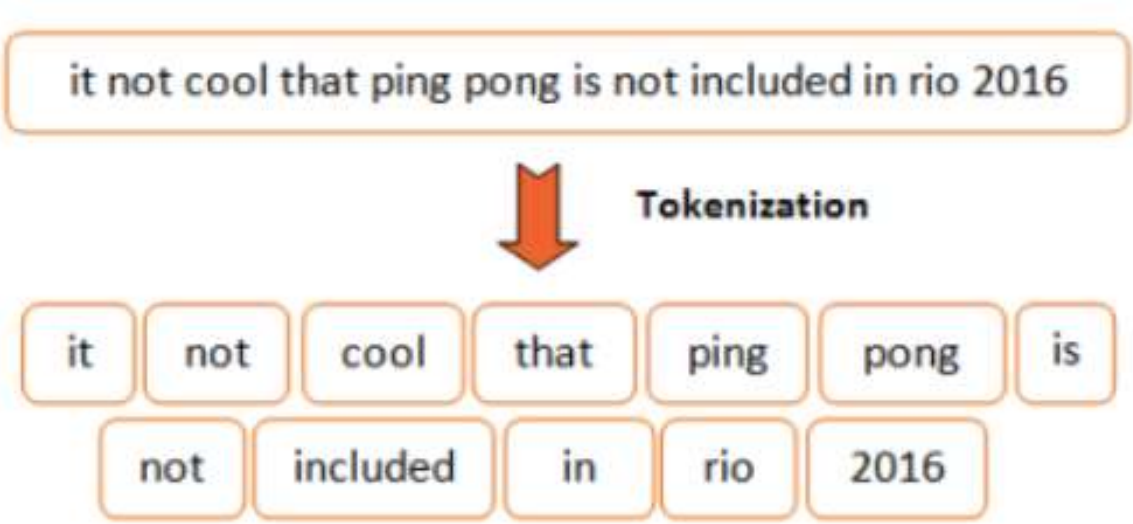
After labelling the data, we try to take a sample of the data as the data is too large. We try to take a balanced dataset.

- positive: 41,634 reviews (51.7%)
- negative: 38,542 reviews (48.3%)

we tried to expand the short forms in the text. For example, can't is replaced with cannot and n't is replaced with not.

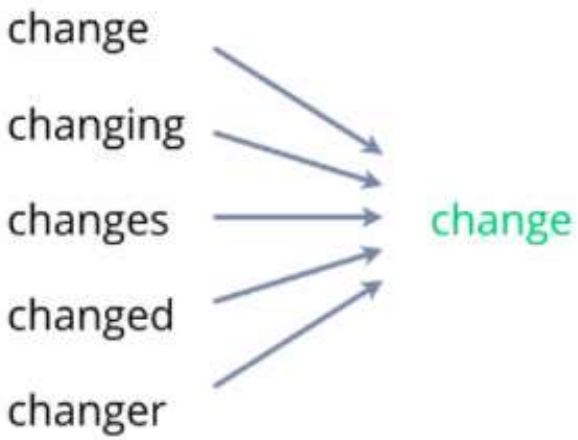
Then we went on replacing all the punctuation marks with blank white space and changed all the text into lowercase.

Then we performed word tokenization. Word tokenization is a process in which we split a large sentence into a list of words.



As shown in above figure the sentence is divided into a list of words.

Then we went on performing word lemmatization. Word lemmatization is a process in which we find the rood word of a word. For example,



In the above picture, we can see that the words change, changing, changes, changed and changer has the same root word change. This process is known as word lemmatization

After all these we performed negation handling. We maintain a variable to know the state of the sentence. If we encounter a word like not, no etc, we set the state variable. If we get a punctuation mark, we mark the sentence as negation statement and if we get a negation word again, we set the state variable again.

We combine all the various pre-processing functions into a one function in our code and the code looks like

```
def preprocessing(text):
    line = html.unescape(text)
    line = line.replace("can't", 'can not')
    line = line.replace("n't", " not")
    pad_punct = str.maketrans({key: " {0} ".format(key) for key in string.punctuation})
    line = line.translate(pad_punct)
    line = line.lower()
    line = line.split()
    lemmatizer = nltk.WordNetLemmatizer()
    line = [lemmatizer.lemmatize(t) for t in line]

    # Negation handling
    tokens = []
    negated = False
    for t in line:
        if t in ['not', 'no']:
            negated = not negated
        elif t in string.punctuation or not t.isalpha():
            negated = False
        else:
            tokens.append('not_' + t if negated else t)
```

```
invalidChars = str(string.punctuation.replace("_", ""))
bi_tokens = list(nltk.bigrams(line))
bi_tokens = list(map('_'.join, bi_tokens))
bi_tokens = [i for i in bi_tokens if all(j not in invalidChars for j in i)]
tri_tokens = list(nltk.trigrams(line))
tri_tokens = list(map('_'.join, tri_tokens))
tri_tokens = [i for i in tri_tokens if all(j not in invalidChars for j in i)]
tokens = tokens + bi_tokens + tri_tokens
return tokens
```

In the above function we can see that we have assembled all the techniques into a one function names as preprocessing.

Now let's see how the output of the text looks like. Let's pass a random review and see the out put

```
review = preprocessing("I think this book is not good. It is full of typos and factual errors that I can't ignore.")
print(review)
```

```
['i', 'think', 'this', 'book', 'is', 'not_good', 'it', 'is', 'full', 'of', 'typo', 'and', 'factual', 'error', 'that', 'i', 'ca
n', 'not_ignore', 'i_think', 'think_this', 'this_book', 'book_is', 'is_not', 'not_good', 'it_is', 'is_full', 'full_of', 'of_typ
o', 'typo_and', 'and_factual', 'factual_error', 'error_that', 'that_i', 'i_can', 'can_not', 'not_ignore', 'i_think_this', 'thin
k_this_book', 'this_book_is', 'book_is_not', 'is_not_good', 'it_is_full', 'is_full_of', 'full_of_typo', 'of_typo_and', 'typo_an
d_factual', 'and_factual_error', 'factual_error_that', 'error_that_i', 'that_i_can', 'i_can_not', 'can_not_ignore']
```

We can observe that we have passed a random review to the preprocess function and we got a list of modified words as output. We can observe the following changes to the sentence.

- All the words are in lower case
- Negation handling is done
- Words like can't are changed to cannot

Design of solution

As we did changes to a sample sentence, let's change the reviews in the whole dataset.

We don’t need all the variables for our modelling. We just need the list of words and the label. So, we create a new data frame and its looks as below.

```
+-----+--------------------+--------------------+
|label|                text|              tokens|
+-----+--------------------+--------------------+
|    0|I am not for sure...|[i, am, not_for, ...|
|    0|This is yet anoth...|[this, is, yet, a...|
|    0|I almost didn't g...|[i, almost, did, ...|
+-----+--------------------+--------------------+
```

After all the pre-processing, we split the data. 70% to the train and 30% to test set. Then I, created a naive bayes model and trained it using 70% of the data and used other 30% percent to test the model.

Modelling

Naive bayes

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 a single algorithm for training such classifiers, but a family of algorithms based on a common principle. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

In this method we have created a pipeline. The various different stages in the pipe line are

- Count vectorizer: It is used to transform given text into a vector using the frequency of a word.
- IDF: It is a method which gives numerical weightage to a word and says how important.
- Naive bayes model: It is a classification model.

The code looks like shown below

```
c_vec = CountVectorizer(inputCol='tokens', outputCol='c_vec', minDF=5.0)

idf = IDF(inputCol="c_vec", outputCol="features")
```

```
nb = NaiveBayes()


nb_pipeline = Pipeline(stages=[c_vec, idf, nb])


nb_model = nb_pipeline.fit(df_train)
nb_test_pred = nb_model.transform(df_test)
nb_test_pred.show(3)
```

we can see that we created 3 variables for count vectorization, IDF and model. Then we pass all these to a pipeline as different stages. Then we train pipeline with the train data.

While testing the pipeline with the test data, we got an accuracy of 86%.

```
Accuracy of the Naive Bayes model: 0.8600814494680851
```

Then I, tried the same naive bayes model with cross validation to see whether we can get a better model.

```
nb_paramGrid = (ParamGridBuilder()
            .addGrid(c_vec.minDF, [3.0, 5.0, 7.0, 10.0, 15.0])
            .addGrid(nb.smoothing, [0.1, 0.5, 1.0])
            .build())
nb_cv = CrossValidator(estimator= nb_pipeline, estimatorParamMaps= nb_paramGrid,
evaluator= nb_Acc_, numFolds= 4)
nb_cv_model = nb_cv.fit(df_train)
```

But we did not see a great improvement in the accuracy. The accuracy with cross validation is

```
Accuracy of the Naive Bayes cross validation model: 0.8609541223404256
```

Previously accuracy was 86.008% and with cross validation it is 86.09%

Results and Analysis

I used the above model to predict the reviews that the model had never seen which is known as out of box testing.

We have sent new reviews to test the model. Sample review is given below

"*I loved this book. The author made me think long and hard about our split government and the trials of these three young women in Paris during the early 1940s when they fell under Nazi rule. I know much of this history but she brought it to life and placed me on the scene. Friends of family who underwent a camp don't talk about it much but to read this, though I've seen films regarding it, she really brought it to life in a way that you can't ignore what was happening*"

By reading the review, we feel it as a positive review. Now let's see what our model predicts. Along with this review we also sent 4 more reviews in which 2 of them are positive and 2 of them are negative. The prediction of the model was as below

```
+--------------------+----------+
|                text|prediction|
+--------------------+----------+
|I loved this book...|       0.0|
|I am never buying...|       1.0|
|I enjoyed reading...|       0.0|
|The book has too ...|       1.0|
|She just passed a...|       0.0|
+--------------------+----------+
```

As we already discussed, a positive review is indicated by class 0 and a negative by class 1. For me as far as the predictions, the model performed very well.

Future works

There is lot of scope for the idea. We can go further and classify each negative review further. The company can find out the reasons for the dissatisfaction and can work in that direction to resolve the issues so, that the same might not repeat again.

Conclusion

While doing this project, I have learnt many things. The main learning was that how to take a balanced sample when we can't use the whole data as it may take time to process the whole data. Apart from the technical learnings, I learnt how to communicate and work with a partner, which is a very important learning, because in future we always work in group.

References

http://jmcauley.ucsd.edu/data/amazon/

http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Kindle_Store_5.json.gz