Natural Language Processing (NLP)

Course-End Project - Solution



**Help Wikipedia Detect and Cleanup Toxic Comments**

**Objective:** 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 -**

1. **Load the data using read\_csv function from pandas package**

import pandas as pd, numpy as np

import re

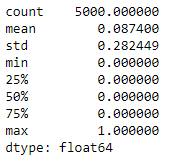
train0 = pd.read\_csv("train.csv")

train0.head()



#About 22% records are toxic

train0.describe().sum(axis=1)



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

comments = train0.comment\_text.values

len(comments)

# 5000

1. **Cleanup:** 
   1. **Using regular expressions, remove IP addresses**
   2. **Normalize the casing**

re.sub('[\d+\.{3}]\d+',"","My ip is 127.0.0.9, friend")



comments\_noip = [re.sub('[\d+\.{3}]\d+',"",txt) for txt in comments]

comments\_lower = [txt.lower() for txt in comments\_noip]

* 1. **Using regular expressions, remove URLs**

re.sub("\w+://\S+","", "@Rahim this course rocks! http://rahimbaig.com/ai")

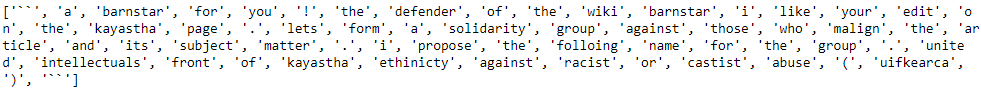
comments\_nourl = [re.sub("\w+://\S+","", txt) for txt in comments\_lower]

Also, removing extra returns and line breaks

comments\_nourl = [txt.replace("\'","") for txt in comments\_nourl]

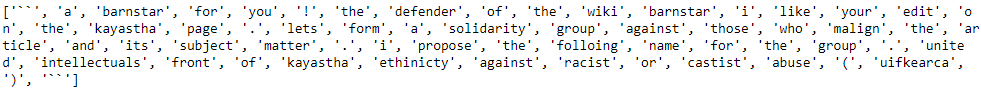
* 1. **Tokenize using word\_tokenize from NLTK**

from nltk.tokenize import word\_tokenize

print(word\_tokenize(comments\_nourl[0]))

comment\_tokens = [word\_tokenize(sent) for sent in comments\_nourl]

print(comment\_tokens[0])



* 1. **Remove stop words**
  2. **Remove punctuation**

from nltk.corpus import stopwords

from string import punctuation

stop\_nltk = stopwords.words("english")

stop\_punct = list(punctuation)

stop\_final = stop\_nltk + stop\_punct + ["...", "``","''", "====", "must"]

def del\_stop(sent):

return [term for term in sent if term not in stop\_final]

comments\_clean = [del\_stop(sent) for sent in comment\_tokens]

1. **Using a counter, find the top terms in the data.** 
   1. **Can any of these be considered contextual stop words?**

from collections import Counter

#Getting terms into one big list

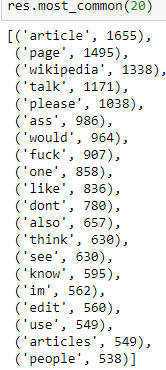
term\_list = []

for sent in comments\_clean:

term\_list.extend(sent)

res = Counter(term\_list)

res.most\_common(20)



* 1. **Words like “Wikipedia”, “page”, “edit” are examples of contextual stop words**
  2. **Drop these from the data**

stop\_context = ["article", "page", "wikipedia", "talk", "articles", "pages"]

stop\_final = stop\_final + stop\_context

comments\_clean = [del\_stop(sent) for sent in comment\_tokens]

comments\_clean = [" ".join(sent) for sent in comments\_clean]

comments\_clean[:2]

1. **Separate into train and test sets**
   1. **Use train-test method to divide your data into 2 sets: train and test**
   2. **Use a 70-30 split**

X = comments\_clean

y = train0.toxic

#Train test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state=42)

1. **Use TF-IDF values for the terms as feature to get into a vector space model**
   1. **Import TF-IDF vectorizer from sklearn**
   2. **Instantiate with a maximum of 4000 terms in your vocabulary**

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max\_features = 4000)

* 1. **Fit and apply on the train set**

len(X\_train), len(X\_test)



X\_train\_bow = vectorizer.fit\_transform(X\_train)

* 1. **Apply on the test set**

X\_test\_bow = vectorizer.transform(X\_test)

X\_train\_bow.shape, X\_test\_bow.shape



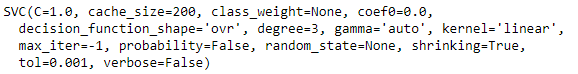
1. **Model building: Support Vector Machine**
   1. **Instantiate SVC from sklearn with a linear kernel**

from sklearn import svm

classifier\_linear = svm.SVC(kernel='linear')

* 1. **Fit on the train data**

classifier\_linear.fit(X\_train\_bow, y\_train)



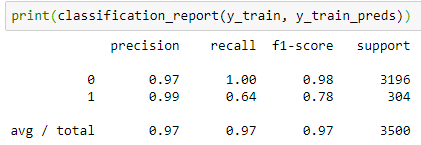
* 1. **Make predictions for the train set**

y\_train\_preds = classifier\_linear.predict(X\_train\_bow)

1. **Model evaluation: accuracy, recall, and f1\_score**
   1. **Report the accuracy on the train set**
   2. **Report the recall on the train set: decent, high, low?**
   3. **Get the f1\_score on the train set**

from sklearn.metrics import classification\_report

print(classification\_report(y\_train, y\_train\_preds))



1. **Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s**
   1. **Adjust the appropriate parameter in the SVC module**

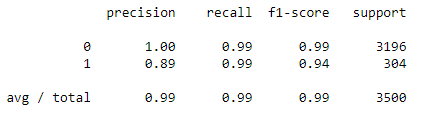
classifier\_linear = svm.SVC(kernel='linear', class\_weight="balanced")

1. **Train again with the adjustment and evaluate**
   1. **Train the model on the train set**
   2. **Evaluate the predictions on the validation set: accuracy, recall, and f1\_score**

classifier\_linear.fit(X\_train\_bow, y\_train)

y\_train\_pred = classifier\_linear.predict(X\_train\_bow)

print(classification\_report(y\_train, y\_train\_pred))



1. **Hyperparameter tuning:**
   1. **Import GridSearch and StratifiedKFold (because of class imbalance)**

from sklearn.model\_selection import GridSearchCV, StratifiedKFold

* 1. **Provide the parameter grid to choose for ‘C’**

# Create the parameter grid based on the results of random search

param\_grid = {

'C': [0.1, 1, 10,1000, 10000, 100000]

}

* 1. **Use a balanced class weight while instantiating the Support Vector Classifier**

classifier\_svm = svm.SVC(random\_state=42, class\_weight="balanced", kernel="linear")

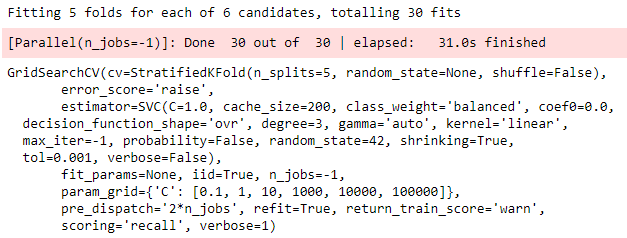
1. **Find the parameters with the best recall in cross validation**
   1. **Choose ‘recall’ as the metric for scoring**
   2. **Choose stratified 5 fold cross validation scheme**
   3. **Fit on the train set**

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = classifier\_svm, param\_grid = param\_grid,

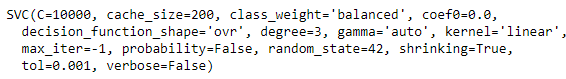
cv = StratifiedKFold(5), n\_jobs = -1, verbose = 1, scoring = "recall" )

grid\_search.fit(X\_train\_bow, y\_train)



1. **What are the best parameters?**

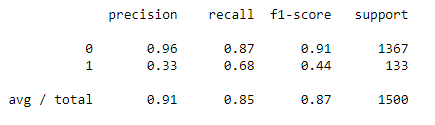
grid\_search.best\_estimator\_



1. **Predict and evaluate using the best estimator**
   1. **Use best estimator from the grid search to make predictions on the test set**

y\_test\_pred = grid\_search.best\_estimator\_.predict(X\_test\_bow)

* 1. **What is the recall on the test set for the toxic comments?**
  2. **What is the f1\_score?**

print(classification\_report(y\_test, y\_test\_pred))

1. **What are the most prominent terms in the toxic comments?**
   1. **Separate the comments from the test set that the model identified as toxic**
   2. **Make one large list of the terms**
   3. **Get the top 15 terms**

toxic\_comments = pd.Series(X\_test)[y\_test\_pred == 1].values

term\_list = []

for comment in toxic\_comments:

term\_list.extend(word\_tokenize(comment))

cts = Counter(term\_list)

cts.most\_common(15)

