Wikipedia Toxicity. Project 2

DESCRIPTION

Using NLP and machine learning, make a model to identify toxic comments from the Talk edit pages on Wikipedia. Help identify the words that make a comment toxic.

Problem Statement:

Wikipedia is the world's largest and most popular reference work on the internet with about 500 million unique visitors per month. It also has millions of contributors who can make edits to pages. The Talk edit pages, the key community interaction forum where the contributing community interacts or discusses or debates about the changes pertaining to a particular topic.

Wikipedia continuously strives to help online discussion become more productive and respectful. You are a data scientist at Wikipedia who will help Wikipedia to build a predictive model that identifies toxic comments in the discussion and marks them for cleanup by using NLP and machine learning. Post that, help identify the top terms from the toxic comments.

Domain: Internet

Analysis to be done: Build a text classification model using NLP and machine learning that detects toxic comments.

Content:

id: identifier number of the comment

comment_text: the text in the comment

toxic: 0 (non-toxic) /1 (toxic)

Steps to perform:

Cleanup the text data, using TF-IDF convert to vector space representation, use Support Vector Machines to detect toxic comments. Finally, get the list of top 15 toxic terms from the comments identified by the model.

Tasks:

Load the data using read_csv function from pandas package

Get the comments into a list, for easy text cleanup and manipulation

Cleanup:

Using regular expressions, remove IP addresses

Using regular expressions, remove URLs

Normalize the casing

Tokenize using word_tokenize from NLTK

Remove stop words

Remove punctuation

Define a function to perform all these steps, you'll use this later on the actual test set

Using a counter, find the top terms in the data.

Can any of these be considered contextual stop words?

Words like "Wikipedia", "page", "edit" are examples of contextual stop words

If yes, drop these from the data

Separate into train and test sets

Use train-test method to divide your data into 2 sets: train and test

Use a 70-30 split

Use TF-IDF values for the terms as feature to get into a vector space model

Import TF-IDF vectorizer from sklearn

Instantiate with a maximum of 4000 terms in your vocabulary

Fit and apply on the train set

Apply on the test set

Model building: Support Vector Machine

Instantiate SVC from sklearn with a linear kernel

Fit on the train data

Make predictions for the train and the test set

Model evaluation: Accuracy, recall, and f1_score

Report the accuracy on the train set

Report the recall on the train set:decent, high, low?

Get the f1_score on the train set

Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s

Adjust the appropriate parameter in the SVC module

Train again with the adjustment and evaluate

Train the model on the train set

Evaluate the predictions on the validation set: accuracy, recall, f1_score

Hyperparameter tuning

Import GridSearch and StratifiedKFold (because of class imbalance)

Provide the parameter grid to choose for 'C'

Use a balanced class weight while instantiating the Support Vector Classifier

Find the parameters with the best recall in cross validation

Choose 'recall' as the metric for scoring

Choose stratified 5 fold cross validation scheme

Fit on the train set

What are the best parameters?

Predict and evaluate using the best estimator

Use best estimator from the grid search to make predictions on the test set

What is the recall on the test set for the toxic comments?

What is the f1_score?

What are the most prominent terms in the toxic comments?

Separate the comments from the test set that the model identified as toxic

Make one large list of the terms

Get the top 15 terms

In [ ]:
```
import pandas as pd
import numpy as np
import spacy
import nltk
import string
import re
import spacy
from nltk.tokenize import sent_tokenize, word_tokenize
import nltk
from nltk.corpus import stopwords
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]:
```
True
```

In [ ]:
```
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [ ]:
```
df=pd.read_csv(r'train.csv')
```

In [ ]:
```
string.punctuation
```

Out[ ]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [ ]:
```
df.head()
```

Out[ ]:

	id	comment_text	toxic
0	e617e2489abe9bca	"\r\n\r\n A barnstar for you! \r\n\r\n The De...	0
1	9250cf637294e09d	"\r\n\r\nThis seems unbalanced. whatever I ha...	0
2	ce1aa4592d5240ca	Marya Dzmitruk was born in Minsk, Belarus in M...	0
3	48105766ff7f075b	"\r\n\r\nTalkback\r\n\r\n Dear Celestia... "	0
4	0543d4f82e5470b6	New Categories \r\n\r\nI honestly think that w...	0

In [ ]:
```
df.info()#there is no null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            5000 non-null   object
 1   comment_text  5000 non-null   object
 2   toxic         5000 non-null   int64 
dtypes: int64(1), object(2)
memory usage: 117.3+ KB
```

In [ ]:
```
feature=df.iloc[:,1]
feature
```

Out[ ]:
```
0       "\r\n\r\n A barnstar for you! \r\n\r\n  The De...
1       "\r\n\r\nThis seems unbalanced.  whatever I ha...
2       Marya Dzmitruk was born in Minsk, Belarus in M...
3           "\r\n\r\nTalkback\r\n\r\n Dear Celestia...  "
4       New Categories \r\n\r\nI honestly think that w...
                              ...                        
4995    "\r\n\r\n Dildo, if you read my response corre...
4996             CALM DOWN, CALM DOWN, DON'T GET A BIG DICK
4997    In my opinion Dougweller is using his privileg...
4998    The style section has been expanded too. I did...
4999    ANY ONE THAT IS NOT AGREEMENT WITH YOU OR IS A...
Name: comment_text, Length: 5000, dtype: object
```

In [ ]:
```
Y=df.toxic#unbalanced class as we can see that the number 0 Class is way more than than 1 Class
```

In [ ]:
```
Y.value_counts()
```

Out[ ]:
```
0    4563
1     437
Name: toxic, dtype: int64
```

In [ ]:
```
comment_list=df.comment_text.to_list()# changing it into  a list for better preprocessing
```

In [ ]:
```
len(comment_list)
```

Out[ ]:
```
5000
```

Cleanup:

Using regular expressions, remove IP addresses

Using regular expressions, remove URLs

Normalize the casing

Tokenize using word_tokenize from NLTK

Remove stop words

Remove punctuation

Define a function to perform all these steps, you'll use this later on the actual test set

In [ ]:
```
def textPreprocessor(feature):
  #remove Webpages addresses

   Url=[re.sub(r'^https?:\/\/.*[\r\n]*','',cmt) for cmt in feature]
 #remove ip address
   Ip=[re.sub(r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b ','',ip) for ip in Url]

     #remove Punctuation
   removePunctuations = [character for character in Ip if character not in string.punctuation]
   sentencesWithoutPunct = ''.join(removePunctuations)
   # c. Normalize the words
   wordNormalized = [ word.lower() for word in sentencesWithoutPunct]
   wordNormalized=''.join(wordNormalized)

   #Tokenize using word_tokenize from NLTK
   word1 = nltk.word_tokenize(wordNormalized)
  #remove stop words
   finalWords = [word for word in word1 if word not in stopwords.words('english')]
   #remove con

   return finalWords
```

In [ ]:
```
res = list(map(textPreprocessor,feature))

all_words = []
for word in res:
    all_words.extend(word)
len(all_words)

count_words = Counter(all_words)
count_words.most_common(15)
```

Out[ ]:
```
[('article', 1658),
 ('page', 1506),
 ('wikipedia', 1130),
 ('talk', 1044),
 ('please', 1039),
 ('would', 965),
 ('one', 855),
 ('like', 836),
 ('dont', 784),
 ('ass', 709),
 ('also', 657),
 ('think', 630),
 ('fuck', 630),
 ('see', 628),
 ('know', 595)]
```

In [ ]:
```
Cw=['wikipedia','article','page','talk','please','would','one','like','dont','ass']
```

In [ ]:
```
feature=[word for word in feature if word not in Cw]#removing conceptual words
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
wordVector = TfidfVectorizer(analyzer=textPreprocessor,max_features = 4000)


finalWordVectorVocab = wordVector.fit_transform(feature)
```

In [ ]:
```
finalWordVectorVocab
```

Out[ ]:
```
<5000x4000 sparse matrix of type '<class 'numpy.float64'>'
	with 109603 stored elements in Compressed Sparse Row format>
```

MOdel

In [ ]:
```
# Train Test Split

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(finalWordVectorVocab,
                                                 Y,
                                                 test_size=0.3,
                                                 random_state=6)
```

In [ ]:
```
from sklearn.svm import SVC
```

In [ ]:
```
Svc=SVC(kernel='linear')
```

In [ ]:
```
Svc.fit(X_train,y_train)
```

Out[ ]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:
```
yp=Svc.predict(X_test)
ypt=Svc.predict(X_train)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, classification_report,f1_score,recall_score
```

In [ ]:
```
print(Svc.score(X_train,y_train))
```

```
0.9697142857142858
```

In [ ]:
```
print(classification_report(y_train, ypt))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      3199
           1       0.99      0.66      0.79       301

    accuracy                           0.97      3500
   macro avg       0.98      0.83      0.89      3500
weighted avg       0.97      0.97      0.97      3500
```

Class **Weight**

In [ ]:
```
Svc=SVC(class_weight="balanced")
```

In [ ]:
```
Svc.fit(X_train,y_train)
```

Out[ ]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:
```
Svc.get_params()
```

Out[ ]:
```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': 'balanced',
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

In [ ]:
```
yp=Svc.predict(X_test)
ypt=Svc.predict(X_train)
```

In [ ]:
```
print(classification_report(y_train, ypt))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3199
           1       0.93      1.00      0.96       301

    accuracy                           0.99      3500
   macro avg       0.96      1.00      0.98      3500
weighted avg       0.99      0.99      0.99      3500
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

In [ ]:
```
param_grid = {
    'C': [0.01,0.1,1,10,100]

}
```

In [ ]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator =Svc , param_grid = param_grid,
                          cv = StratifiedKFold(4), n_jobs = -1, verbose = 1, scoring = "recall" )
```

In [ ]:
```
grid_search.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 5 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:   27.4s finished
```

Out[ ]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=None, shuffle=False),
             error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight='balanced', coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100]}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=False, scoring='recall', verbose=1)
```

In [ ]:
```
grid_search.best_estimator_
```

Out[ ]:
```
SVC(C=0.01, break_ties=False, cache_size=200, class_weight='balanced',
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [ ]:
```
y_test_pred = grid_search.best_estimator_.predict(X_test)
y_train_pred = grid_search.best_estimator_.predict(X_train)
```

In [ ]:
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95      1364
           1       0.00      0.00      0.00       136

    accuracy                           0.91      1500
   macro avg       0.45      0.50      0.48      1500
weighted avg       0.83      0.91      0.87      1500
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarnin
g: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]: