Statistical Machine Learning with Python Week #3

Ching-Shih Tsou (Ph.D.), Distinguished Prof. at the Department of Mechanical Engineering/Director of the Center for Artifical Intelligence & Data Science, Ming Chi University of Technology

September/2020 at MCUT

Example – filtering mobile phone spam with the Naive Bayes algorithm

We'll begin by importing the CSV data and saving it in a data frame

We see that the sms_raw data frame includes 5,559 total SMS messages with two features: type and text. The SMS type has been coded as either ham or spam. Additionally, we see that 747 (about 13 percent) of SMS messages in our data were labeled as spam, while the others were labeled as ham

sms_spam.csv is a collection of 5559 SMS messages, The text field contains the text message content

The frequency distribution of the type field shows that 4,812 (approximately 87%) of them are normal The 747 was spam (about 13 percent)

```
import pandas as pd
# Import mobile text message data set
sms_raw = pd.read_csv("_data/sms_spam.csv")
# type Spam or normal text. text:SMS text content
print(sms_raw.dtypes)
```

```
## type    object
## text    object
## dtype: object
```

```
#type frequency distribution, ham is the majority,But not excessively unbalanc
ed
print(sms_raw['type'].value_counts()/len(sms_raw['type']))
```

```
## ham     0.865623
## spam    0.134377
## Name: type, dtype: float64
```

Text field English text content is available through the Python natural language processor NLTK is used to clean up the word segmentation and corpus

```
# Python natural language processing toolset(Natural Language ToolKit)
import nltk
nltk.download('punkt')
# String derivation completes the participle
```

```
## True
##
## [nltk_data] Downloading package punkt to /Users/Vince/nltk_data...
## [nltk_data]   Package punkt is already up-to-date!
```

```
token_list0 = [ nltk.word_tokenize(txt) for txt in sms_raw['text']]
print(token_list0[3][1:7])
```

```
## ['4', 'STAR', 'Ibiza', 'Holiday', 'or', '£10,000']
```

In general, corpus cleaning includes: • turn lowercase • Remove stop words • Remove punctuation • Remove Numbers • Remove possible blank words • Morphological reduction (lemmatization)

```
#String derivation completes the transformation to lowercase(Ibiza to ibiza)
token_list1 = [[word.lower() for word in doc]
for doc in token_list0]
print(token_list1[3][1:7])
```

```
## ['4', 'star', 'ibiza', 'holiday', 'or', '£10,000']
```

```
# String derivation removes stop words
from nltk.corpus import stopwords
nltk.download('stopwords')
# 179 English stop words
```

```
## True
##
## [nltk_data] Downloading package stopwords to /Users/Vince/nltk_data...
## [nltk_data]   Package stopwords is already up-to-date!
```

```
print(len(stopwords.words('english')))
```

```
## 179
```

```
#The stop word or has been removed
token_list2 = [[word for word in doc if word not in
stopwords.words('english')] for doc in token_list1]
print(token_list2[3][1:7])
```

```
## ['4', 'star', 'ibiza', 'holiday', '£10,000', 'cash']
```

```
# String derivation removes punctuation marks
import string
token_list3 = [[word for word in doc if word not in
string.punctuation] for doc in token_list2]
print(token_list3[3][1:7])
```

```
## ['4', 'star', 'ibiza', 'holiday', '£10,000', 'cash']
```

```
# String derivation removes all Numbers , '4' is missing
token_list4 = [[word for word in doc if not word.isdigit()]
for doc in token_list3]
print(token_list4[3][1:7])
```

```
## ['star', 'ibiza', 'holiday', '£10,000', 'cash', 'needs']
```

```
# Three-layer nested sequences derive situations in which digits or punctuatio
n marks are removed from a character
token_list5 = [[''.join([i for i in word if not i.isdigit() and i not in strin
g.punctuation]) for word in doc] for doc in token_list4]
```

£10,000 to £

```
print(token_list5[3][1:7])
```

```
## ['star', 'ibiza', 'holiday', '£', 'cash', 'needs']
```

Finally,list(filter(None, doc)) filters out all null tokens, and WordNetLemmatizer() was used for word reduction

```
# String derivation removes empty elements
token_list6 =[list(filter(None, doc)) for doc in token_list5]
print(token_list6[3][1:7])
```

```
## ['star', 'ibiza', 'holiday', '£', 'cash', 'needs']
```

```
# Load the "WordNet" lexical restore library of "Nltk.stem"
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
# Declaring the Word reductor
```

```
## True
##
## [nltk_data] Downloading package wordnet to /Users/Vince/nltk_data...
## [nltk_data]   Package wordnet is already up-to-date!
```

```
lemma = WordNetLemmatizer()
# String derivation Complete word reduction ('needs' to 'need')
token_list6 = [[lemma.lemmatize(word) for word in doc]
for doc in token_list6]
print(token_list6[3][1:7])
```

```
## ['star', 'ibiza', 'holiday', '£', 'cash', 'need']
```

Once you've cleared everything, reassemble the tokens for each text into a short paragraph using the join() method of white space characters

```
# The concatenation derivation completes the concatenation of each word
# The join() method concatenates the separate characters in each SMS doc
token_list7 = [' '.join(doc) for doc in token_list6]
print(token_list7[:2])
```

```
## ['hope good week checking', 'kgive back thanks']
```

```
import pandas as pd
# Load the word frequency calculation and DTM construction category from the "
Feature_extraction" module
from sklearn.feature_extraction.text import CountVectorizer
# Declaring the empty mold
vec = CountVectorizer()
# The message is passed in to fit the real mode and converted to DTM sparse ma
trix X
X = vec.fit_transform(token_list7)
# Scipy suite sparse matrix category
print(type(X))
```

```
## <class 'scipy.sparse.csr.csr_matrix'>
```

```
# A way to store word frequency in sparse matrices:(columns, rows)Word frequen
cy
print(X[:2])# The word frequency of the first two messages
```

```
##   (0, 2945)  1
##   (0, 2604)  1
##   (0, 7217)  1
##   (0, 1073)  1
##   (1, 3426)  1
##   (1, 487)   1
##   (1, 6516)  1
```

```
# X goes to the regular matrix(X.toarray()) and build the Pandas data frame
sms_dtm = pd.DataFrame(X.toarray(),
columns=vec.get_feature_names())
#structure of 5559 columns() ,7612  row(word)
print(sms_dtm.shape)
```

```
## (5559, 7612)
```

```
print(len(vec.get_feature_names())) # There are 7,612 words.
```

```
## 7612
```

```
print(vec.get_feature_names()[300:305])
```

```
## ['apology', 'app', 'apparently', 'appeal', 'appear']
```

```
# Part of DTM
print(sms_dtm.iloc[4460:4470, 300:305])
```

```
##       apology  app  apparently  appeal  appear
## 4460        0    1           0       0       0
## 4461        0    0           0       0       0
## 4462        0    0           0       0       0
## 4463        0    0           0       0       0
## 4464        0    0           0       0       0
## 4465        0    0           0       0       0
## 4466        0    0           0       0       0
## 4467        0    0           0       0       0
## 4468        0    0           0       0       0
## 4469        1    0           0       0       0
```

Before the training model, the training and test data set is established. The segmented objects include the original data box, DTM matrix, and the cleaned corpus were segmented and the label category distribution of the two subsets was confirmed Similar to the distribution of the original data set

```
# Training and test sets are segmented (sms_raw, sms_dtm, token_list6)
sms_raw_train = sms_raw.iloc[:4170, :]
sms_raw_test = sms_raw.iloc[4170:, :]
sms_dtm_train = sms_dtm.iloc[:4170, :]
sms_dtm_test = sms_dtm.iloc[4170:, :]
token_list6_train = token_list6[:4170]
token_list6_test = token_list6[4170:]
# Check the distribution of subsets
print(sms_raw_train['type'].value_counts()/
len(sms_raw_train['type']))
```

```
## ham     0.864748
## spam    0.135252
## Name: type, dtype: float64
```

```
print(sms_raw_test['type'].value_counts()/
len(sms_raw_test['type']))
```

```
## ham     0.868251
## spam    0.131749
## Name: type, dtype: float64
```

Text cloud

A word cloud is a way to visually depict the frequency at which words appear in text data. The cloud is composed of words scattered somewhat randomly around the figure. Words appearing more often in the text are shown in a larger font, while less common terms are shown in smaller fonts. This type of figures grew in popularity recently, since it provides a way to observe trending topics on social media websites.

Word Cloud is a commonly used visual model of text data. We first draw a text cloud for all the training corpus, and then make a text cloud for the garbage in the training set and the normal text message set respectively.

```
tokens_train = [token for doc in token_list6_train
for token in doc]
print(len(tokens_train))
```

```
## 38103
```

```
#WordCloud(), statistical word frequency should be combined across all words
tokens_train_spam = [token for is_spam, doc in
zip(sms_raw_train['type'] == 'spam' , token_list6_train)
if is_spam for token in doc]
# The logical value index is combined with the ZIP () function,
tokens_train_ham = [token for is_ham, doc in
zip(sms_raw_train['type'] == 'ham' , token_list6_train)
if is_ham for token in doc]
# Take out the normal message
str_train = ','.join(tokens_train)
str_train_spam = ','.join(tokens_train_spam)
str_train_ham = ','.join(tokens_train_ham)
```

1. The text cloud is still drawn in the same way that the Python language normally runs, first loaded Suite
2. Next, set the drawing specification
3. Then the data (str_train, STR_train_spam , Str_train_ham) is passed into the generate() method to produce text cloud objects
4. Finally, the Matplotlib. pyplot module can draw and save the text cloud of Figure 5.7, 5.8 and 5.9

It can be seen from the results of the text cloud that the words appearing in the garbage messages in the training corpus are indeed different from those appearing in the overall training set and normal messages

```
# Python text cloud suite
from wordcloud import WordCloud
## p.s if not have "wordcloud" library you need to command "pip install wordcl
oud"
# Declare text cloud objects (max_words default to 200)
wc_train = WordCloud(background_color="white",
prefer_horizontal=0.5)
# Incoming data statistics, and production of text cloud
wc_train.generate(str_train)
# call the imshow() method from  matplotlib.pyplot module to plot
```

```
## <wordcloud.wordcloud.WordCloud object at 0x7feb804aa610>
```

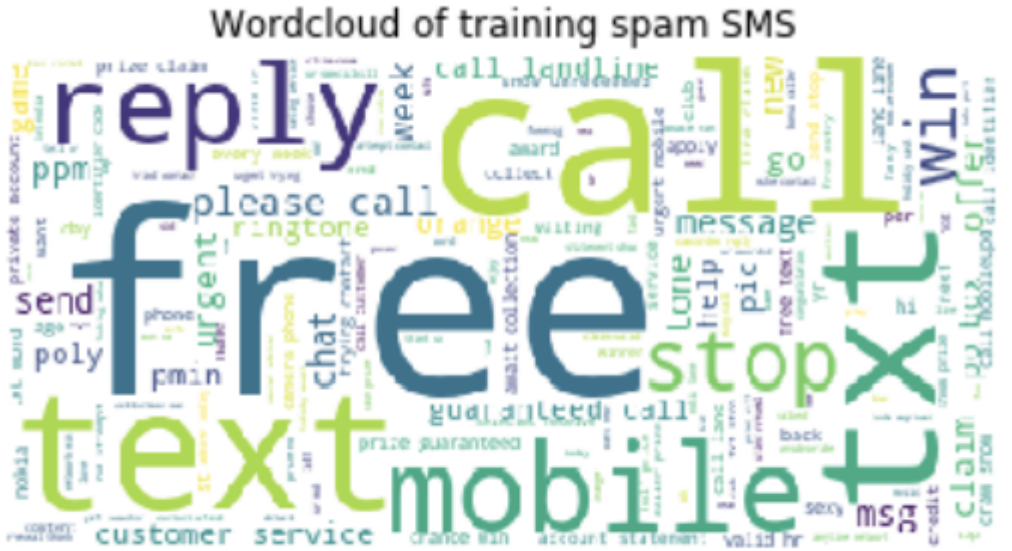
```
import matplotlib.pyplot as plt
plt.imshow(wc_train)
plt.axis("off")
# plt.show()
# plt.savefig('wc_train.png')
```

```
## (-0.5, 399.5, 199.5, -0.5)
```

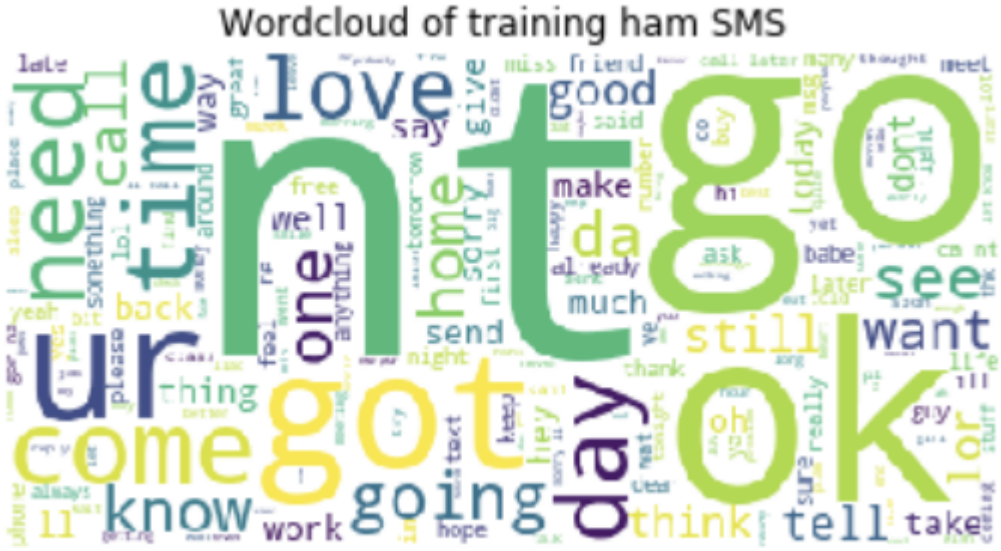
```
# polynomial naive bayes model
from sklearn.naive_bayes import MultinomialNB
# Model definition, adaptation and prediction
clf = MultinomialNB()
```

5.7 Text cloud for SMS training corpus

5.8 Spam text cloud in training Corpus



5.9 The normal text cloud in the training corpus




```
clf.fit(sms_dtm_train, sms_raw_train['type'])
```

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

```
train = clf.predict(sms_dtm_train)
print(" The training set accuracy rate is {}".format(sum(sms_raw_train['type']
== train)/len(train)))
```

```
##  The training set accuracy rate is 0.9887290167865708
```

```
pred = clf.predict(sms_dtm_test)
print(" The test set accuracy is {}".format(sum(sms_raw_test['type'] == pred)/
len(pred)))
```

```
##  The test set accuracy is 0.9690424766018718
```

```
# The number of samples used for training
print(clf.class_count_)
```

```
## [3606.  564.]
```

```
# A cross list of two classes with 7612 attributes
print(clf.feature_count_)
```

```
## [[1. 2. 1. ... 1. 0. 0.]
##  [0. 0. 0. ... 0. 0. 1.]]
```

```
print(clf.feature_count_.shape)
```

```
## (2, 7612)
```

```
#Log value of conditional probability Pr[x_i|y] for each attribute
print(clf.feature_log_prob_[:, :4])
```

```
## [[ -9.77004187  -9.36457676  -9.77004187 -10.46318905]
##  [ -9.64212279  -9.64212279  -9.64212279  -9.64212279]]
```

```
print(clf.feature_log_prob_.shape)
```

```
## (2, 7612)
```

Finishing

Call cross_val_score() to calculate the performance score for each cross-validation model fit, and print out the correct rate and Standard Error (SE) for each training.

$$SE = \frac{s}{\sqrt{n-1}},$$

s = standard deviation N = sample size

The difference between a standard error and a standard deviation is that SE estimates variability between samples, while S estimates variability between samples It measures variation within a single sample

```
import numpy as np
# Load the Sklearn cross-validation function for model selection
from sklearn.model_selection import cross_val_score, KFold
from scipy.stats import sem
# The custom k value is the performance calculation function of cross-validati
on model
def evaluate_cross_validation(clf, X, y, K):
    # Create a "K" cross validation iterator(iterator), For the sharding of X
and y
    cv = KFold(n_splits=K, shuffle=True, random_state=0)
    scores = cross_val_score(clf, X, y, cv=cv)
    print("{} results of cross-validation are as follows:\n{}".format(K, score
s))
    tmp = " Average accuracy:{0:.3f}(+/- Standard error {1:.3f})"
    print(tmp.format(np.mean(scores), sem(scores)))
evaluate_cross_validation(clf, sms_dtm, sms_raw['type'], 5)
```

```
## 5 results of cross-validation are as follows:
## [0.9721223  0.96223022 0.96942446 0.96852518 0.97029703]
##  Average accuracy:0.969(+/- Standard error 0.002)
```