Natural Language Processing ¶

Natural Language Processing(自然语言处理)，或简称为 NLP，是 AI 的子领域，重点放在使计算机能够理解和处理人类语言。
计算机更擅长理解结构化数据(如电子表格和数据库表)，让计算机去理解人类语言(text and language)，那就是——把人类语言（尽可能）结构化。

Teaching machines to understand what is said in spoken and written word is the focus of Natural Language Processing.

For example:

- Whenever you dictate something into your iPhone / Android device that is then converted to text, that’s an NLP algorithm in action.
- You can also use NLP on a text review to predict if the review is a good one or a bad one.
- You can use NLP on an article to predict some categories of the articles you are trying to segment.
- You can use NLP on a book to predict the genre of the book.
- And it can go further, you can use NLP to build a machine translator or a speech recognition system (you use classification algorithms to classify language. Speaking of classification algorithms)

Most of NLP algorithms are classification models, and they include Logistic Regression, Naive Bayes, CART which is a model based on decision trees, Maximum Entropy again related to Decision Trees, Hidden Markov Models which are models based on Markov processes.

A very well-known model in NLP is the Bag of Words model. It is a model used to preprocess the texts to classify before fitting the classification algorithms on the observations containing the texts.

Main NLP library examples:

- Natural Language Toolkit - NLTK (very powerful)
- SpaCy
- Stanford NLP
- OpenNLP

参考:

在机器学习中做任何复杂的事情通常意味着需要建立一条流水线 (pipeline)。这个想法是把你的问题分解成非常小的部分，然后用机器学习来分别解决每个部分，最后通过把几个互相馈送结果的机器学习模型连接起来，就可以解决非常复杂的问题。自然语言处理是如何工作的？一步步教你构建 NLP 流水线 (https://www.jiqizhixin.com/articles/081203)

GitHub上还有一套NLP课程。GitHub NLP Course (https://github.com/yandexdataschool/nlp_course)
课程为期13周，从文本嵌入分类开始，讲到Seq2Seq，再到机器翻译、对话系统、对抗学习等等，内容丰富。入门选手可以考虑。

Import TSV file

Use .tsv file, because no one usually uses 'tab' in review. So it can parse text easily by using tsv.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
# Pandas will know the two columns are separated by 'tab'
# To avoid double quotes "" issue (quoting = 3 to ignore "double quotes")
dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t',quoting=3)
dataset.head()
```

Out[2]:

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

Clean Texts to Prepare Them for the Machine Learning Models

STEP 1: Only keep letters (A-Z) in the review and remove the numbers(unless the numbers can have a significant impact), punctuations(ie.'...'). Removed letters will be replaced by ' '(space).
STEP 2: Putting all the letters in lowercase.
STEP 3: Remove the non-significant words ('the','that','and'...). Keep the words which are relevant to predict the positive or negative reviews.
STEP 4: Stemming (taking the root of the words. ie. love,loved,loves). In order not to have too many words in the end.
STEP 5: Transform list back to string

Clean the 1st review

STEP 1: Only keep letters (A-Z) in the review and remove the numbers(unless the numbers can have a significant impact), punctuations(ie.'...'). Removed letters will be replaced by ' '(space).

In [3]:
```
# Regular Expression 正则表达式
import re

# [^] is what we want to keep
review = re.sub('[^a-zA-Z]',' ',dataset['Review'][0])
review
```

Out[3]:
```
'Wow    Loved this place '
```

STEP 2: Putting all the letters in lowercase.

In [4]:
```
review = review.lower()
review
```

Out[4]:
```
'wow    loved this place '
```

STEP 3: Remove the non-significant words ('the','that','and'...). Keep the words which are relevant to predict the positive or negative reviews.

The **stop words list** contains all the words that are generically irrelevant in any text to predict the category

In [5]:
```
import nltk

# use download function in nltk library
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package stopwords to C:\Users\Cai's
[nltk_data]     family\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

STEP 4: Stemming (taking the root of the words. ie. love,loved,loves). In order not to have too many words in the end.

In [6]:
```
from nltk.stem.porter import PorterStemmer
```

In [7]:
```
# Split a string into each single word in a list
review = review.split()

# Create Stemming object
ps = PorterStemmer()

# go through all the words in a review,
# and remove the words which are in the stop words list
# stopwords has a lot of list in different language, we only focus on english.
# set will be way faster to go through all the different words than list
# (recommend to cast list to set to speed up)
# apply stem function onto each single word
review = [ps.stem(word) for word in review
          if not word in set(stopwords.words('english'))]
review
```

Out[7]:
```
['wow', 'love', 'place']
```

STEP 5: Transform list back to string

In [8]:
```
# use join function to combine all words together to obtain a string
# but we still need separate each word by something,
# otherwise all words will stick together (ie. Iloveyou)
review = ' '.join(review) # join all words together and separated by the space
review
```

Out[8]:
```
'wow love place'
```

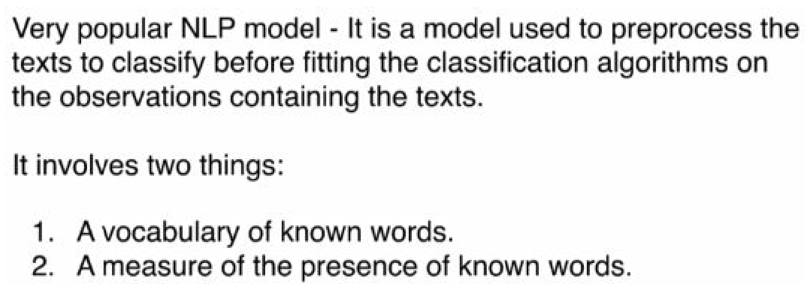
Clean all the reviews

In [9]:
```
# initialize the new list of the 1000 clean reviews
# in NLP, a corpus is a common word and a collection of text at the same type
# (articles, books, web, html pages...)
corpus = []

for i in range(0,dataset.shape[0]):
    review = re.sub('[^a-zA-Z]',' ',dataset['Review'][i])
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    review = [ps.stem(word) for word in review
              if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Create a Bag of Words model

Bag of Words Model



Very popular NLP model - It is a model used to preprocess the texts to classify before fitting the classification algorithms on the observations containing the texts.

It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

To create a Bag of Words model is:

- Take all the different but unique words of the cleaned text
- Create one column for each word
- Put the column into a table, the rows will be each item in corpus, each cell will be the number of word (column) occur in row.

In the case, the table (matrix) will contain a lot of zeros, which is called a sparse matrix (bag of words model).
The fact that we have a lot of zeros is called sparsity.
We try to reduce the sparsity as much as possible when we work with the machine learning model.

In the bag of words, each column can be corresponding to one specific word is one independent variable itself.

We will use the bag of words model to predict the categorical result (ie. positive or negative) in the end. Now, it becomes the Classification machine learning model, so it needs to train the model with those cleaned words and will find the correlation between words and results by itself.

Tokenization

Use CountVectorizer Class to create the huge sparse matrix

CountVectorizer Class also has some input parameters to clean the text:

- lowercase - (True, default) Convert all characters to lowercase before tokenizing.
- token_pattern - Regular expression denoting what constitutes a "token", only used if analyzer == 'word'. The default regexp select tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).
- stop_words - Remove the non-significant words ('the','that','and'...). Keep the words which are relevant to predict the positive or negative reviews.

In the case, we can clean the text directly in the Class CountVectorizer by applying with these parameters.

However, cleaning text manually gives us more control and options and give the possibility to clean fully the text as we want.

In [10]:
```
from sklearn.feature_extraction.text import CountVectorizer

# 可以使用在 CoutVectorizer() 中的 'max_features' 变量
# keep the most frequent words and remove the non-relevant words
# that appear only in one or two times
cv = CountVectorizer(max_features=1500) #保留最常用的1500个words
# To create the huge sparse matrix
# since we need the matrix of features, in order to get the matrix
# we need add .toarray() in the end
X = cv.fit_transform(corpus).toarray()

# Create Dependent variables in order to train the model
# 为了得到 nparray, 需要用 .values
y = dataset['Liked'].values
```

Apply Machine Learning Models onto the Bag of Words model

因为是通过 review 中的 words 来推测 positive or negative。所以是 Classification 问题。

在 NLP 中，最常用的 Classification Model 是 Naive Bayes, Decision Tree 或者 Random Forest classification.

In [11]:
```
# We don't need feature scaling
# because the sparse matrix has mostly zeros and a few ones
# and a very few twos or threes
# so feature scaling is unnecessary.

## Feature Scaling
#from sklearn.preprocessing import StandardScaler
#sc = StandardScaler()
#X = sc.fit_transform(X)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20,random_
state=0)
```

In [12]:
```
# To evaluate the model performance

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.metrics import classification_report
```

Naive Bayes Classification

In [13]:
```python
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB().fit(X_train,y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_nb = confusion_matrix(y_test,y_pred)
print(cm_nb)

print('Accuracy = ', accuracy_score(y_test,y_pred))
print('Prcision = ', precision_score(y_test,y_pred))
print('Recall = ', recall_score(y_test,y_pred))
print('F1 Score = ', f1_score(y_test,y_pred,average='weighted'))

print(classification_report(y_test, y_pred))
```

```
[[55 42]
 [12 91]]
Accuracy =  0.73
Prcision =  0.6842105263157895
Recall =  0.883495145631068
F1 Score =  0.722465894997933
              precision    recall  f1-score   support

           0       0.82      0.57      0.67        97
           1       0.68      0.88      0.77       103

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.72       200
weighted avg       0.75      0.73      0.72       200
```

Decision Tree Classification

In [14]:
```
# Fitting Decision Tree to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy',random_state=0).fit(X_
train,y_train)
y_pred = classifier.predict(X_test)
cm_dt = confusion_matrix(y_test,y_pred)
print(cm_dt)

print('Accuracy = ', accuracy_score(y_test,y_pred))
print('Prcision = ', precision_score(y_test,y_pred))
print('Recall = ', recall_score(y_test,y_pred))
print('F1 Score = ', f1_score(y_test,y_pred,average='weighted'))

print(classification_report(y_test, y_pred))
```

```
[[74 23]
 [35 68]]
Accuracy =  0.71
Prcision =  0.7472527472527473
Recall =  0.6601941747572816
F1 Score =  0.7094775297767992
              precision    recall  f1-score   support

           0       0.68      0.76      0.72        97
           1       0.75      0.66      0.70       103

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.71      0.71      0.71       200
```

Random Forest Classification

In [15]:
```
from sklearn.ensemble import RandomForestClassifier as RFC
# n_estimators: 将使用多少 decision trees 来组成 random forest (default = 10)
classifier = RFC(n_estimators=10,criterion='entropy',random_state=0).fit(X_tra
in,y_train)
y_pred = classifier.predict(X_test)
cm_rf = confusion_matrix(y_test,y_pred)
print(cm_rf)

print('Accuracy = ', accuracy_score(y_test,y_pred))
print('Prcision = ', precision_score(y_test,y_pred))
print('Recall = ', recall_score(y_test,y_pred))
print('F1 Score = ', f1_score(y_test,y_pred,average='weighted'))

print(classification_report(y_test, y_pred))
```

```
[[87 10]
 [46 57]]
Accuracy =  0.72
Prcision =  0.8507462686567164
Recall =  0.5533980582524272
F1 Score =  0.7122659846547316
              precision    recall  f1-score   support

           0       0.65      0.90      0.76        97
           1       0.85      0.55      0.67       103

    accuracy                           0.72       200
   macro avg       0.75      0.73      0.71       200
weighted avg       0.76      0.72      0.71       200
```

Evaluate the Performance

Accuracy is not enough, so we should also look at other performance metrics like

- Precision (measuring exactness)
- Recall (measuring completeness)
- F1 Score (compromise between Precision and Recall)

Please find below these metrics formulas (TP = # True Positives, TN = # True Negatives, FP = # False Positives, FN = # False Negatives):

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $F1Score = 2\frac{Precision*Recall}{Precision+Recall}$

In [16]:
```
def evaluate_performance(cm): # cm is confusion_matrix
    TP = cm[0,0]
    TN = cm[1,1]
    FP = cm[0,1]
    FN = cm[1,0]
    accuracy = (TP + TN) / (TP + TN + FP + FN)
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    F1_score = 2 * (precision * recall) / (precision + recall)
    print('Accuracy = ', accuracy)
    print('Prcision = ', precision)
    print('Recall = ', recall)
    print('F1 Score = ', F1_score)
```

Navie Bayes

In [17]:
```
evaluate_performance(cm_nb)
```

```
Accuracy =  0.73
Prcision =  0.5670103092783505
Recall =  0.8208955223880597
F1 Score =  0.6707317073170731
```

Decision Tree

In [18]:
```
evaluate_performance(cm_dt)
```

```
Accuracy =  0.71
Prcision =  0.7628865979381443
Recall =  0.6788990825688074
F1 Score =  0.7184466019417477
```

Random Forest

In [19]:
```
evaluate_performance(cm_rf)
```

```
Accuracy =  0.72
Prcision =  0.8969072164948454
Recall =  0.6541353383458647
F1 Score =  0.7565217391304349
```

In [ ]:
```

```