Documentation for Beginner's Hypothesis by Saswat Das (sns)
18312025
MSM 1st Year
https://github.com/saswatpp/BHcerebro

Steps Followed in making the Model:
Data Preprocessing:
1. Data set imported named as TrainX and TrainY
2. Converted to Numpy Arrays

For Working with text Data , Sci Kit Learn library was used .
The CountVectorizer was imported for the bag-of-words representation .
The feature Names were viewed from the vocabulary and it was found that it contained the following :
1. Numbers like `'1zrv91769090467148 ' , ' 00 ' , ' 000 ' , ' 06 ' , ' 07 ' , ' 07356130 '`
These conveyed little or no information about the label .
2 . Words like ' customize', `' customized '` that convey same information but are considered distinct in the Vocbulary.

A simple Logistic Regression Model was used to check whether the feature engineering is helping or not . Some observations Made are:
1 . The CountVectorizer Parameter min_df(The number of Documents where the words must appear to be considered in the vocabulary) when set to 2 , reduced the number of unnecessary words but the Cross-Validation Score was not affected in large extent .
2 . Stopwords removal reduced the CV score . Most probably because the documents were short and simple english words helped convey meaning .

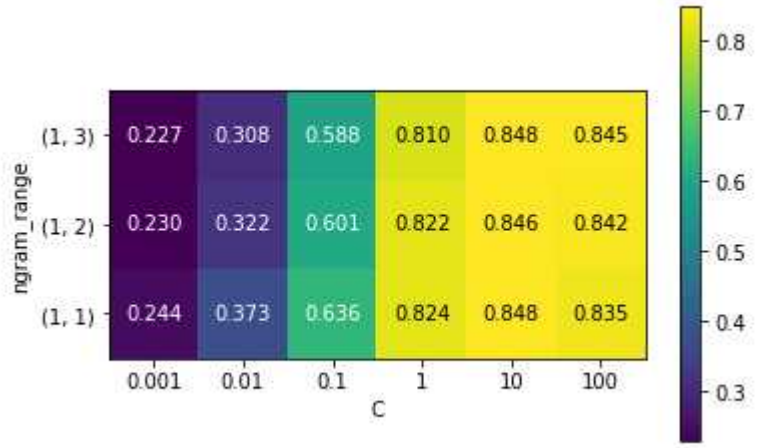
```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print("Number of stop words : { } ".format(len(ENGLISH_STOP_WORDS)))
print("Every 10th stopword : \ n { } ".format( list ( ENGLISH_STOP_WORDS ) [  :: 10] ) )
```

Following are some Stopwords :
after , about , all , already , found that convey some meaning in classification and hence removing them drastically reduced the cross validation score .

3. Term frequency–inverse document frequency(tf-idf) approach was applied but it did not help much probably because of the insufficient data set .
Instead of dropping features that are deemed unimportant, another approach is to rescale features by how informative we expect them to be. One of the most common ways to do this is using the term frequency–inverse document frequency (tf–idf) method .

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

The maximum CV score was 84.2% in this approach with n-gram range of (1,2)
4. Using N-grams increased the Vocabulary to around 7000(1,2) resulted in Overfitting and the CV score was low . For the N-grams range of (1,3) resulted in even higher overfitting with 15000 words in Vocab . This slowed the learning process(very slow with Gradient Boosted trees)
The heatmap for ngram_range and C value for LogisticRegression with tfidfVectorizer():



The Cross Validation Score didnot seem to increase more than 84%.
So Stemming was to be utilised .
PorterStemmer() from nltk was not helping in the Accuracy . Therefore Lemmatization was an option to find word roots from its english dictionary .
Lemmatization :
Stemming is always restricted to trimming the word to a stem, so "was" becomes "wa", while lemmatization can retrieve the correct base verb form, "be". Similarly, lemmatization can normalize "worse" to "bad", while stemming produces "wors".

In general, lemmatization is a much more involved process than stem- ming, but it usually produces better results than stemming when used for normaliz- ing tokens for machine learning.

Lemmatization conflates a certain features . It acts as a regularizer and can give best generalization when the data set is small like as in this case .

The lemmatizer was implemented using our custom Tokenizer for the CountVectorizer .
(lemma_vect) using a min_df of 2 to remove the unwanted words .

Technicality : We want to use the regexp based tokenizer
that is used by CountVectorizer and only use the lemmatization
from spacy . To this end, we replace en_nlp.tokenizer (the spacy tokenizer)
with the regexp based tokenization .

This was tried along with tfidfVectorizer but that didnot generalize well .

Lemmatization with Custom Tokenizer gave overall better CV score across all the Classification Algorithms . Min_df was not increased further as due to less data, generalization would be poor .

Then I wanted to find the best Algorithms with the best Cross Validation score .

For this I used GridSearch on individual Algorithms to find their best Cross-Validation Accuracy .
Following is an example code :

```
from sklearn.model_selection import GridSearchCV
param_grid = {'learning_rate': [0.001, 0.01, 0.1, 1, 10],'max_depth':[1,2,3]}
grid = GridSearchCV(GradientBoostingClassifier(), param_grid, cv=5)
grid.fit(Xtrain, Y_train)
```

```
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters: ", grid.best_params_)
```

This was for finding the best Accuracy using the Gradient Boosted Trees for Classification.
It was found that the best parameters were : max_depth = 3 and learning rate = 0.1 .
But the Cross Validation Score was around 84% .

Other algorithms tried were :

LogisticRegression
```
from sklearn.linear_model import LogisticRegression
```

SVC
```
from sklearn.svm import SVC
```

LinearSVC
```
from sklearn.svm import LinearSVC
```

RandomForestClassifier
```
from sklearn.ensemble import RandomForestClassifier
```

The best to perform was the LinearSVC. It gave a whooping Cross Validation Score of 88% . This is the highest among all of my previous tests .

Linear SVC :
It stands for Linear Support Vector Machines .
For Logistic Regression and Linear SVC the trade-off parameter that determines the strength of the regularization is called C, and higher values of C correspond to less regularization. When you use a high value for the parameter C, Logis ticRegression and LinearSVC try to fit the training set as best as possible, while with low values of the parameter C, the models put more emphasis on finding a coefficient vector (w) that is close to zero.

The best parameter value of C was found by grid search:

```
from sklearn.model_selection import GridSearchCV
param_grid = { ' C ' :  [ 0.001 ,  0.01 ,  0.1 ,  1 ,  10 ] }
grid = GridSearchCV(LinearSVC(), param_grid, cv=6)
grid.fit( Xtrain , Y_train )
print( " Best cross-validation score : { :.2f } ".format( grid.best_score_ ) )
print( " Best parameters : ", grid.best_params_ )
```

It was found that the best Cross Validation Score was 87 % and C = 0.1;
Additional tweaking of C greater than and less than 0.1 was tried before finaly .