Text Classification with Bag of Words - Natural Language Processing

> *"Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data."* - Wikipedia

> ***Bag of Words**: The bag-of-words (BOW) model is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears.*

Outline:

1. Download and explore a real-world dataset
2. Apply text preprocessing techniques
3. Implement the bag of words model
4. Train ML models for text classification
5. Make predictions and submit to Kaggle

Dataset: https://www.kaggle.com/c/quora-insincere-questions-classification

Download and Explore the Data

Outline:

1. Download the dataset from Kaggle to Colab

2. Explore the data using Pandas
3. Create a small working sample

Download the Data to Colab

Upload your `kaggle.json` to Colab. Get it here: https://www.kaggle.com/docs/api#authentication

```
!ls .
```

```
sample_data
```

```
import os
```

```
os.chmod("/content/kaggle.json",600)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-6-b233aa070a27> in <cell line: 1>()
----> 1 os.chmod("/content/kaggle.json",600)

FileNotFoundError: [Errno 2] No such file or directory: '/content/kaggle.json'

SEARCH STACK OVERFLOW
```

```
os.environ ['KAGGLE_CONFIG_DIR'] = '.'
```

```
!kaggle competitions download -c quora-insincere-questions-classification -f train.csv -p data
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in .. Or use the environment method.
```

```
IS_KAGGLE = 'KAGGLE_KERNEL_RUN_TYPE' in os.environ
```

```
if IS_KAGGLE:
    data_dir = '../input/quora-insincere-questions-classification'
    train_fname = data_dir + '/train.csv'
    test_fname = data_dir + '/test.csv'
    sample_fname = data_dir + '/sample_submission.csv'
else:
    os.environ['KAGGLE_CONFIG_DIR'] = '.'
    !kaggle competitions download -c quora-insincere-questions-classification -f train.csv -p data
    !kaggle competitions download -c quora-insincere-questions-classification -f test.csv -p data
    !kaggle competitions download -c quora-insincere-questions-classification -f sample_submission.csv -p data
    train_fname = 'data/train.csv.zip'
    test_fname = 'data/test.csv.zip'
    sample_fname = 'data/sample_submission.csv.zip'
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in .. Or use the environment method.
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in .. Or use the environment method.
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in .. Or use the environment method.
```

Explore the Data using Pandas

```
import pandas as pd
```

```
raw_df = pd.read_csv(train_fname)
```

```
raw_df
```

```
sincere_df = raw_df[raw_df.target == 0]
```

```
sincere_df.question_text.values[:10]
```

```
insincere_df = raw_df[raw_df.target == 1]
```

```
insincere_df.question_text.values[:10]
```

```
raw_df.target.value_counts(normalize=True)
```

```
raw_df.target.value_counts(normalize=True).plot(kind='bar')
```

```
test_df = pd.read_csv(test_fname)
```

```
test_df
```

```
sub_df = pd.read_csv(sample_fname)
```

```
sub_df
```

```
sub_df.prediction.value_counts()
```

Create a Working Sample

```
if IS_KAGGLE:
    SAMPLE_SIZE = len(raw_df)
else:
    SAMPLE_SIZE = 100_000
```

```
sample_df = raw_df.sample(SAMPLE_SIZE, random_state=42)
```

```
sample_df
```

Text Preprocessing Techniques

Outline:

1. Understand the bag of words model
2. Tokenization
3. Stop word removal

4. Stemming

Bag of Words Intuition

1. Create a list of all the words across all the text documents
2. You convert each document into vector counts of each word

Limitations:

1. There may be too many words in the dataset
2. Some words may occur too frequently
3. Some words may occur very rarely or only once
4. A single word may have many forms (go, gone, going or bird vs. birds)

```
q0 = sincere_df.question_text.values[1]
```

```
q0
```

```
q1 = raw_df[raw_df.target == 1].question_text.values[0]
```

```
q1
```

Tokenization

splitting a document into words and separators

```
import nltk
```

```
from nltk.tokenize import word_tokenize
```

```
nltk.download('punkt')
```

```
q0
```

```
word_tokenize(q0)
```

```
word_tokenize(' this is (something) with, a lot of, punctuation;')
```

```
q1
```

```
word_tokenize(q1)
```

```
q0_tok = word_tokenize(q0)
q1_tok = word_tokenize(q1)
```

Stop Word Removal

Removing commonly occuring words

```
q1_tok
```

```
from nltk.corpus import stopwords
```

```
nltk.download('stopwords')
```

```
english_stopwords = stopwords.words('english')
```

```
", ".join(english_stopwords)
```

```
def remove_stopwords(tokens):
    return [word for word in tokens if word.lower() not in english_stopwords]
```

```
q0_tok
```

```
q0_stp = remove_stopwords(q0_tok)
```

```
q0_stp
```

```
q1_stp = remove_stopwords(q1_tok)
```

```
q1_tok
```

```
q1_stp
```

Stemming

"go", "gone", "going" -> "go" "birds", "bird" -> "bird"

```
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer(language='english')

stemmer.stem('going')

stemmer.stem('supposedly')

q0_stm = [stemmer.stem(word) for word in q0_stp]

q0_stp

q0_stm

q1_stm = [stemmer.stem(word) for word in q1_stp]

q1_stp

q1_stm
```

Lemmatization

"love" -> "love" "loving" -> "love" "lovable" -> "love"

Implement Bag of Words

Outline:

1. Create a vocabulary using Count Vectorizer
2. Transform text to vectors using Count Vectorizer
3. Configure text preprocessing in Count Vectorizer

Create a Vocabulary

```
sample_df
```

```
small_df = sample_df[:5]
```

```
small_df
```

```
small_df.question_text.values
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
small_vect = CountVectorizer()
```

```
small_vect.fit(small_df.question_text)
```

```
small_vect.get_feature_names_out()
```

Transform documents into Vectors

```
vectors = small_vect.transform(small_df.question_text)
```

```
vectors
```

```
vectors.shape
```

```
small_df.question_text.values[0]
```

```
vectors[0].toarray()
```

```
vectors.toarray()
```

Configure Count Vectorizer Parameters

```
stemmer = SnowballStemmer(language='english')
```

```
def tokenize(text):
    return [stemmer.stem(word) for word in word_tokenize(text)]
```

```
tokenize('What is the really (dealing) here?')
```

```
vectorizer = CountVectorizer(lowercase=True,
                             tokenizer=tokenize,
                             stop_words=english_stopwords,
                             max_features=1000)
```

```
%%time
vectorizer.fit(sample_df.question_text)
```

```
len(vectorizer.vocabulary_)
```

```
vectorizer.get_feature_names_out()[:100]
```

```
%%time
inputs = vectorizer.transform(sample_df.question_text)
```

```
inputs.shape
```

```
inputs
```

```
sample_df.question_text.values[0]
```

```
test_df
```

```
%%time
test_inputs = vectorizer.transform(test_df.question_text)
```

ML Models for Text Classification

Outline:

- Create a training & validation set
- Train a logistic regression model
- Make predictions on training, validation & test data

Split into Training and Validation Set

```
sample_df
```

```
inputs.shape
```

```
from sklearn.model_selection import train_test_split
```

```
train_inputs, val_inputs, train_targets, val_targets = train_test_split(inputs, sample_df.target, test_size=0.3, random_state=42)
```

```
train_inputs.shape
```

```
train_targets.shape
```

```
val_inputs.shape
```

```
val_targets.shape
```

Train Logistic Regression model

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

```
MAX_ITER = 1000
```

```
model = LogisticRegression(max_iter=MAX_ITER, solver='sag')
```

```
%%time
model.fit(train_inputs, train_targets)
```

Make predictions using the model

```
train_preds = model.predict(train_inputs)
```

```
train_targets
```

```
train_preds
```

```
pd.Series(train_preds).value_counts()
```

```
pd.Series(train_targets).value_counts()
```

```
from sklearn.metrics import accuracy_score

accuracy_score(train_targets, train_preds)

import numpy as np

accuracy_score(train_targets, np.zeros(len(train_targets)))

from sklearn.metrics import f1_score

f1_score(train_targets, train_preds)

f1_score(train_targets, np.zeros(len(train_targets)))

random_preds = np.random.choice((0, 1), len(train_targets))
f1_score(train_targets, random_preds)

val_preds = model.predict(val_inputs)

accuracy_score(val_targets, val_preds)

f1_score(val_targets, val_preds)

sincere_df.question_text.values[:10]

sincere_df.target.values[:10]

model.predict(vectorizer.transform(sincere_df.question_text.values[:10]))

insincere_df.question_text.values[:10]

insincere_df.target.values[:10]

model.predict(vectorizer.transform(insincere_df.question_text.values[:10]))
```

Make Predictions and Submit to Kaggle

```
test_df
```

```
test_inputs.shape
```

```
test_preds = model.predict(test_inputs)
```

```
sub_df
```

```
sub_df.prediction = test_preds
```

```
sub_df.prediction.value_counts()
```

```
sub_df
```

```
sub_df.to_csv('submission.csv', index=None)
```

```
!head submission.csv
```