Introduction

The topic of this report is to use Python to implement Multinomial Naïve Bayes algorithm to do text classification.

Naïve Bayes method makes assumption that features of a measurement are independent of each other. Multinomial Naïve Bayes is one of Naïve Bayes that is good at dealing with feature variables that are discrete and conform to a multinomial distribution. In document classification, feature variables are reflected in the number of occurrences of a word, or the TF-IDF value of a word.

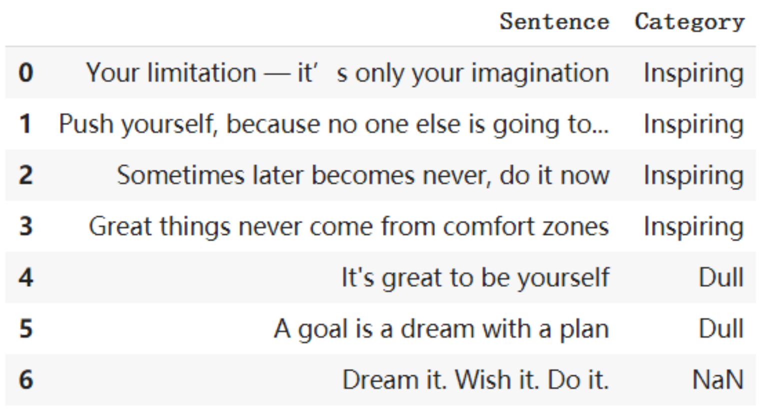
The main emphasis of this presentation is to display the learning process of computer's learning result. The text is first transformed into feature vector, then parameters are estimated from the training data using a simple Bayesian model, and finally these probabilistic parameters are used to make classification predictions for a sample of test texts that are also transformed into integers.

Steps

1. Importing data, remove punctuations and stop words in each sentence
2. Convert words into numeric data: integer encoding
3. Transform data into vectors: vectorization
4. Find out the more significant words in each sentence: TF-IDF Algo
5. Define the training data: Sentences (values) & Categories(labels) and predicting data(test texts)
6. Build the model
7. Performance evaluation

Work & Findings

1. Importing data, remove punctuations and stop words in each sentence

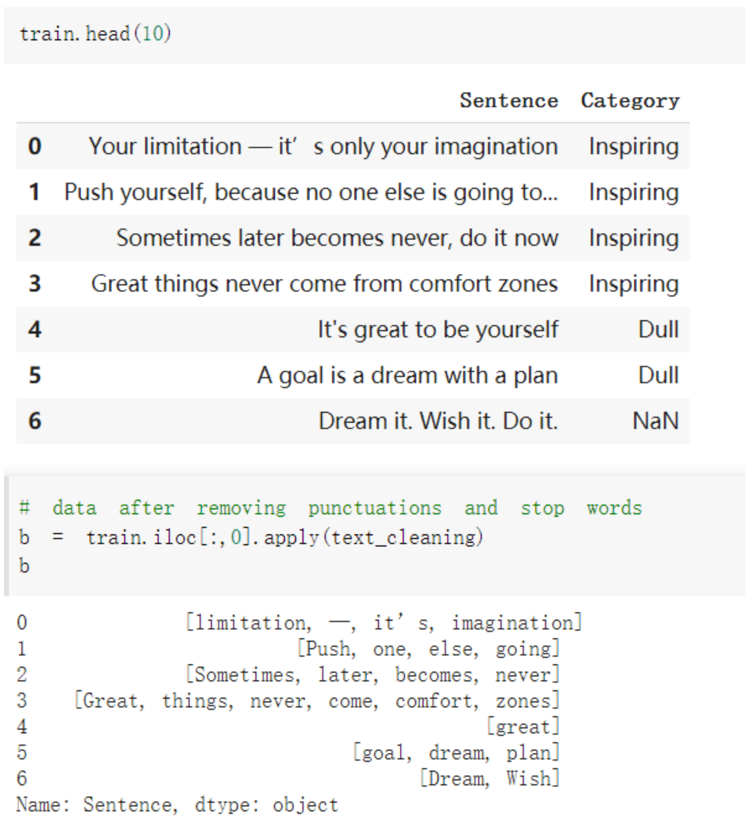


	Sentence	Category
0	Your limitation — it’ s only your imagination	Inspiring
1	Push yourself, because no one else is going to...	Inspiring
2	Sometimes later becomes never, do it now	Inspiring
3	Great things never come from comfort zones	Inspiring
4	It's great to be yourself	Dull
5	A goal is a dream with a plan	Dull
6	Dream it. Wish it. Do it.	NaN

Import string for punctuation library
From nltk.corpus import stopwords for stop words library, define as an set called ‘stopper’
Examples of stoppers:

```
{'a’,
 'about’,
 'above’,
 'after’,
 'again’,
 'against’,
 'ain’,
 'all’,
 'am’,
 'an’,
 'and’,
 'any’,
 'are’,
 'aren’,
 ″aren’t″,
 'as’,
 'at’,
 'be’,
 'because’,
 'been’,
 'before’,
 'being’,
 'below’,
 'between’,
 'both’,
 'but’,
 'by’,
 'can’,
 'couldn’,
 ″couldn’t″,
 'd’,
 'did’,
```

Define a function ‘text_cleaning’ to implement the cleaning actions.



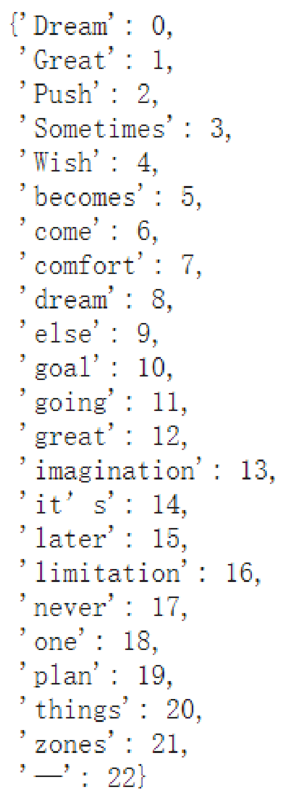
```
train.head(10)
```

	Sentence	Category
0	Your limitation — it’ s only your imagination	Inspiring
1	Push yourself, because no one else is going to...	Inspiring
2	Sometimes later becomes never, do it now	Inspiring
3	Great things never come from comfort zones	Inspiring
4	It's great to be yourself	Dull
5	A goal is a dream with a plan	Dull
6	Dream it. Wish it. Do it.	NaN

```
# data after removing punctuations and stop words
b = train.iloc[:,0].apply(text_cleaning)
b
```

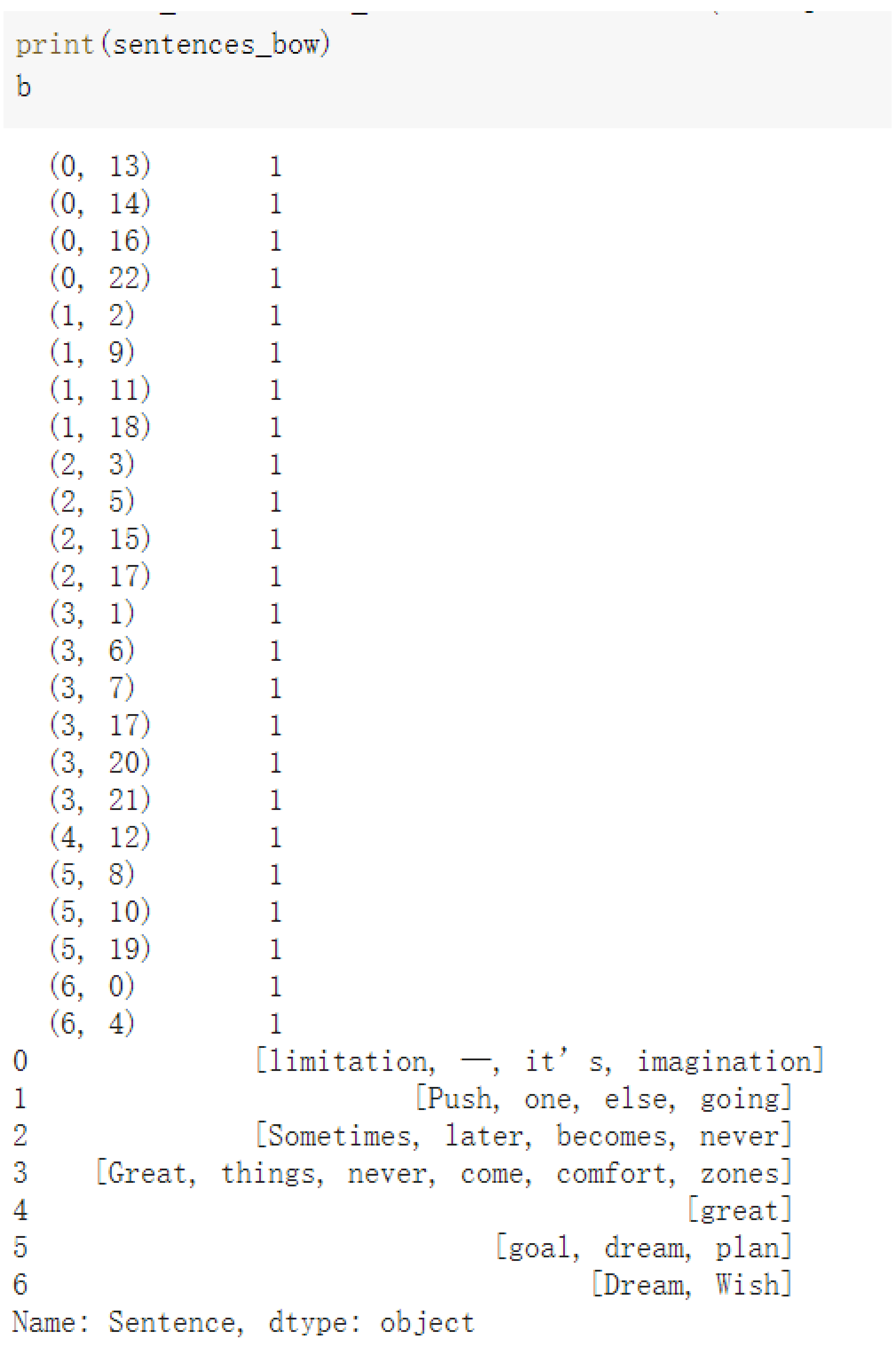
```
0            [limitation, —, it’ s, imagination]
1                      [Push, one, else, going]
2            [Sometimes, later, becomes, never]
3     [Great, things, never, come, comfort, zones]
4                                       [great]
5                             [goal, dream, plan]
6                                 [Dream, Wish]
Name: Sentence, dtype: object
```

2. Convert words into numeric data: integer encoding
Import the feature vector transformation module for text from sklearn.feature_extraction.text and simply assign one number to one word



```
{’Dream’: 0,
 ’Great’: 1,
 ’Push’: 2,
 ’Sometimes’: 3,
 ’Wish’: 4,
 ’becomes’: 5,
 ’come’: 6,
 ’comfort’: 7,
 ’dream’: 8,
 ’else’: 9,
 ’goal’: 10,
 ’going’: 11,
 ’great’: 12,
 ’imagination’: 13,
 ’it’ s’: 14,
 ’later’: 15,
 ’limitation’: 16,
 ’never’: 17,
 ’one’: 18,
 ’plan’: 19,
 ’things’: 20,
 ’zones’: 21,
 ’—’: 22}
```

3. Transform data into vectors



```
print(sentences_bow)
b
```

```
  (0, 13)	1
  (0, 14)	1
  (0, 16)	1
  (0, 22)	1
  (1, 2)	1
  (1, 9)	1
  (1, 11)	1
  (1, 18)	1
  (2, 3)	1
  (2, 5)	1
  (2, 15)	1
  (2, 17)	1
  (3, 1)	1
  (3, 6)	1
  (3, 7)	1
  (3, 17)	1
  (3, 20)	1
  (3, 21)	1
  (4, 12)	1
  (5, 8)	1
  (5, 10)	1
  (5, 19)	1
  (6, 0)	1
  (6, 4)	1
0            [limitation, —, it’s, imagination]
1                     [Push, one, else, going]
2            [Sometimes, later, becomes, never]
3    [Great, things, never, come, comfort, zones]
4                                      [great]
5                          [goal, dream, plan]
6                                [Dream, Wish]
Name: Sentence, dtype: object
```

Explanation:
For the vector (0,13) in variable ‘sentences_bow’, it represents in sentence 0, the first word is assigned with interger 13 as its code, the number following the vector refers to the time frequency of its appears in the whole dataset.

4. Find out the more significant words in each sentence. Use TF-IDF Algo -term frequency- inverse document frequency.

```
print(sentences_tfidf)
```

```
  (0, 22)	0.5
  (0, 16)	0.5
  (0, 14)	0.5
  (0, 13)	0.5
  (1, 18)	0.5
  (1, 11)	0.5
  (1, 9)	0.5
  (1, 2)	0.5
  (2, 17)	0.43218152024617124
  (2, 15)	0.5206467559864713
  (2, 5)	0.5206467559864713
  (2, 3)	0.5206467559864713
  (3, 21)	0.4192570829702294
  (3, 20)	0.4192570829702294
  (3, 17)	0.3480193843688467
  (3, 7)	0.4192570829702294
  (3, 6)	0.4192570829702294
  (3, 1)	0.4192570829702294
  (4, 12)	1.0
  (5, 19)	0.5773502691896257
  (5, 10)	0.5773502691896257
  (5, 8)	0.5773502691896257
  (6, 4)	0.7071067811865476
  (6, 0)	0.7071067811865476
```

5. Define the training set and predicting data

```
train_sentences_tfidf = sentences_tfidf[:-1,]
train_category = train['Category'].iloc[:-1]
pred = sentences_tfidf[-1,]
```

6. Build the model and make prediction

```
# Importing plain Bayesian models from sklearn.navie_bayes
from sklearn.naive_bayes import MultinomialNB
# Initialize the plain Bayesian model using the default configuration
model = MultinomialNB()
# Estimation of model parameters using training data
model.fit(train_sentences_tfidf, train_category)
```

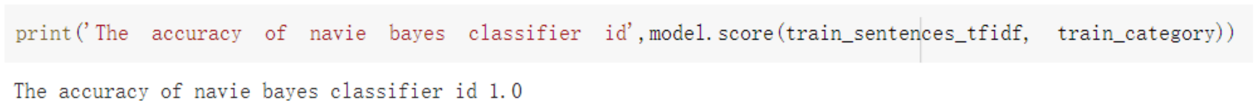
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
# Category prediction for test sample and print it out
print(model.predict(pred))
```

```
['Inspiring']
```

The computer's answer of category of the new sentence is 'Inspiring', which matches our expectation.

7. Performance evaluation of the performance of the plain Bayesian classifier on textual data



```
print('The  accuracy  of  navie  bayes  classifier  id',model.score(train_sentences_tfidf,  train_category))
```

```
The accuracy of navie bayes classifier id 1.0
```

Because there is only one data, our results are 100% accurate. We should increase the number of database entries to improve the training effect of the machine, so that it can identify more samples at once, while ensuring high accuracy.

Discussion

The plain Bayesian model is widely used for massive Internet text classification tasks. The “Naive” assumption that the Naive Bayes classifier makes is that the probability of observing a word is independent of each other. The result is that the “likelihood” is the product of the individual probabilities of seeing each word in the set of target categories.

Step 1. Identify the prerequisites to train a Naive Bayes classifier

Step 2. Computing the Term-Document Matrix (TDM) for each class

Sept 3. Compute frequencies

Step 4. Recall the Naive Bayes rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Step 5. Compute the probability of the word in a sentence belongs to target categories

Due to its strong assumption of feature independence, the estimated parameter size for model prediction is reduced from exponential to linear, which greatly saves memory consumption and computation time. However, the strong assumption that the model cannot take into account the connection between the features when training makes the model perform poorly on other classification tasks where the data features are highly correlated.

References

NLTK 3.5 documentation. Retrieved from, https://www.nltk.org/install.html

N.N (n.d.), sklearn.naive_bayes.MultinomialNB. Retrieved from,

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

Code With Aarohi, (2020 May 29). Retrieved from, https://www.youtube.com/watch?v=oq68P8Kv7nE

Chávez G., (2019 February 28) Implementing a Naive Bayes classifier for text categorization in five steps. Retrieved from, https://towardsdatascience.com/implementing-a-naive-bayes-classifier-for-text-categorization-in-five-steps-f9192cdd54c3