News Article Classification

Presented by

Anand Geed (IDS2022005)

Table of Contents

1. Introduction
    a. Background
    b. Literature Review
    c. Problem Formulation
    d. Motivation
    e. Project Questions
    f. Scope/Limitation
    g. Target Group
2. Proposed Methodology
    a. Tokenization
    b. N Gram
    c. Removing Stopwords
    d. TF – IDF
    e. Navie Bayes
3. Tools Description
4. Dataset Description
5. Implementation
6. Result and Analysis
7. References

Introduction

In today's society a large portion of the world's population get their news on their electronic devices. Many of the major newspapers have apps that can be used on phones and tablets, in which their articles are displayed in a flow, typically sorted on time of publication. Not all readers are interested in all of the articles published each day.

Many newspapers have divided their articles into categories where one can display all articles about e.g., Sports, Politics or Science. The problem is that a user would have to search for different pages to read about different categories. In this project we explore the possibility of using text classification to teach a machine to classify the news articles based given tags to help users easily find their topic or category of interest.

Background

Machine learning is a technique where one lets a machine learn about an area based on data it is fed with. The collection of data can be either supervised or unsupervised, or a mix of both.

Unsupervised learning concerns finding natural categorizations in data. The machine does not know what categories an object belongs to or what classifications exist. In supervised learning pre-defined classes are used and the machine knows what categories an object belongs to.

Text classification is the process of classifying a text document into categories. This is done by using machine learning. A document can belong to one, many or no categories. In this project the data will always belong to just one category.

There are many different methods available that can be used for text classification. One of them is Best Individual Feature (BIF), in which an evaluation method is used based on a single word in the document. E.g., the evaluation can be based on the frequency of how often that word is used. Another method is Sequential Forward Selection (SFS), in which a single word is selected as the best one depending on some criteria and the following words are added until a specific number of words has been reached. SFS often gives better results than BIF, however since SFS is more costly in terms of computation and memory cost it is not often used.

Naïve Bayes is a family of models for text classification in which naive assumptions are made to categorize text into different groups. One area of use in which Naïve Bayes have shown great results is spam filtering.

The naive assumption is that words are individually significant and does not have a connection to each other. Through this assumption you are able to determine what category a text belongs to through statistical significance. The percentage of correctly classified text will be described as classification accuracy in the rest of this paper.

Literature Review

In a study done by William B. Cavnar and John M. Trenkle they investigated how one can use the N-Gram model to classify a text into categories. The N-Gram model is used because it has some tolerance for e.g., spelling errors. They achieved 99.8% classification accuracy for news articles and about 80% classification accuracy for computer-related articles

Thorsten Joachims investigated how well a Support Vector Machine (SVM) classifies text compared to other classification models. The comparison is done by classifying a number of news articles into categories. SVM achieved a classification accuracy of about 86% while Naïve Bayes achieved 72%

In the article Learning and Revising User Profiles: The Identification of Interesting Web Sites the authors M. Pazzani and D. Billsus used a Naïve Bayes classifier to determine which web pages a user could be interested in. They

tested the classifier on nine different areas and it proved to be both effective and accurate. The authors noted that the classification accuracy increased in relation to the amount of examples that were used, which was expected based on previous research

Problem formulation

The goal of this study is to evaluate if it is possible to identify a news article as interesting for a user with the help of the Naïve Bayes model. The evaluation will be done on small datasets of articles. The datasets will be in three different sizes, 500, 1000 and 2000 articles. In this study both the articles "categories" and the text of the article is used as input data for the evaluation of articles and the two are compared to see which one gives the most accurate result.

Motivation

Since many people use their phones and tablets to access newspaper articles it is easier for users to find new newspapers from all around the world. This leads to a much broader span of customers for the news companies, which makes it more difficult for the companies to keep their customers satisfied. To make this easier they must consider customer care and personalization. We could not find any news companies that offer the kind of personalization presented in this paper. Most have just created pages for different general topics like Sports and Tech. The research done in this project could help newspapers to personalize the flow of articles they have on their apps and provide articles that they know the users wants to read.

Project Questions

Q1	How good is the classification accuracy when using Naive Bayes to classify news articles as different categories?
Q2	How will the classification accuracy change when using 500, 1000, or 2225 articles in a dataset?
Q3	How will the classification accuracy change when different N-Grams are used as input features?

Scope/Limitation

We have decided to use the newspaper site BBC. It is one of the biggest newspaper sites in world. It contains a broad range of topics and there are a lot of new articles written every day. They also have a tag system for their articles. Although they do not seem to have a strict set of rules regarding categories, and categories seem to vary in both quality and quantity. Our goal with this experiment is not to implement a complete solution for classifying newspaper articles based on given categories. Instead, our goal is to investigate how can we improve the accuracy with different techniques. We are not developing an application but instead we will just be testing the methods using existing software. Since one of our research questions is to see if the result is different when we use different types of N Grams and Different size of datasets. For that experiment we will choose a dataset that only consists of the articles with two or more tags. One of our limitations is the relatively small test group. If we would have used outside people to collect the information, we would have been dependent on their completion of the classification and since we wanted to be sure we could complete the study in time we decided on a smaller test group.

Target group

This project is mostly interesting for companies that provide news services. If successful, the experiment will provide these companies with the information on how to make their services more personalized for customers. This will, hopefully, in turn give them an edge on the market. The report should be understandable even for someone with no prior knowledge about text classification.

Methodology

TOKENIZATION process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

N-GRAM represents a sequence of words. Depending on the value of the NGRAM setting words are grouped together in groups the size of the value. For instance if the NGRAM value is 1 the text “He went to the store” is organized into the groups “he”, “went”, “to”, “the” and “store”. If the NGRAM value is 2

the text would be grouped into “he went”, “went to”, “to the” and “the store”. In this project, we used only the NGRAM values of 1 and 2 for testing.

REMOVING STOPWORDS A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. Stopwords are available in abundance in any human language. By removing these words, we remove the low-level information from our text in order to give more focus to the important information. In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task. Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.

TF-IDF text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Inverse Document Frequency (IDF). The term frequency is the number of occurrences of a specific term in a document. Inverse Document frequency is the number of documents containing a specific term. Inverse Document frequency indicates how common the term is.

$$w_{i,j} = tf_{i,j} \times idf_i \qquad idf_i = \log(\frac{n}{df_i})$$

tf – term frequency

$$\text{tf} = \frac{frequency\ of\ the\ word\ in\ the\ sentence}{total\ number\ of\ words\ in\ the\ sentence}$$

idf = inverse document frequency

$$\text{idf} = \log(\frac{total\ number\ of\ sentence}{number\ of\ sentence\ contain\ word\ atleast\ once})$$

E.g. Sent_1 : This movie is very scary and long
Sent_2 : This movie is not scary and is slow
Sent_3 : This movie is spooky and good

Doc. Cor. : “This” , “movie” , “is”, “very”, “scary”, “and”, “long”, “not”, “slow”, “spooky”, “good”

The values are shown in below table

	This	movie	is	very	scary	and	long	not	slow	spooky	good
TF	1/7	1/7	1/7	1/7	1/7	1/7	1/7	0	0	0	0
IDF	Log(3/3) = 0	log(3/3) = 0	Log(3/3) = 0	Log(3/1) = 0.48	Log(3/2) =0.176	log(3/3) = 0	Log(3/1) = 0.48	log(3/0) = inf	log(3/0) = inf	log(3/0) = inf	log(3/0) = inf
TF * IDF	0	0	0	0.07	0.025	0	0.07	0	0	0	0

Navie Bayes

Navie Bayes is a Probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Using Bayes theorem, we can find the probability of **class y** happening, for a given **set of features x**.

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

The Naïve Bayes model we used is in our project is commonly called a Multinomial Naïve Bayes model. That means we have multiple classes and takes into account the number of occurrences of an attribute in a document.

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

We find the above value for each class and we assign the class which has highest value for that X

This can be explained by a simple example. Say that we have these two classification categories with a total of 250 documents. A document can be classified as either interesting or not interesting. For each document, we count how many times each word is used and save it in a table.

Classification	Football	Politics	I	Love	Total
Interesting	100	10	50	20	150
Not Interesting	10	100	50	10	100

Table shows that in the 150 articles that are interesting, the word football is mentioned 100 times. If we want to classify a text, we will use the formula in Equation 1 to calculate the probability of the text belonging to each of the classification categories and choose the one with the highest value. Probability that "I love football" belongs to the interesting category :

$$P(Interesting|I, Love, Football)$$

$$= \frac{P(I|Interesting)P(Love|Interesting)P(Football|Interesting)P(Interesting)}{P(I)P(Love)P(Football)}$$

$$= \frac{0.33 * 0.13 * 0.67 * 0.6}{P(evidence)}$$

$$= \frac{0.0172458}{P(evidence)}$$

Probability that “I love football” belongs to the Not interesting category :

$$P(NotInteresting|I, Love, Football)$$

$$= \frac{P(I|NotInteresting)P(Love|NotInteresting)P(Football|NotInteresting)P(NotInteresting)}{P(I)P(Love)P(Football)}$$

$$= \frac{0.5 * 0.1 * 0.1 * 0.4}{P(evidence)}$$

$$= \frac{0.002}{P(evidence)}$$

Since 0.0172458 > 0.002 we have classified the text “i love football” as interesting.

Tools description

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. The Scikit-learn provides **sklearn.naive_bayes.GaussianNB** to implement the Gaussian Naïve Bayes algorithm for classification.

NLTK Library is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.

Dataset Description

The data is from public data set on BBC news articles: D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

- http://mlg.ucd.ie/datasets/bbc.html

We got the cleaned up version exported from

- https://storage.googleapis.com/dataset-uploader/bbc/bbc-text.csv

Our data has

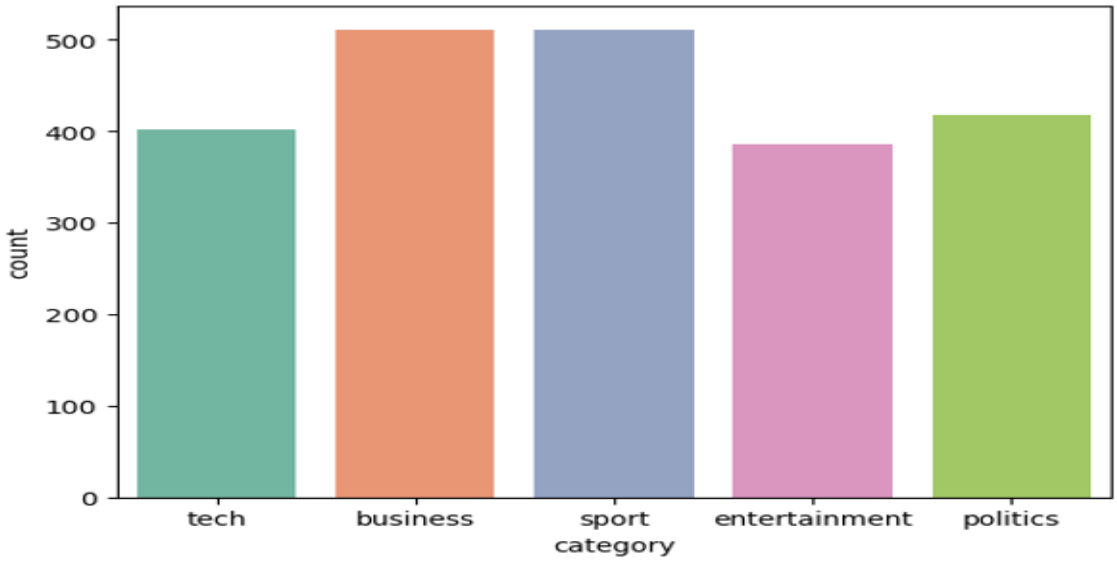
1. 5 categories of news articles
2. 2225 new articles

	category	text
0	tech	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	sport	tigers wary of farrell gamble leicester say ...

Implementation

Exploratory Data Analysis

(1) Bar plot of different categories



(2) Word Cloud Of Different Categories Text

(i) Tech

(ii) Business

(iii) Sport

(iv) Entertainment

(v) Politics

Text Pre-processing

(i) Expand contractions

```
import re
def decontracted(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

(ii) Removing Stop words

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

(iii) Removing Punctuation, URLs, and Lower casing

```
from bs4 import BeautifulSoup
from tqdm import tqdm
preprocessed_reviews = []
for sentance in tqdm(data['text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 2225/2225 [00:01<00:00, 1626.82it/s]
```

```
print("normal text :- " ,data["text"][0])
print("-"*2000)
print("preprocessed text :- ", preprocessed_reviews[0])
```

```
normal text :-  tv future in the hands of viewers with home theatre systems  plasma high-definition tvs  and digital video recorders moving into the living room  the way
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
preprocessed text :-  tv future hands viewers home theatre systems plasma high definition tvs digital video recorders moving living room way people watch tv radically di
```

Vectorization Of Text

(i) Bag Of Words

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (2225, 5000)
the number of unique words including both unigrams and bigrams  5000
```

(ii) TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['abandoned', 'abc', 'ability', 'able', 'able get', 'abn', 'abn amro', 'abroad', 'absence', 'absent']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2225, 7167)
the number of unique words including both unigrams and bigrams  7167
```

Classification Of Text Using Different Machine Learning Algorithms

(i) GaussianNB Classifier

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9101123595505618
```

(ii) Support Vector Classifier

```
from sklearn.svm import SVC
svmclf = SVC(kernel = 'linear',gamma = 'scale', shrinking = False)
svmclf.fit(X_train, y_train)
```

```
SVC(kernel='linear', shrinking=False)
```

```
y_pred_svm = svmclf.predict(X_test)
accuracy_score(y_test, y_pred_svm)
```

```
0.9662921348314607
```

Training GaussianNB Model With Different Training Samples

(i) 500 samples

```
X_500_train, X_500_test, y_500_train, y_500_test = train_test_split(X.iloc[:500,:-2], X.iloc[:500,-1
y_500_pred = gnb.fit(X_500_train, y_500_train).predict(X_500_test)
accuracy_score(y_500_test, y_500_pred)
```

```
0.83
```

(ii) 1000 samples

```
X_1000_train, X_1000_test, y_1000_train, y_1000_test = train_test_split(X.iloc[:1000,:-2], X.iloc[:10
y_1000_pred = gnb.fit(X_1000_train, y_1000_train).predict(X_1000_test)
accuracy_score(y_1000_test, y_1000_pred)
```

```
0.87
```

(iii) 2225 samples

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9101123595505618
```

Training The Model With Different N-gram ranges

(i) N-Grams = 1 or 2

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['abandoned', 'abc', 'ability', 'able', 'able get', 'ab
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2225, 7167)
the number of unique words including both unigrams and bigrams  7167
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9101123595505618
```

(ii) N-Grams = 2 or 3

```
tf_idf_vect_2_3 = TfidfVectorizer(ngram_range=(2,3), min_df=10)
tf_idf_vect_2_3.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect_2_3.get_feature_names()[0:10])
print('='*50)

final_tf_idf_2_3 = tf_idf_vect_2_3.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf_2_3))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_2_3.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf_2_3.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['able get', 'abn amro', 'ac milan', 'academy awards', 'a
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2225, 1802)
the number of unique words including both unigrams and bigrams  1802
```

```
y_2_3_pred = gnb.fit(X_2_3_train, y_2_3_train).predict(X_2_3_test)
accuracy_score(y_2_3_test, y_2_3_pred)
```

```
0.8921348314606742
```

Results and Analysis

(i) SVC performed better in classifying the Textual data compared to NB Classifier.

Model	Accuracy
GNB	91.01
SVC	96.63

(ii) As we increasing our training data size, the accuracy of the model was increasing.

Samples	Accuracy using GNB
500	83.0
1000	87.0
2225	91.01

(iii) Using larger N Grams decreasing the accuracy, But if use 1 or 3, we are getting better accuracy.

N Grams	Accuracy
1 or 2	91.01
2 or 3	89.21

References

1. *J. T. W.B.Cavnar, "N-gram-based text categorization," In Proceedings of SDAIR94, 3rd Annual Symposium on Document Analysis and Information Retrieval, pp. 161–175, 1994.d*
2. *T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," Proceeding ECML '98 Proceedings of the 10th European Conference on Machine Learning, vol. 1398, pp. 137–142, 1998.*
3. *M. Pazzani, "Learning and revising user profiles: The identification of interesting web sites," Machine Learning, vol. 27, no. 3, pp. 313–331, 1997.*