How to detect 'toxic' comments using Machine Learning techniques

Metrics and limitations

Let's assume the program could make a mistake in 1 out of 20 cases

Metrics used:

1. Recall

Bad comments ratio

If equal to 1, all bad comments were found

2. Precision

Probability of a found bad comment to be bad

We don't want it to be lower than 0.95

Goals:

1. Develop a Binary classificator (input = comment(text); output =)
2. Maximize Recall
3. Precision >= 0.95

Dataset Composition

Dataset used:

Russian Language Toxic Comments

https://www.kaggle.com/datasets/blackmoon/russian-language-toxic-comments?resource=download

```
# Importing useful libraries

import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
nltk.download('stopwords')

#divides a text into a list of sentences
#by using an unsupervised algorithm to build a model for abbreviation
#words, collocations, and words that start sentences.
nltk.download('punkt')

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import precision_score, recall_score, precision_recall_curve
from matplotlib import pyplot as plt

from sklearn.metrics import precision_recall_curve

import sklearn.metrics as skm
from sklearn.metrics import precision_recall_curve

import numpy as np
from sklearn.model_selection import GridSearchCV
```

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

Exploratory data analysis (EDA)

```
# Read data
df = pd.read_csv('labeled.csv', sep=',')
```

```
# Data shape
df.shape
```

```
(14412, 2)
```

14412 samples and 2 columns

```
# Return first 7 rows
df.head(7)
```

	comment	toxic
0	Верблюдов-то за что? Дебилы, бл...\n	1.0
1	Хохлы, это отдушина затюканого россиянина, мол...	1.0
2	Собаке - собачья смерть\n	1.0
3	Страницу обнови, дебил. Это тоже не оскорблени...	1.0
4	тебя не убедил 6-страничный пдф в том, что Скр...	1.0
5	Для каких стан является эталоном современная с...	1.0
6	В шапке были ссылки на инфу по текущему фильму...	0.0

```
# Change data type from float to integer for data in the 'toxic' column
df['toxic'] = df['toxic'].apply(int)
```

```
# Result:
df.head(7)
```

	comment	toxic
0	Верблюдов-то за что? Дебилы, бл...\n	1
1	Хохлы, это отдушина затюканого россиянина, мол...	1
2	Собаке - собачья смерть\n	1
3	Страницу обнови, дебил. Это тоже не оскорблени...	1
4	тебя не убедил 6-страничный пдф в том, что Скр...	1
5	Для каких стан является эталоном современная с...	1
6	В шапке были ссылки на инфу по текущему фильму...	0

```
# Data distribution: how many toxic comments in the dataset
df['toxic'].value_counts()
```

```
0    9586
1    4826
Name: toxic, dtype: int64
```

There are almost twice as many good comments as toxic ones.

```
# Return first 7 toxic comments
for i in df[df['toxic'] == 1]['comment'].head(7):
  print(i)
```

```
Верблюдов-то за что? Дебилы, бл...
Хохлы, это отдушина затюканого россиянина, мол, вон, а у хохлов еще хуже. Если бы хохлов не было, кисель их бы придумал.
Собаке - собачья смерть
Страницу обнови, дебил. Это тоже не оскорбление, а доказанный факт - не-дебил про себя во множественном числе писать не будет. Или мы в теб
тебя не убедил 6-страничный пдф в том, что Скрипалей отравила Россия? Анализировать и думать пытаешься? Ватник что ли?)
Для каких стан является эталоном современная система здравоохранения РФ? Для Зимбабве? Ты тупой? хохлы
УПАД Т! ТАМ НЕЛЬЗЯ СТРОИТЬ! ТЕХНОЛОГИЙ НЕТ! РАЗВОРУЮТ КАК ВСЕГДА! УЖЕ ТРЕЩИНАМИ ПОШ Л! ТУПЫЕ КИТАЗЫ НЕ МОГУ
```

```
# Return first 7 good comments
for i in df[df['toxic'] == 0]['comment'].head(7):
```

```
  print(i)
```

```
В шапке были ссылки на инфу по текущему фильму марвел. Эти ссылки были заменены на фразу Репортим брипидора, игнорируем его посты. Если 
Почитайте посты у этого автора,может найдете что нибудь полезное. Надеюсь помог) https: pikabu.ru story obyichnyie budni dezsluzhbyi
Про графику было обидно) я так то проходил все серии гта со второй части по пятую, кроме гта 4. И мне не мешала графика ни в одной из часте
https: pp.userapi.com c848520 v848520411 11627b cOhWqFbGjWE.jpg
Возьмём как пример Россию, западноевропейские страны и США. Идёт метисация, сознательная политика замещения белого населения на пришлое черно
Может и старый, может и маразматик. Про то писать кириллицей или латинницей вам виднее, не спорю. Но как задвигают русский язык уже видно.
Шизофазия (речевая разорванность) симптом психических расстройств, выражающийся в нарушении структуры речи, при которой, в отличие от речевой
```

This Dataset is not perfect. It is quite hard to create a flawless model having flawed Dataset.

```
# I will use 500 samples to train the model
train_df, test_df = train_test_split(df, test_size=500)
```

```
# Checking if test subset consists of 500 samples
test_df.shape
```

```
(500, 2)
```

```
# Distribution of good and toxic comments in the test subset
test_df['toxic'].value_counts()
```

```
0    343
1    157
Name: toxic, dtype: int64
```

Good:toxic ratio is 2.1:1

```
# Distribution of good and toxic comments in the train subset
train_df['toxic'].value_counts()
```

```
0    9243
1    4669
Name: toxic, dtype: int64
```

Good:toxic ratio is 1.98:1

Text preprocessing

I will use logistic regression to create model of scikit learn library

Dataset prerocessing:

1. Tokenization of data (division of Data into a list of sentences)
2. Delete punctuation marks (e.g. question marks, commas, etc.) and stop words (insignificant words such as 'and', 'or', 'wow!', etc.)
3. Stemming (reducing derivative words to their word stem)

1. Tokenization

```
# Gets the 13th comment
sentence_example = df.iloc[13]['comment']
print(f'Initial text: {sentence_example}')
```

```
Initial text: Уроды!! у нас в семье 3 поколения там родились
```

```
# Tokenization using nltk library
tokens = word_tokenize(sentence_example, language='russian')
print(f'Tokens: {tokens}')
```

```
Tokens: ['Уроды', '!', '!', 'у', 'нас', 'в', 'семье', '3', 'поколения', 'там', 'родились']
```

2. Delete punctuation marks

```
# Deleting punctuation marks using punctuation from string module
tokens_without_punctuation = [i for i in tokens if i not in string.punctuation]
print(f'Tokens without punctuation: {tokens_without_punctuation}')
```

```
Tokens without punctuation: ['Уроды', 'у', 'нас', 'в', 'семье', '3', 'поколения', 'там', 'родились']
```

```
# Getting stop words from nltk library
#russian_stop_words = stopwords.words['russian']
russian_stop_words = stopwords.words('russian')
print(russian_stop_words)
```

```
['и', 'в', 'во', 'не', 'что', 'он', 'на', 'я', 'с', 'со', 'как', 'а', 'то', 'все', 'она', 'так', 'его', 'но', 'да', 'ты', 'к', 'у
```

```
# Deleting stop words
tokens_without_stopwords_and_punctuation = [i for i in tokens_without_punctuation if i not in russian_stop_words]
print(f'Tokens without stopwords and punctuation: {tokens_without_stopwords_and_punctuation}')
```

```
Tokens without stopwords and punctuation: ['Уроды', 'семье', '3', 'поколения', 'родились']
```

3. Stemming

```
# Stemming (lowercase and deleting suffixes)
snowball = SnowballStemmer(language='russian')
stemmed_tokens = [snowball.stem(i) for i in tokens_without_stopwords_and_punctuation]
print(f'Tokens after stemming: {stemmed_tokens}')
```

```
Tokens after stemming: ['урод', 'сем', '3', 'поколен', 'род']
```

```
# Function to apply stemming to every sentence in the Dataset

snowball = SnowballStemmer(language='russian')
russian_stop_words = stopwords.words('russian')

def tokenize_sentence(sentence: str, remove_stop_words: bool = True):
  # Tokenizing the Data
  tokens = word_tokenize(sentence, language='russian')
  # Removing punctuation marks
  tokens = [i for i in tokens if i not in string.punctuation]
  if remove_stop_words:
    tokens = [i for i in tokens if i not in russian_stop_words]
  tokens = [snowball.stem(i) for i in tokens]
  return tokens
```

```
# Checking if function works correctly
print(f'Tokens after stemming: {tokenize_sentence(sentence_example)}')
```

```
Tokens after stemming: ['урод', 'сем', '3', 'поколен', 'род']
```

Model training

```
# Converting Data into TF-IDF matrix

vectorizer = TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))
```

```
# Train stage

features = vectorizer.fit_transform(train_df['comment'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528: UserWarning: The parameter 'token_patterr
  warnings.warn(
```

```
# Class creation of Logistic Regression
model = LogisticRegression(random_state=0)
# Train model with features and labels (train_df['toxic'])
model.fit(features, train_df['toxic'])
```

```
LogisticRegression
LogisticRegression(random_state=0)
```

```
# Checking how does the model work
model.predict(features[0])
```

```
array([0])
```

The model predicts that the 1st comment is a 'good' one

```
# Let's read that 1st comment
train_df['comment'].iloc[0]
```

```
'На госпочту письмо приходит от ГИБДД или ФССП сразу, как сформировано уведомление. А в личном кабинете госуслуг - после обработки уведомл
ения от тех же ГИБДД и т.д. У меня бывало дня два проходило, пока на госуслугах штраф появится.\n'
```

As we can see, the prediction was correct

```
# Pipeline receives tuples with the component name and the class
model_pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
    ('model', LogisticRegression(random_state=0))
]
)
```

```
model_pipeline.fit(train_df['comment'], train_df['toxic'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528: UserWarning: The parameter 'token_patterr
  warnings.warn(
```

```
Pipeline
TfidfVectorizer
LogisticRegression
```

```
# Let's compose our own 'good' comment to test if the model works correctly
test_comment = "Нормальный ноутбук за приемлемую цену"

model_pipeline.predict([test_comment])
```

```
array([0])
```

The model predicted that the comment is 'good'. Which is true.

```
# Let's compose our own 'toxic' comment to test if the model works correctly
test_comment = "Дурацкий ноутбук и продавец дебил"

model_pipeline.predict([test_comment])
```

```
array([1])
```

The model predicted that the comment is 'toxic'. That is true.

Metrics

```
# Precision
precision_score(y_true=test_df['toxic'], y_pred=model_pipeline.predict(test_df['comment']))
```

```
0.9174311926605505
```

Precision is 0.917 which is lower than our goal of 0.95

```
# Recall
recall_score(y_true=test_df['toxic'], y_pred=model_pipeline.predict(test_df['comment']))
```

```
0.6369426751592356
```

```
# We can obtain a much better prediction result adjusting the threshold value
precision, recall, thresholds = precision_recall_curve(y_true=test_df['toxic'], probas_pred=model_pipeline.predict_proba(test_
```

```
# Find indexes > 0.95
np.where(precision > 0.95)
```

```
(array([415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,
        428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 459,
        462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474,
        475, 476, 477, 478, 479, 490, 491, 492, 493, 494, 495, 496, 497,
        498, 499, 500]),)
```

With threshold greater than 415 the precision is greater than 0.95

```
thresholds[415]
```

```
0.6010245351718095
```

```
precision_score(y_true=test_df['toxic'], y_pred=model_pipeline.predict_proba(test_df['comment'])[:, 1] > thresholds[415])
```

```
0.9523809523809523
```

The precision is now above 0.95

```
recall_score(y_true=test_df['toxic'], y_pred=model_pipeline.predict_proba(test_df['comment'])[:, 1] > thresholds[391])
```

```
0.6305732484076433
```

Improving the model

```
grid_pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
    ('model',
     GridSearchCV(
        LogisticRegression(random_state=0),
        param_grid={'C': [0.1, 1, 10.]},
        cv=3,
        verbose=4
     )
  )
])
```

```
grid_pipeline.fit(train_df['comment'], train_df['toxic'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528
  warnings.warn(
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV 1/3] END ..........................C=0.1;, score=0.683 total time=   0.
[CV 2/3] END ..........................C=0.1;, score=0.687 total time=   0.
[CV 3/3] END ..........................C=0.1;, score=0.689 total time=   0.
[CV 1/3] END ............................C=1;, score=0.839 total time=   0.
[CV 2/3] END ............................C=1;, score=0.838 total time=   0.
[CV 3/3] END ............................C=1;, score=0.837 total time=   1.
[CV 1/3] END ........................C=10.0;, score=0.867 total time=   1.
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

The best score 0.868 is for C=10.0

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
```

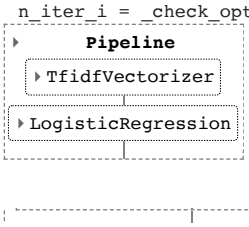
```
model_pipeline_c10 = Pipeline([
    ('vectorizer', TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
    ('model', LogisticRegression(random_state=0, C=10.))
]
)
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
model_pipeline_c10.fit(train_df['comment'], train_df['toxic'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528: Use
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```




```
precision_c10, recall_c10, thresholds_c10 = precision_recall_curve(y_true=test_df['toxic'], probas_pred=model_pipeline_c10.pre
```

```
np.where(precision_c10 > 0.95)
```

```
(array([410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422,
        423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435,
        436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448,
        449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461,
        462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474,
        475, 476, 477, 478, 479, 483, 484, 485, 486, 487, 488, 489, 490,
        491, 492, 493, 494, 495, 496, 497, 498, 499, 500]),)
```

```
precision_score(y_true=test_df['toxic'], y_pred=model_pipeline_c10.predict_proba(test_df['comment'])[:, 1] > thresholds_c10[41
```

```
0.9550561797752809
```

```
recall_score(y_true=test_df['toxic'], y_pred=model_pipeline_c10.predict_proba(test_df['comment'])[:, 1] > thresholds_c10[410])
```

```
0.5414012738853503
```

Result: Now the precision value is greater than 0.95 But the previous recall value was better at 0.61. Even though the precision increased, we missed some 'toxic' comments.

✓ 0s completed at 22:58