Text Classification

Text Classification assigns one or more classes to a document according to their content. Classes are selected from a previously established classes.

Types of Text Classification Problems

- Binary classification like spam filtering (NOT-SPAN, SPAM) or simple sentiment analysis (POSITIVE, NEGATIVE)
- Multiple class classification like selecting one category among several alternatives - movie genre classification (thriller, terror, romantic, etc ...)

Data Set For SPAM Classification:

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according being ham or spam.

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.
Data is collected from http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...
10	ham	I'm gonna be home soon and i don't want to tal...

Text Cleaning:

Removal of Stop-words: When data analysis needs to be data driven at the word level, the commonly occurring words (stop-words) should be removed. One can either create a long list of stop-words or one can use predefined language specific libraries.

Removal of Punctuations: The commonly occurring Punctuations like '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~' should be removed while we are analysing text. This Punctuations not more descriptive.

Stemming: A stemming algorithm reduces the words "working", "worked" to the root word, "work".

```
import string
from nltk.corpus import stopwords
import nltk
from nltk.stem import PorterStemmer
def text_Cleaning(mess):
    non_punc ="".join([char for char in mess if char not in string.punctuation])  # REMOVING PUNCTUATION
    tokens = nltk.word_tokenize(non_punc)
    non_stop = [word for word in tokens if word.lower() not in stopwords.words('english')] # REMOVING STOP-WORDS
    st = PorterStemmer()
    clean_txt = " ".join([st.stem(w) for w in non_stop])  #STEMMING
    clean_txt = clean_txt.split()
    return clean_txt
```

```
messages['message'].head(10).apply(text_Cleaning)
```

```
0    [Go, jurong, point, crazi, avail, bugi, n, gre...
1                         [Ok, lar, joke, wif, u, oni]
2    [free, entri, 2, wkli, comp, win, FA, cup, fin...
3        [U, dun, say, earli, hor, U, c, alreadi, say]
4    [nah, dont, think, goe, usf, live, around, tho...
5    [freemsg, hey, darl, 3, week, word, back, Id, ...
6    [even, brother, like, speak, treat, like, aid,...
7    [per, request, mell, mell, oru, minnaminungint...
8    [winner, valu, network, custom, select, receiv...
9    [mobil, 11, month, U, R, entitl, updat, latest...
Name: message, dtype: object
```

After removing of stop-words , punctuations and then stemming of text we get the clean text.

Our main issue with our data is that it is all in text format (strings). The classification algorithms that we have learned about so far will need some sort of numerical feature vector in order to perform the classification task.

we need to vectorise the text message then only we can apply any classifier.

Feature Extraction

Vectorisation of Words:

Now we'll convert each message, represented as a list of tokens above, into a vector that machine learning models can understand.

We'll do that in three steps using the bag-of-words model:The bag-of-words model is commonly used in methods of document classification **where the (frequency of) occurrence of each word is used as a feature for training a classifier.**

```
from sklearn.feature_extraction.text import CountVectorizer
word_vector = CountVectorizer(analyzer=text_Cleaning).fit(messages['message'])
```

```
word_vector.vocabulary_
```

```
{'Go': 960,
 'jurong': 4228,
 'point': 5713,
 'crazi': 2366,
 'avail': 1463,
 'bugi': 1869,
 'n': 5047,
 'great': 3532,
 'world': 7974,
 'la': 4371,
 'e': 2804,
 'buffet': 1867,
 'cine': 2147,
 'got': 3495,
 'amor': 1274,
 'wat': 7772,
 'Ok': 1001,
 'lar': 4406,
 'joke': 4194,
```

```
text_Cleaning(messages['message'][6])
```

```
['even', 'brother', 'like', 'speak', 'treat', 'like', 'aid', 'patent']
```

```
message7 = messages['message'][6]
print(bow_transformer.transform([message7]))
```

```
  (0, 1197)	1
  (0, 1839)	1
  (0, 2962)	1
  (0, 4494)	2
  (0, 5540)	1
  (0, 6728)	1
  (0, 7410)	1
```

```
bow_transformer.get_feature_names()[4494]
```

```
'like'
```

This means that there are seven unique words in message number 7 (after removing common stop words). One of them appear twice, the rest only once.

TF-IDF stands for *term frequency-inverse document frequency:* The tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

TF: Term Frequency:

which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalisation:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency:

which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log(Total number of documents / Number of documents with term t in it).

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf7 = tfidf_transformer.transform(bow_message7)
print(tfidf7)
```

```
  (0, 7410)	0.34430022595872367
  (0, 6728)	0.31377588072154017
  (0, 5540)	0.46387575652685364
  (0, 4494)	0.43094677413459165
  (0, 2962)	0.266191993050295
  (0, 1839)	0.3393506736692894
  (0, 1197)	0.4428195507968606
```

Create Model

Train Test Split: Dividing original data into to sets one is for train and another for test.

```
from sklearn.model_selection import train_test_split

# Divides Train Data 70% and test Data 30%
msg_train, msg_test, label_train, label_test = train_test_split(messages['message'], messages['label'], test_size=0.3)
print(len(msg_train), len(msg_test), len(msg_train) + len(msg_test))

3900 1672 5572
```

And the parameters are feature-set , target, test_size. Here test_size = 0.3 is 30% data is separated for testing.

Pipeline

Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods. The final estimator only needs to implement fit.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_Cleaning)),  # strings to token integer counts
    ('classifier', MultinomialNB()),  # Naive Bayes classifier
])
```

In this pipeline for features extraction we used CountVectorizer () and for class-action we used MultinomialNB() which Naive_Bayes.

How Naive Bayes algorithm works?

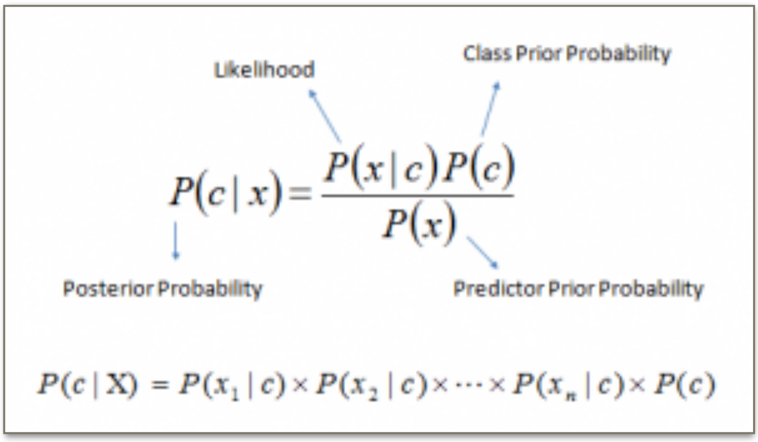
Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like e.g: (XXXX) probability = 0.29 and probability of spam is 0.64.

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

$$p(C_k/x_1, x_2, x_3 \ldots .)$$

P is the probability of class C_k where $x_1, x_2, x_3 \ldots .$ are the features.



$p(x_1/c)$ is the probability of x1 feature when class is given.
And $p(c)$ is probability of the class with out looking to the data.

Train and Test the Model

fit the training set to the pipelined model . After training predict for test samples.

```
pipeline.fit(msg_train,label_train)
predictions = pipeline.predict(msg_test)
```

```
print(predictions)
```

```
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

Evaluation

```
# PREDICTION REPORT GENERATE
from sklearn.metrics import classification_report
print(classification_report(predictions,label_test))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.98      0.99      1447
       spam       0.89      0.97      0.93       225

avg / total       0.98      0.98      0.98      1672
```

Accuracy score:

Pression = TP / TP+FP

Recall = TP / FN +TP

Accuracy = TP+TN / TP+TN+FP+FN

TP = True positive = correctly identified
FP = False positive = incorrectly identified
TN = True negative = correctly rejected
FN = False negative = incorrectly rejected

RESULTS:

Feature extraction = Count vectoriser
Classifier = Multinomial Naive-Bayes

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_Cleaning)),  # strings to token integer counts
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])

pipeline.fit(msg_train,label_train)
predictions = pipeline.predict(msg_test)

# PREDICTION REPORT GENERATE
from sklearn.metrics import classification_report
print(classification_report(predictions,label_test))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.98      0.99      2428
       spam       0.90      0.97      0.93       358

avg / total       0.98      0.98      0.98      2786
```

Feature extraction = TF-IDF vectoriser
Classifier = Multinomial Naive-Bayes

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_Cleaning)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])

pipeline.fit(msg_train,label_train)
predictions = pipeline.predict(msg_test)

# PREDICTION REPORT GENERATE
from sklearn.metrics import classification_report
print(classification_report(predictions,label_test))
```

```
/Users/dineshmaharana/anaconda3/lib/python3.6/site-packages/sklearn/feature_extraction/text.p
onversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. I
ated as `np.float64 == np.dtype(float).type`.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

	precision	recall	f1-score	support
ham	1.00	0.95	0.97	2540
spam	0.64	1.00	0.78	246
avg / total	0.97	0.95	0.96	2786

Feature extraction = Count vectoriser
Classifier = SGDClassifier(Stocastic Gradient Descent)

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_Cleaning)),  # strings to token integer counts
    ('classifier', SGDClassifier()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])

pipeline.fit(msg_train,label_train)
predictions = pipeline.predict(msg_test)

# PREDICTION REPORT GENERATE
from sklearn.metrics import classification_report
print(classification_report(predictions,label_test))
```

```
/Users/dineshmaharana/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gra
ning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_gr
in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, m
x_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

	precision	recall	f1-score	support
ham	1.00	0.98	0.99	2434
spam	0.89	0.97	0.92	352
avg / total	0.98	0.98	0.98	2786

Feature extraction = TF-IDF vectoriser
Classifier = SGDClassifier(Stocastic Gradient Descent)

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_Cleaning)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', SGDClassifier()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])

pipeline.fit(msg_train,label_train)
predictions = pipeline.predict(msg_test)

# PREDICTION REPORT GENERATE
from sklearn.metrics import classification_report
print(classification_report(predictions,label_test))
```

```
/Users/dineshmaharana/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_
ning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_
in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None,
x_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/dineshmaharana/anaconda3/lib/python3.6/site-packages/sklearn/feature_extraction/text.p
onversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. I
ated as `np.float64 == np.dtype(float).type`.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

```
             precision    recall  f1-score   support

        ham       1.00      0.99      0.99      2426
       spam       0.91      0.97      0.94       360

avg / total       0.98      0.98      0.98      2786
```