In this programming assignment, we have to build a supervised text classifier i.e., sentiment classification with given dataset. The main goal is to classify the sentiment/label of user reviews into broadly two classes negative and positive reviews. **Accuracy** obtained on **Base Model**: With logistic regression and white space tokenizer:

Accuracy on train is: 0.9821038847664775
Accuracy on dev is: 0.777292576419214

2.1 Guided Feature Engineering

TD-IDF stands for term frequency and inverse document frequency. It is typically used in machine learning and information retrieval to measure the significance of text representations such as words and phrases in a group of documents. In another words it is used to measure how important the word is, in a given set of corpus (collection of all documents). In this assignment we are asked to build supervised text classifier so it is very important to understand the significance of each word in documents so that we can build a robust classifier. TF-IDF[1] evaluates the importance of each word in a document and scaling that word using its significance in corpus set.

$$\mathrm{TF}-\mathrm{IDF}(\mathrm{word},\mathrm{document},\mathrm{Corpus})=\mathrm{TF}(\mathrm{word},\mathrm{document})\cdot IDF(word,Corpus)$$

Term frequency is calculated using number of times the word has appeared in a single document divided by the total number of words in a document. **Inverse document frequency** of a word is calculated using total number of documents in a corpus divided by the number of documents in which particular word is found. In this question we are going to use sklearn library to implement TF-IDF

from sklearn.feature extraction.text import TfidfVectorizer

For the training of the model, we are going to use logistic regression:

from sklearn.linear_model import LogisticRegression

LogisticRegression(random_state=0, solver='lbfgs', max_iter=10000)

In this task TF-IDF takes a parameter n which represents n-gram feature length. We also have to vary regularizer parameter C in logistic regression. We have conducted several experiments using combinations of different values of C and n to generate results.

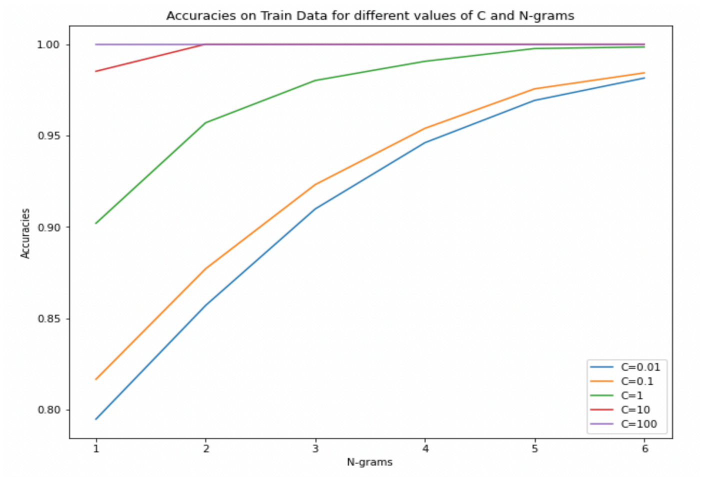
Now we must vary n and C: C varies from [0.01,0.1,1,10,100] and n varies for 1,2,3,4,5,6.

LogisticRegression(random_state=0, solver='lbfgs', max_iter=10000, **C**)

TfidfVectorizer(**ngram_range**)

After conducting experiments, we have obtained following accuracies on **train data**

	1-gram	**2-gram**	**3-gram**	**4-gram**	**5-gram**	**6-gram**
C = 0.01	0.79484941	0.85704932	0.90986469	0.94609341	0.96922741	0.98144915
C = 0.1	0.81667394	0.87712789	0.92317765	0.95395024	0.97555653	0.98428634
C = 1	0.90200786	0.95700567	0.98013968	0.99061545	0.9975993	0.99847228
C = 10	0.98515932	1.0	1.0	1.0	1.0	1.0
C = 100	1.0	1.0	1.0	1.0	1.0	1.0

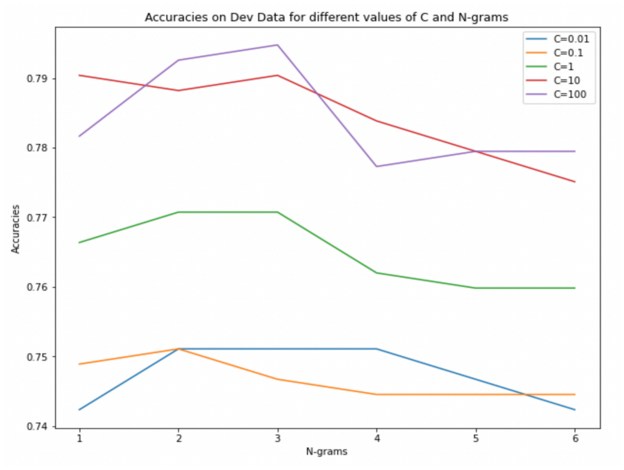


Accuracy Graph on train data for different values of C and n

As we change value of n from 1 to 6, accuracy varies for different vales of n. When the value of n is high it, classifier becomes specific but sparse whereas when n is small classifier becomes dense but general. Choosing n is really a tricky part. Selecting different values of "n" can change the way predictions are made by the classifier. As we choose high value of n the number of times we see n-gram words decreases thus specific but sparse whereas we when we decrease the value of n we tend to see more n-grams in the data thus dense but general. So, there is really a trade-off between high and small value of n. The optimal value of n obtained by conducting experiment is 3 which produces the best accuracy of 0.79475983. As n=3 falls between upper and lower range.

After building model, we have obtained following accuracies on **dev data**

	1-gram	**2-gram**	**3-gram**	**4-gram**	**5-gram**	**6-gram**
C = 0.01	0.74235808	0.7510917	0.7510917	0.7510917	0.74672489	0.74235808
C = 0.1	0.7489083	0.7510917	0.74672489	0.74454148	0.74454148	0.74454148
C = 1	0.76637555	0.77074236	0.77074236	0.76200873	0.75982533	0.75982533
C = 10	0.79039301	0.78820961	0.79039301	0.78384279	0.77947598	0.77510917
C = 100	0.78165939	0.79257642	0.79475983	0.77729258	0.77947598	0.77947598



Accuracy Graph on dev data for different values of C and n

Regularization parameter in logistic regression determines the amount of logistic regression applied to the model. The main aim of regularization parameter is to reduce overfitting of the model, which in turn lowers variance of the model with the selected param of logistic regression. By increasing regularization parameter, it lowers the magnitude of hypothesis parameter. Thus, regularization parameter reduces overfitting of the model, but it comes at the cost of increase in bias. Thus, selecting optimal value of regularization parameter is very important. In our experiment we have chosen different values of C ranging from 0.01, 0.1, 1, 10, 100. The most optimal value obtained for C from the experiments is 100 which produces the best accuracy of 0.79475983.

Thus, the most optimal value obtained for **n is 3 and C is 100** which gives the best accuracy of 0.79475983.

2.2 Independent Feature Engineering

In this section, I have tried different tokenization techniques other than white space tokenization by adding and removing feature. I have tried different approaches and evaluated results by plotting graphs and tables.

a.) **Stop Words Removal** and converting to lower case: The intuition behind stop words removal is that by removing stop words from the text we can consider important words for our model. Stop words[2] like "a, an, the, and, it, for, or…" does not contain much information, therefore we can remove these words. So, this stop words does not help in improving the accuracy of sentiment classifier. Stop words seems to be non-critical for the model training as it does not yield much information. Along with stop words removal all the words are converted to lower case in order to avoid redundance of similar words like "Hello" and "hello" both the words have same meaning, but model considers it as two different words because the case for both the words is different.

Stop words are present in large number in all documents, by removing stop words, we remove lower-level information from the documents, and only taking important words as features for the model.

Accuracy on train is: 0.9827586206896551
Accuracy on dev is: 0.7663755458515283

Original Sentence: " Hello! My name is John. This sentence is used for stop words removal check. Enjoy your day"

After Tokenization: ['hello', '!', 'name', 'john', '.', 'sentence', 'used', 'stop', 'words', 'removal', 'check', '.', 'enjoy', 'day', '.']

b.) **Stemming** is used to convert all the words to its derived words i.e., basically convert the words to its root words. For e.g., "playing", "plays", "played", "play" all these words are derived from the same word "play".

Stemming[3] is applied to change the word to its original form in order to process better in the logistic regression model. All the words are converted to lower case before stemming it.

Accuracy on train is: 0.9753382802269751
Accuracy on dev is: 0.7685589519650655

Original Sentence: “I was working day and night to get better results, by trying different techniques to improve model accuracy”

After Tokenization: ['i', 'wa', 'work', 'day', 'and', 'night', 'to', 'get', 'better', 'result', ',', 'by', 'tri', 'differ', 'techniqu', 'to', 'improv', 'model', 'accuraci']

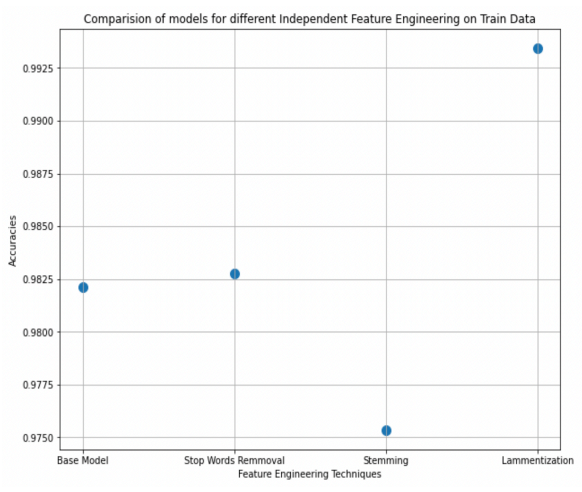
c.) **Lemmatization**: It is one of the most commonly used techniques for feature engineering in NLP. Lemmatization chops off the word to meaningful word by removing inflectional part which is at the end of the word. Lemmatization[4] follows vocabulary of the dictionary to change the word to by preserving its meaning unlike stemming. Intuition behind lemmatization is that it converts all the words to its basic lemma form, so if there are different form of same word it will convert it to base form and preserve its meaning, so this will help the model to remove redundant

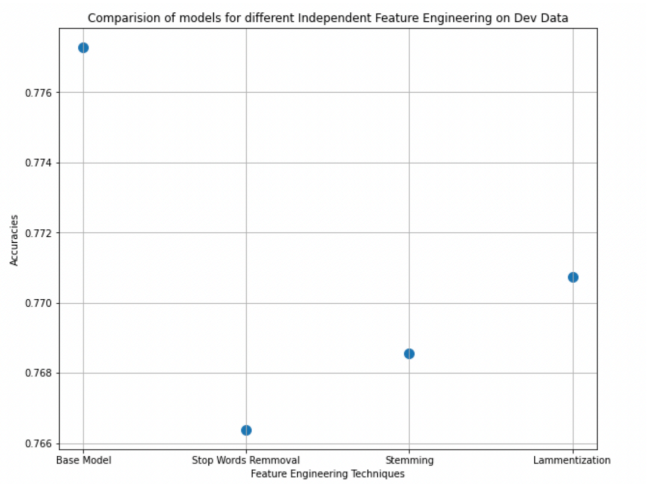
Accuracy on train is: 0.9934526407682235
Accuracy on dev is: 0.7707423580786026

Original Sentence: “I was working day and night to get better results, by trying different techniques to improve model accuracy”

After Tokenization: ['“i', 'wa', 'working', 'day', 'and', 'night', 'to', 'get', 'better', 'results,', 'by', 'trying', 'different', 'technique', 'to', 'improve', 'model', 'accuracy”']

	Accuracy on Train Data	**Accuracy on Dev Data**
Base Model	0.982104	0.777293
Stop Words Removal	0.982759	0.766376
Stemming	0.975338	0.768559
Lemmatization	0.993453	0.770742





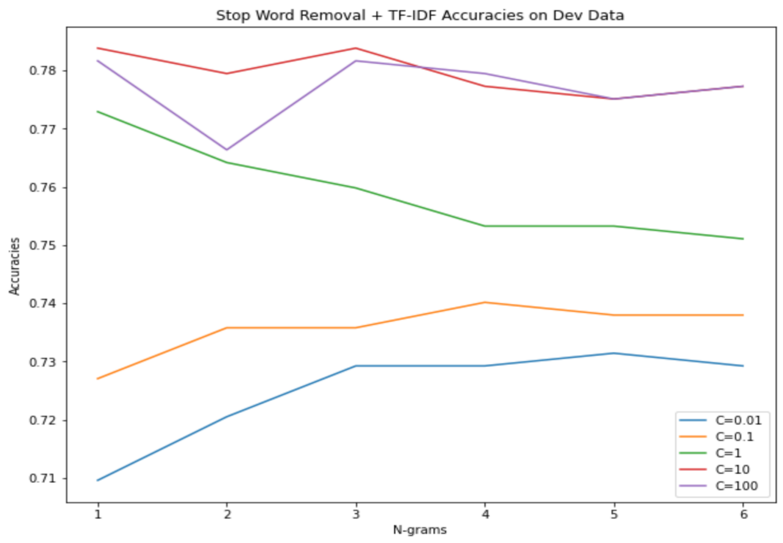
Thus, using the above three techniques it does not perform better than baseline model, because the models are not able to capture the significance of each word i.e., the importance of each word throughout the document. Now applying these techniques along with TF-IDF to vectorize and use as tokenizer and perform different experiments.

Now combining the techniques, the above three techniques with TF-IDF as a form of tokenizer:

d.) Stop Words Removal + TF-IDF: In this technique firstly, we will convert the sentence to lower case then remove the stop words from the sentence then TF-IDF is performed to form tokenized vector. The intuition behind this is to remove stop words then measure the significance of each word. Thus, it will enhance the perform. In this experiment we will run for different values of C and n just like we did in 2.1 section.

For different values of c and N the following is the accuracy on dev data:

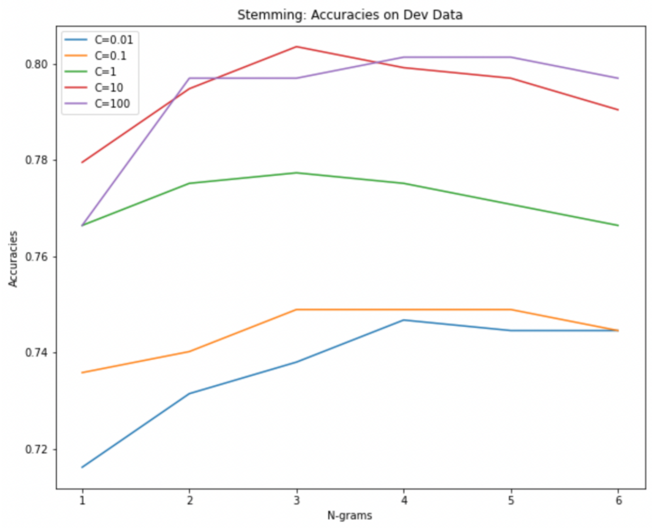
	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
C = 0.01	0.74235808	0.7510917	0.7510917	0.7510917	0.74672489	0.74235808
C = 0.1	0.7489083	0.7510917	0.74672489	0.74454148	0.74454148	0.74454148
C = 1	0.76637555	0.77074236	0.77074236	0.76200873	0.75982533	0.75982533
C = 10	0.79039301	0.78820961	0.79039301	0.78384279	0.77947598	0.77510917
C = 100	0.78165939	0.79257642	0.79475983	0.77729258	0.77947598	0.77947598



The optimal value of C is 10 and n-gram is 3, the accuracy obtained for these values is 0.79039301 which beats the accuracy of base model.

e.) Stemming + TF-IDF: In this technique firstly, we will convert the sentence to lower case then perform stemming on the sentence, once the stemming is performed, we will apply TF-IDF to capture the significance of stem words i.e., from derived word. In this experiment we will run for different values of C and n just like we did in 2.1 section.

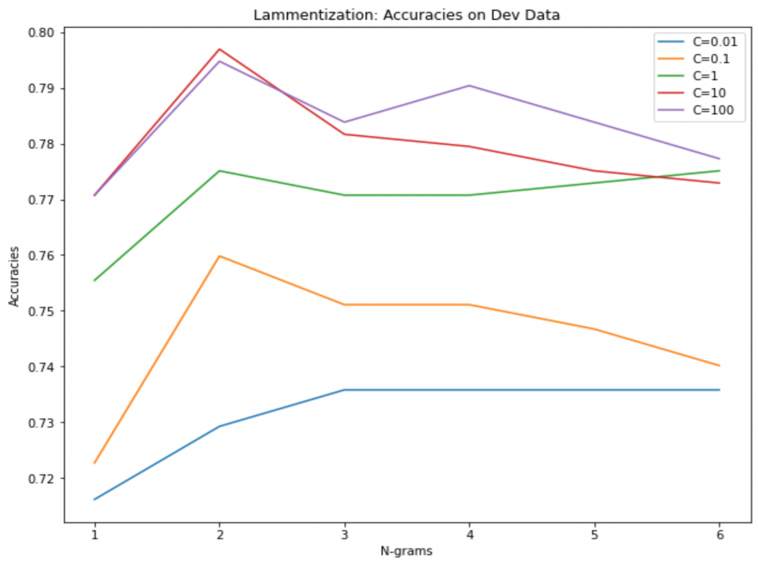
	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
C=0.01	0.71615721	0.73144105	0.73799127	0.74672489	0.74454148	0.74454148
C=0.1	0.73580786	0.74017467	0.7489083	0.7489083	0.7489083	0.74454148
C=1	0.76637555	0.77510917	0.77729258	0.77510917	0.77074236	0.76637555
C=10	0.77947598	0.79475983	0.80349345	0.79912664	0.79694323	0.79039301
C=100	0.76637555	0.79694323	0.79694323	0.80131004	0.80131004	0.79694323



The optimal value of C is 10 and n-gram is 3, the accuracy obtained for these values is 0.803493451 which beats the accuracy of base model.

f.) Lemmatization + TF-IDF: In this technique firstly, we will convert the sentence to lower case then perform lemmatization on the sentence, once the lemmatization is performed, we will apply TF-IDF to capture the significance of basic lemma word form. In this experiment we will run for different values of C and n just like we did in 2.1 section.

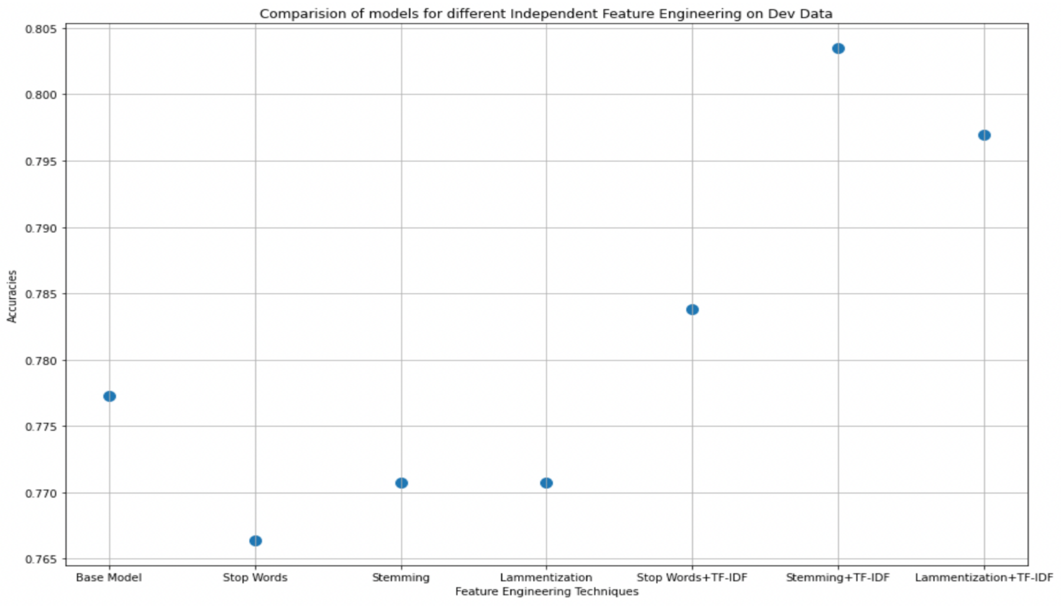
	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
C=0.01	0.71615721	0.72925764	0.73580786	0.73580786	0.73580786	0.73580786
C=0.1	0.72270742	0.75982533	0.7510917	0.7510917	0.74672489	0.74017467
C=1	0.75545852	0.77510917	0.77074236	0.77074236	0.77292576	0.77510917
C=10	0.77074236	0.79694323	0.78165939	0.77947598	0.77510917	0.77292576
C=100	0.77074236	0.79475983	0.78384279	0.79039301	0.78384279	0.77729258



The optimal value of C is 10 and n-gram is 2, the accuracy obtained for these values is **0.79694323** which beats the accuracy of base model.

Final Comparisons of different Independent Feature Engineering Techniques:

	Best Accuracies on Dev Data
Base Model	0.777292576419214
Stop Words Remmoval	0.7663755458515280
Stemming	0.7707423580786030
Lammentization	0.7707423580786030
Stop Words Remmoval + TF-IDF	0.78384279
Stemming + TF-IDF	0.80349345
Lammentization + TF-IDF	0.79694323



The best accuracy obtained among all the methods is Stemming + TF-IDF, as stemming converts all the words into its stem word thus preserves significance of root word. Since stemming is applied to change the word to its original form to process better in the logistic regression model with TF-IDF tokenizer, the accuracy obtained is 0.80349345.

References:

[1]. Arroyo-Fernández, I., Méndez-Cruz, C. F., Sierra, G., Torres-Moreno, J. M., & Sidorov, G. (2019). Unsupervised sentence representations as word information series: Revisiting TF–IDF. *Computer Speech & Language*, *56*, 107-129.

[2]. Ladani, D. J., & Desai, N. P. (2020, March). Stopword identification and removal techniques on tc and ir applications: A survey. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)* (pp. 466-472). IEEE.

[3]. Jivani, A. G. (2011). A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, *2*(6), 1930-1938.

[4]. Plisson, J., Lavrac, N., & Mladenic, D. (2004, May). A rule based approach to word lemmatization. In *Proceedings of IS*(Vol. 3, pp. 83-86).