SENTIMENT CLASSFICATION ON PRODCUT REVIEWS

Author 1	(Family name) Kulkarni	(Given names) **Prashantkumar**
Author 2	(Family name) Hanumanthrao Patil	(Given names) **Vikram**

June 02, 2019

Table of Contents

1 Introduction *3*

1.1 Project Description **3**

1.2 Goal of the project **3**

1.3 Work distribution in group **4**

2 Pre-processing and feature generation *5*

2.1 Pre-processing **5**

2.2 EDA **6**

2.2.1: Distribution of 5 labels in the dataset 6
2.2.2: Text length 6
2.2.3: Punctuation 7
2.2.4: Most words for label-1: strong negative sentiment 8
2.2.4: Most words for label-2: weak negative sentiment 8
2.2.5: Most words for label-3: neutral sentiment 9
2.2.6: Most words for label-4: weak positive sentiment 9
2.2.7: Most words for label-7: strong positive sentiment 10

2.3 Feature generation **10**

2.3.1 Description of TFIDF and features: 10
2.3.2 Feature generation steps 11

3. Models *13*

3.1 Model 1 – Linear SVC (Support Vector Classification) **13**

3.2 Model 2 (Multinomial Naive Bayes classifier) **14**

3.3 Model 3 – Logistic regression(Multinomial) and Chi-square (dimension reduction) **14**

3.4 Discussion of model differences **15**

3.4.1 Part-1: TFIDF + Model: 15
3.4.2 Part-2: Chi-square Feature selection(dimension reduction) + TFIDF + Model: 16

4 Experiment setups *18*

5 Experimental results *19*

6. Conclusion *20*

References *21*

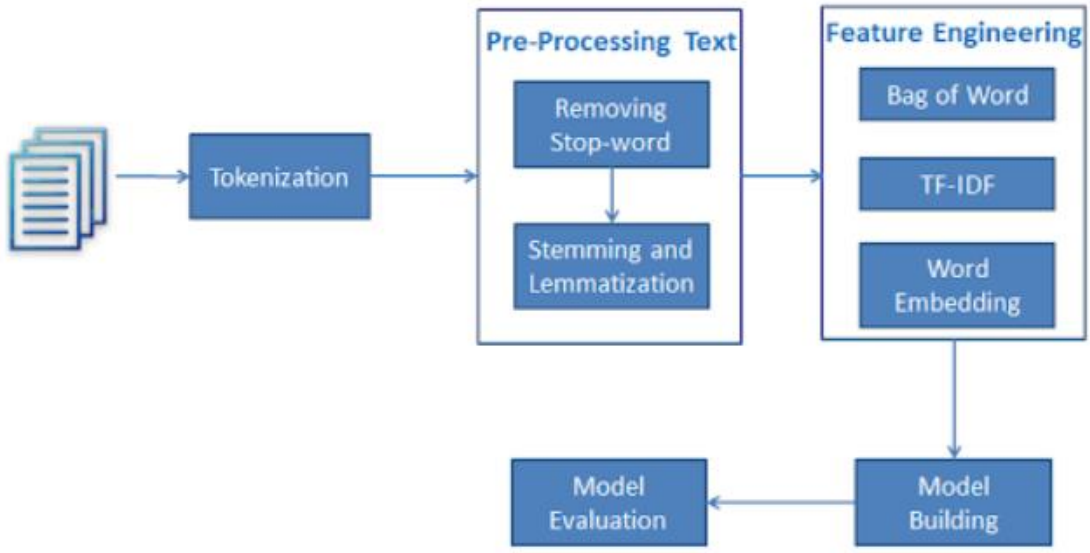
1 Introduction

1.1 Project Description

Sentiment analysis is an automatic process of finding the opinion of a subject on social media through voice or text. In the recent era, digitization has scaled at great extent, almost every product is soled online, this in turn generates massive reviews on products by customer. These review or sentiments can be understood and used for analytics to improve the customer experience. Yelp is one such platform wherein, customers react to various products such as restaurant reviews and so on. This project mainly concentrates on developing an automatic sentiment classifier, wherein based on the customer review, the project will classify the review as either strong positive, weak positive, neutral, weak negative or strong negative.

1.2 Goal of the project

Supervised technique such as text classification, is implemented in the project using python to identify the sentiments of customer feedback extracted from yelp platform. For example, a review such as, *"Our delivery was cancelled suddenly, and no one is answering the phone. Shame",* will be classified as strong negative.



1 Approach for sentiment analysis, retrieved from [1]

The figure1 explains the approach used for sentiment classification in this project [1]. All the reviews are tokenized and processed. Once, pre-processing is completed, feature engineering is performed to select best features. On the selected features, supervised model is trained and evaluated.

1.3 Work distribution in group

As a team we both didn't restrict to one area. We both performed all the tasks such as data pre-processing, model building, testing and evaluation and report. Initially, Vikram started with data wrangling and pre-processing, and model building whereas Prashanth tried different models to improve the accuracy. With above approach, we learnt our mistake that data pre-processing had to be improved to increase the accuracy. Hence, we both tried different data pre-processing techniques such as use of stop-words or not, use of normalization in data or not. Nearly 5 data-preprocessing techniques were tried and below is the screenshot(Figure 2) which explains various techniques we divided with each other.

```
def tokenize_stem_lower_customstopwords_twoletterremove(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = re.split('\W+', text) #tokenizing
    tokens=[word for word in tokens if word.isalpha()] #Removing digits
    tokens = [wn.lemmatize(word) for word in tokens]
    tokens = [word for word in tokens if word not in custom_stopwords]
    text = " ".join([word for word in tokens if len(word)>2])
    return text

def tokenize_stem_upper_customstopwords(text):
    text = "".join([word for word in text if word not in string.punctuation])
    tokens = re.split('\W+', text) #tokenizing
    tokens=[word for word in tokens if word.isalpha()] #Removing digits
    tokens = [wn.lemmatize(word) for word in tokens]
    text = " ".join([word for word in tokens if word not in custom_stopwords])
    return text


def remove_punc_stopword(text):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    remove_punc = [word for word in text if word not in string.punctuation]
    remove_punc = ''.join(remove_punc)
    return [word.lower() for word in remove_punc.split() if word.lower() not in stopwords]
```

2 Different pre-processing techniques used

In final, we selected the below pre-processing technique, figure 3.

```
def pre_process(text):
    """
    Takes in a string of text, then performs the following:
    1. converts to lower
    2. Splits the sentence into tokens
    3. Decontract the words in the sentence. For example: "won't" --> "will not"
    4. Lemmatization, reduces words to their base word
    5. Returns the sentence of the cleaned text
    """
    text = "".join([word.lower() for word in text])
    tokens = text.split(" ")
    tokens = [appos[word] if word in appos else word for word in tokens]
    text = " ".join([wn.lemmatize(word) for word in tokens])
    return text
```

3 Final pre-processing technique

At next steps, we both tried different model techniques to understand which gives the best accuracy. Vikram tried Random forest, SVM linear, AdaBoostClassifier. Prashanth tried logistic, MultinomialNB and LinearDiscriminantAnalysis.

Report wise, we equally divided the sections.

2 Pre-processing and feature generation

2.1 Pre-processing

To understand the data, we have divided this section into two:

1) Adding punctuation percentage and text length of each review for analysis
2) Perform pre-processing techniques

Punctuation percentage and text length of each review for analysis:

```
#-------------------------------------------------------------
# Adding each review's text length and punctuation percentage
#-------------------------------------------------------------


def count_punct(text):
    """
    Takes in a string of text, then performs the following:
    1. sums the punctuation count
    2. return the percentage

    """
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(" ")), 3)*100



df['body_len'] = df['text'].apply(lambda x: len(x) - x.count(" "))
df['punct%'] = df['text'].apply(lambda x: count_punct(x))

#-------------------------------------------------------------------------------
```

4 punctuation count and body length

In the above figure 4, we are determining the punctuation percentage and total body length of each review.

Pre-processing step:

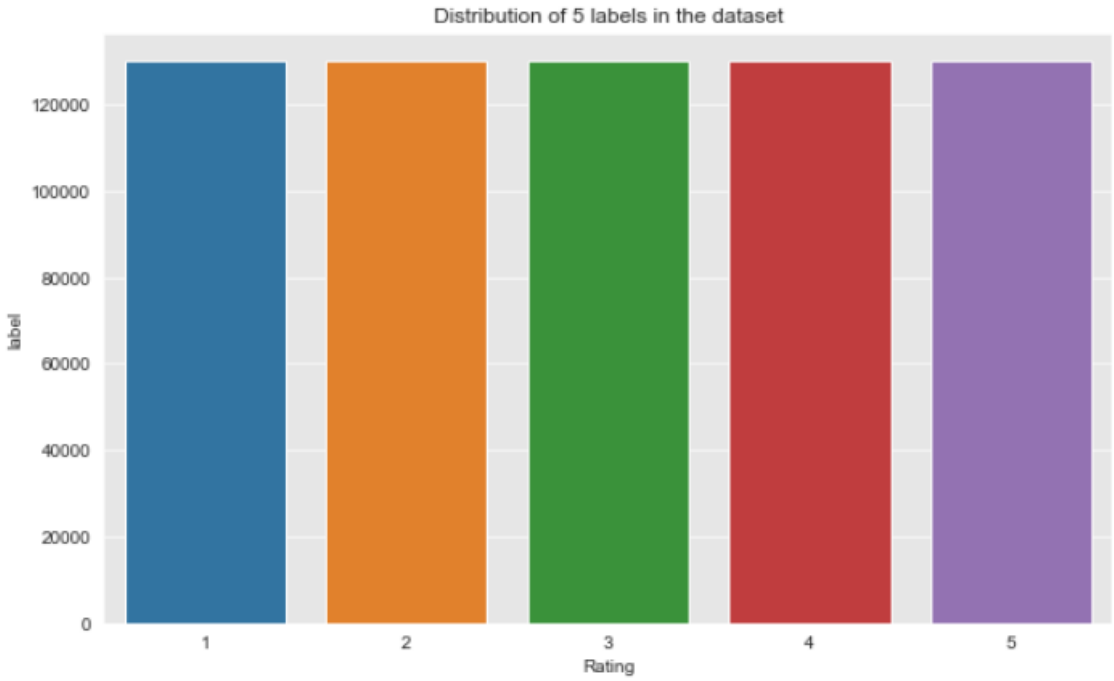
```
def pre_process(text):
    """
    Takes in a string of text, then performs the following:
    1. converts to lower
    2. Splits the sentence into tokens
    3. Decontract the words in the sentence. For example: "won't" --> "will not"
    4. Lemmatization, reduces words to their base word
    5. Returns the sentence of the cleaned text
    """
    text = "".join([word.lower() for word in text])
    tokens = text.split(" ")
    tokens = [appos[word] if word in appos else word for word in tokens]
    text = " ".join([wn.lemmatize(word) for word in tokens])
    return text
```

5 pre-processing step

In the above figure 5 , we are performing data pre-processing such as extracting all lowered tokens, de-contracting the tokens with lemmatize and joining them. The function returns cleaned sentence. This technique is needed to clear the noise from the actual data.

2.2 EDA

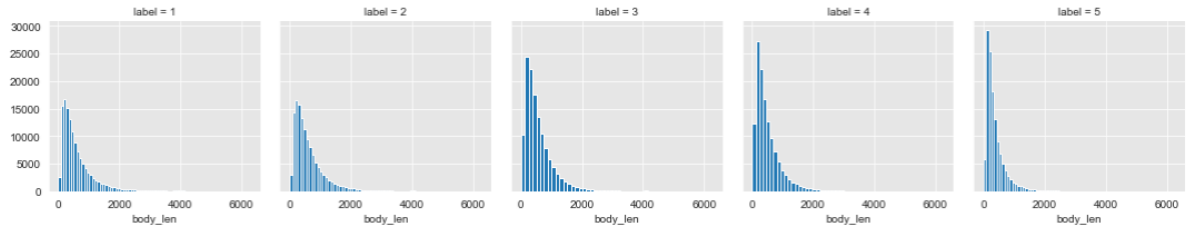
2.2.1: Distribution of 5 labels in the dataset



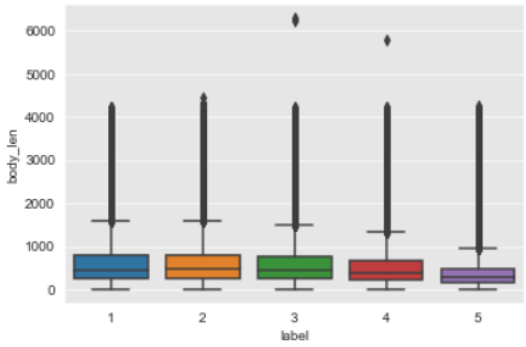
6 label distribution

From above figure 6, we can confirm that each label has equal number of data.

2.2.2: Text length



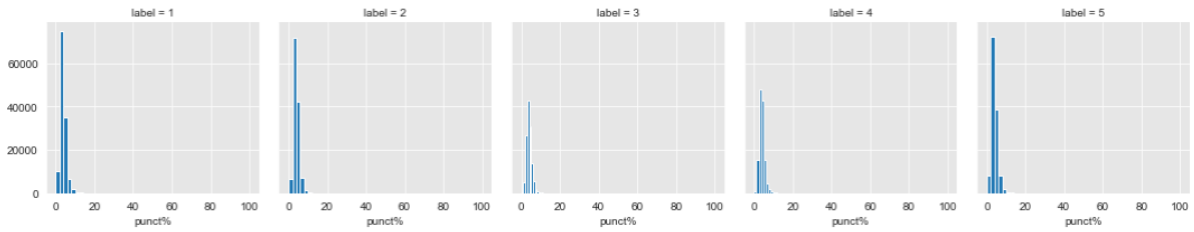
7 text length



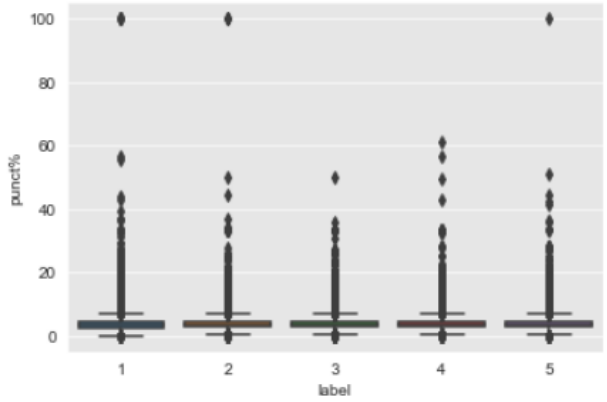
8 box plot of text length

From the plots figure 7 and 8, looks like the label 4 and 5 have much longer text, but there are many outliers (which can be seen as points above the boxes). Because of this, maybe text length won't be such a useful feature to consider after all.

2.2.3: Punctuation



9 punctuation distribution



10 punctuation box plot

From the plots figure 9 and 10, looks like label 3 has lowest punctuations, whereas label 1 and label 5 have almost similar pattern.

2.2.4: Most words for label-1: strong negative sentiment

11 label-1: most frequent words

From figure11, we can understand that for label1: strong negative sentiment words are customer service, will, never and etc.

2.2.4: Most words for label-2: weak negative sentiment

12 *label-2: most frequent words*

From figure12, we can understand that for label2: weak negative sentiment words are service, food, ok, instead, left, two star and so on.

2.2.5: Most words for label-3: neutral sentiment

13 label3: most frequent words

From figure13, we can understand that for label3: neutral sentiment words are good, service, food, seem, well, pretty and so on.

2.2.6: Most words for label-4: weak positive sentiment

14 label4: most frequent words

From figure14, we can understand that for label4: weak positive sentiment words are good, food, great, love, great place and so on.

2.2.7: Most words for label-7: strong positive sentiment

15 label5: most frequent words

From figure15, we can understand that for label4: strong positive sentiment words are highly recommend, love place, love, definitely, great food and so on.

2.3 Feature generation

The technique used to generate the features in our project is TFIDF.

2.3.1 Description of TFIDF and features:

It's an acronym for "Term frequency- Inverse document frequency". The output of this method contains the scores assigned to each word or token from pre-processing stage. The term frequency is how frequent the token appeared in each review or document; whereas, Inverse document frequency measures the amount of information of a word across the document. Hence, the value of each token is a combination of TF and IDF. Basically, this step highlights the interesting words but not across documents.[2]

In the below figure 16, let's take 3 documents for example, TFIDF generates below features such as I, love, dogs…passion. And for each feature, a value is generated against each document.

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	**0.48**	0.18							
Doc 2	0.18		0.18	**0.48**	0.18	0.18				
Doc 3					0.18	0.18	**0.48**	**0.95**	**0.48**	**0.48**

16 tfidf example

These features are normalised value between 0 and 1 and be directly used for machine learning.

2.3.2 Feature generation steps

We have divided this task into two

2.3.2.1 Pre-processing steps

- Convert all texts to lowercase
- Splits the sentence into tokens
- Decontract the words. For example: "won't" --> "will not"
- Lemmatization, reduces words to their base word
- Join all the words and returns the sentence of the cleaned text

2.3.2.1 Feature generation

We have divided this approach into two parts:

Part-1: Using TFIDF:

```
from sklearn.feature_extraction.text import TfidfVectorizer

#define vectorizer parameters
tfidf_vectorizer = TfidfVectorizer(max_df=0.99,min_df=0.01,
                                   use_idf=True, ngram_range=(1,3))

tfidf_matrix = tfidf_vectorizer.fit_transform(df['text'])
```

17 TFIDF vectorizer

In the above figure 17, we define that the. max_df select words that occur in a maximum of 99% of the documents. Too common words are not required for the classification. Similarly, min-df is set to 1%, tokens that occur in a minimum of 1% of documents

```
print(tfidf_matrix.shape)
print(tfidf_vectorizer.get_feature_names())
```

```
(65000, 2155)

['00', '10', '10 minute', '100', '11', '12', '15', '15 minute', '20', '20 minute', '25', '30', '30 minute', '40', '45',
'50', '99', 'able', 'able to', 'about', 'about it', 'about the', 'about this', 'about this place', 'above', 'absolutely',
'across', 'across the', 'actual', 'actually', 'add', 'added', 'after', 'after the', 'afternoon', 'again', 'ago', 'ahead',
'air', 'all', 'all in', 'all in all', 'all of', 'all of the', 'all over', 'all the', 'almost', 'alone', 'along', 'along w
ith', 'already', 'also', 'also had', 'also the', 'although', 'always', 'am', 'am in', 'am not', 'am not sure', 'amazing',
'ambiance', 'amount', 'amount of', 'an', 'an appointment', 'an hour', 'and', 'and all', 'and am', 'and asked', 'and chees
e', 'and could', 'and decided', 'and did', 'and did not', 'and do', 'and do not', 'and even', 'and food', 'and for', 'and
```

18 tfidf vocab

From the above figure 18, total documents selected is 65000 and from those documents 2155 features are selected, which is size of vocab.

Part-2: Feature selection using chi-square test:

5.1- Dimension reduction technique (Chi-square)

```
#dimension reduction using chi-square

x_train, x_validation, y_train, y_validation = train_test_split(df['text'], df['label'], test_size=.02)

tvec = TfidfVectorizer(max_features=100000,ngram_range=(1, 3))
x_train_tfidf = tvec.fit_transform(x_train)
x_validation_tfidf = tvec.transform(x_validation)

#reference[2]
```

```
from sklearn.feature_selection import SelectKBest, chi2
ch2_result = []
for n in np.arange(5000,100000,5000):
    ch2 = SelectKBest(chi2, k=n)
    x_train_chi2_selected = ch2.fit_transform(x_train_tfidf, y_train)
    x_validation_chi2_selected = ch2.transform(x_validation_tfidf)
    clf = LogisticRegression()
    clf.fit(x_train_chi2_selected, y_train)
    pred = clf.score(x_validation_chi2_selected, y_validation)
    ch2_result.append(pred)
    print("chi2 feature selection evaluation of {} features and accuracy is {}".format(n,pred))
```

```
chi2 feature selection evaluation of 5000 features and accuracy is 0.6322307692307693
chi2 feature selection evaluation of 10000 features and accuracy is 0.6407692307692308
chi2 feature selection evaluation of 15000 features and accuracy is 0.6416923076923077
chi2 feature selection evaluation of 20000 features and accuracy is 0.6447692307692308
chi2 feature selection evaluation of 25000 features and accuracy is 0.646
chi2 feature selection evaluation of 30000 features and accuracy is 0.643
chi2 feature selection evaluation of 35000 features and accuracy is 0.6454615384615384
chi2 feature selection evaluation of 40000 features and accuracy is 0.6472307692307693
chi2 feature selection evaluation of 45000 features and accuracy is 0.6446153846153846
chi2 feature selection evaluation of 50000 features and accuracy is 0.646
```

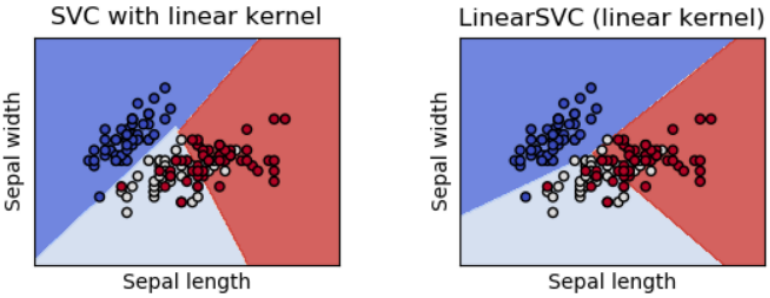
19 Chi-square feature selection

In the above figure(19), we have used chi-square feature selection(dimension reduction technique) to select best features. According to the above figure, total 650000 documents were passed and out of max features 100000, best 40000 features gave the best accuracy according to above figure.

3. Models

3.1 Model 1 – Linear SVC (Support Vector Classification)

We choose Linear SVC over SVM(kernel=linear). The reason is the estimators for SVC is liblinear and not libsvm, liblinear has the feature of applying penalties on the intercepts, thus runs faster [4]



20 difference between SVM (kernel=linear) and LinearSVC

Figure 20 explains the difference between SVM(kernel=linear) and LinearSVC. Support vector algorithm works based on the hyperplane, which segregates the data into required classification. Best hyper plane is selected based on the highest distance between the data points and the hyperplane. [4].

3.2 Model 2 (Multinomial Naive Bayes classifier)

The multinomial Naive Bayes classifier is reasonable for classification with discrete features. Fractional counts such as tf-idf can be used as multinomial factors. Multinomial Naive Bayes order calculation will in general be a gauge answer for sentiment task. The fundamental thought of Naive Bayes strategy is to get the probabilities of classes allocated to the texts by utilizing the joint probabilities of words and classes.[7]

Assumptions:

- For training our Multinomial NB model, we are using TFIDF vectorizer.
- MultinomialNB is used for naïve
- All the parameters of classifier is set to default

Advantages:

1. Easy to implement, simple and is fast model
2. Provides better results even with less training data
3. It can handle both continuous and discrete data
4. Insensitive to the irrelevant features
5. Can be used for both two class or multi class classifications

Disadvantages:

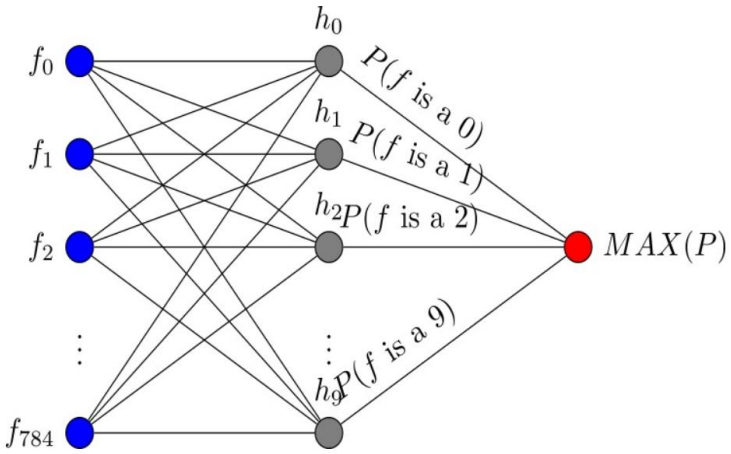
1. Strong feature independence makes it disadvantageous as the important features or the distinguishing features will lose their value.[8]
2. Naive Bayes is known as a bad estimator, so the probability outputs are not to be taken with too much importance.
3. If the category which was observed in training dataset is not present in the test dataset then that category will be assigned with probability zero, which will nullify all other probabilities resulting in no prediction.

3.3 Model 3 – Logistic regression(Multinomial) and Chi-square (dimension reduction)

We have divided this task into two steps:

1) Feature selection based on chi-square test (Dimension reduction technique)
    - Since, we had huge datasets, we didn't go for backward or forward feature selection and instead we choose Chi-square test, its ranks the features based on prearranged numerical functions that measure the importance of the terms
    - It takes less time to execute on large datasets.
    - It measures the lack of independence between a feature (in this case, one term within a review) and class (whether the labels are 1 or 2 or 3 or 4 or 5).
    - It is a supervised technique.[5]

2) Applying Logistic regression(Multinomial)model on the selected features:



21 Multinomial logistic regression, source [6]

The logistic regression using sigmoid function is for a single "neuron" [6] sued for binary classification. We can also create multiple neurons to accomplish the task, this technique is known as multinomial, wherein multi neurons are trained to accomplish the goal. Figure 21, explains the visualization. It's also known as softmax regression.

3.4 Discussion of model differences

The discussion can be divied into two parts:

3.4.1 Part-1: TFIDF + Model:

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

#define vectorizer parameters
tfidf_vectorizer = TfidfVectorizer(max_df=0.99,min_df=0.01,
                                   use_idf=True, ngram_range=(1,2))

tfidf_matrix = tfidf_vectorizer.fit_transform(df['text'])
```

```
X_features2=pd.DataFrame(tfidf_matrix.toarray(),columns=tfidf_vectorizer.get_feature_names())
X_features2.head()
```

22 Feature creation using TFIDF

For the first two models (LinearSVC and Multinomial Naive Bayes), initially we selected the features by using TFIDF method, refer figure 22.

Next we divided the features into training and validation sets. LinearSVC gave accuracy of 0.63 and whereas, Multinomial Naïve Bayes gave accuracy of 0.59. (refer below Figure 23)

LinearSVC

```
from sklearn.svm import LinearSVC
clf = LinearSVC()
clf.fit(x_train, y_train)
score = clf.score(x_test, y_test)
score
```

```
18]: 0.6335384615384615
```

cross-validation

```
k_fold = KFold(n_splits=3)
cross_val_score(svclassifier_linear, x_train, y_train, cv=k_fold, scoring='accuracy', n_jobs=-1)
```

```
19]: array([0.62377198, 0.62290836, 0.62272468])
```

Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(x_train, y_train)
score = clf.score(x_test, y_test)
score
```

```
16]: 0.5955384615384616
```

cross-validation

```
k_fold = KFold(n_splits=3)
cross_val_score(clf, x_test, y_train, cv=k_fold, scoring='accuracy', n_jobs=-1)
```

```
17]: array([0.59843925, 0.59567754, 0.59774505])
```

23 LinearSVC and Multinomial Bayes model

3.4.2 Part-2: Chi-square Feature selection(dimension reduction) + TFIDF + Model:

Since these two techniques didn't give better accuracy, we tried different approach in selecting the features.

Using chi-squared test (dimension reduction technique), firstly we divided the actual pre-processed data into training and validation. Then using tfidf transformed the x_train x_test. With varying size of k, chi-square test was executed. This step gave the best result for us, refer figure 24 below. For 40,000 features, best accuracy was observed.

5.1- Dimension reduction technique (Chi-square)

```
#dimension reduction using chi-square

x_train, x_validation, y_train, y_validation = train_test_split(df['text'], df['label'], test_size=.02)

tvec = TfidfVectorizer(max_features=100000,ngram_range=(1, 3))
x_train_tfidf = tvec.fit_transform(x_train)
x_validation_tfidf = tvec.transform(x_validation)

#reference[2]
```

```
from sklearn.feature_selection import SelectKBest, chi2
ch2_result = []
for n in np.arange(5000,100000,5000):
    ch2 = SelectKBest(chi2, k=n)
    x_train_chi2_selected = ch2.fit_transform(x_train_tfidf, y_train)
    x_validation_chi2_selected = ch2.transform(x_validation_tfidf)
    clf = LogisticRegression()
    clf.fit(x_train_chi2_selected, y_train)
    pred = clf.score(x_validation_chi2_selected, y_validation)
    ch2_result.append(pred)
    print("chi2 feature selection evaluation of {} features and accuracy is {}".format(n,pred))
```

```
chi2 feature selection evaluation of 5000 features and accuracy is 0.6322307692307693
chi2 feature selection evaluation of 10000 features and accuracy is 0.6407692307692308
chi2 feature selection evaluation of 15000 features and accuracy is 0.6416923076923077
chi2 feature selection evaluation of 20000 features and accuracy is 0.6447692307692308
chi2 feature selection evaluation of 25000 features and accuracy is 0.646
chi2 feature selection evaluation of 30000 features and accuracy is 0.643
chi2 feature selection evaluation of 35000 features and accuracy is 0.6454615384615384
chi2 feature selection evaluation of 40000 features and accuracy is 0.6472307692307693
chi2 feature selection evaluation of 45000 features and accuracy is 0.6446153846153846
chi2 feature selection evaluation of 50000 features and accuracy is 0.646
```

24 chi-square

Once the feature size was selected, with that feature size, we trained final model using Multinomial logistic regression and validated it using cross-validation. Since the result of cross-validation was accurate, we choose this model as the final one. (Refer figure 25)

Multinomial logistic regression

```
ch = SelectKBest(chi2, k=40000)
x_train_feature_selected=ch.fit_transform(x_train_tfidf, y_train)
x_test_chi_selected = ch.transform(x_validation_tfidf)

from sklearn import linear_model

clf = linear_model.LogisticRegression(multi_class='multinomial',solver = 'newton-cg')
clf.fit(x_train_feature_selected, y_train)
score = clf.score(x_test_chi_selected, y_validation)
score
```

```
.2]: 0.6512307692307693
```

```
from sklearn.model_selection import KFold, cross_val_score

#rf = RandomForestClassifier(n_jobs=-1)
k_fold = KFold(n_splits=3)
cross_val_score(clf, x_train_chi2_selected, y_train, cv=k_fold, scoring='accuracy', n_jobs=-1)
```

```
.3]: array([0.64903878, 0.64781734, 0.64900887])
```

25 Multinomial logistic regression model

4 Experiment setups

Generic settings applicable for all the models are below

- The entire train data is split into train and test with test size as 0.2
    - x_train, x_validation, y_train, y_validation = train_test_split(df['text'], df['label'], test_size=.02)
- TFIDF vectorizer is used to transform the features to count vectors, the maximum features is limited to 100000 and we have considered unigrams, bigrams and trigrams (ngram_range=(1,3)).
    - TfidfVectorizer(max_features=100000,ngram_range=(1, 3))
- ch = SelectKBest(chi2, k=55000)
    - For chi square test, the number of features with best chi square statitics k is set to 55000
- For calculating the accuracy, we are importing f1_score from sklearn.metrics package, hence when the predictions and the response data are sent an accuracy score will be displayed.
- For kfold cross validation we are using package sklearn.model_selection to import KFold, cross_val_score functions.

    k_fold = KFold(n_splits=3)
    cross_val_score(clf, x_train_feature_selected, y_train, cv=k_fold, scoring='accuracy', n_jobs=-1)

- Splits are three which means the number of folds is set to 3. In cross_val_score we are sending three parameters estimator, object to fit, target variable, scoring is set to accuracy, number of cpu's to be used is set to 1.

Model 1 – Multinomial Logistic Regression

- linear_model package is used for logistic regression. Following are the parameter setup for logistic regression.
- linear_model.LogisticRegression(multi_class='multinomial',solver = 'newton-cg')
- We are specifying multi_class as 'multinomial' since the classification classes are more than 2.
- Solver we are using is 'newton-cg', there are solvers like 'saga' which when tried yielded less accuracy hence we are sticking to newton-cg.
- All the other features are set to default.

Model 2 – LinearSVC

- In this model we are importing LinearSVC() from the package sklearn.svm.
- Following are the settings for the model. We are using the LinearSVC with all the default values, hence we are just calling the function as LinearSVC() into our model.

Model 3 – Multinomial Naïve Bayes

- Following package are imported.
- from sklearn.naive_bayes import MultinomialNB
- All the values of function MultinomialNB is set to default, hence we are directly calling the function into model.

5 Experimental results

Cross-Validated Classification Accuracy

Feature Set	Model	Accuracy
Feature Set 1	**Model 1** Linear SVC	0.62377198
	Model 2 Multinomial Naive Bayes	0.59843925
	Model 3 Chisquare + Logistic(Multinomial)	0.64903878
Feature Set 2	**Model 1** Linear SVC	0.62290836
	Model 2 Multinomial Naive Bayes	0.59567754
	Model 3 Chisquare + Logistic(Multinomial)	0.64781734
Feature Set 3	**Model 1** Linear SVC	0.62272468
	Model 2 Multinomial Naive Bayes	0.59774505
	Model 3 Chisquare + Logistic(Multinomial)	0.64900887

From above we can confirm that model-3(Chisquare + Logistic(Multinomial)) performed best among three

6. Conclusion

The optimal classifier was Logistic regression(Multinomial). This worked best because of the chi-square dimension reduction technique. Using this technique, best features which with no inter correlation is selected and used for the model.

To sum up, this project was us wide variety of tools and new concepts to learn and implement. We also explored deep neural networks to achieve the task, due to system capacity and time consumption we couldn't tune the LSTM model. On the other hand, we tried many pre-processing techniques to achieve the model, every step we identified best techniques.

Overall, this whole project gave us an experience to solve real world problem with Kaggle competition as the base. I believe, participating in the competition increased our spirits every day.
Thank you.

References

[1] https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk

[2] https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

[3] https://github.com/tthustla/twitter_sentiment_analysis_part8/blob/master/Capstone_part4-Copy6.ipynb

[4] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[5] https://valiancesolutions.com/variable-reduction-an-art-as-well-as-science/

[6] https://www.pugetsystems.com/labs/hpc/Machine-Learning-and-Data-Science-Multinomial-Multiclass-Logistic-Regression-1007/

[7] https://scikit learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[8] https://www.codershood.info/2019/01/14/naive-bayes-classifier-using-python-with-example/