In [3]:
```
cat 1.txt
```

```
This is a story about cats
our feline pets
Cats are furry animals
```

In [4]:
```
cat 2.txt
```

```
This story is about surfing
Catching waves is fun
Surfing is a popular water sport
```

In [7]:
```
# Build Vocabulary
vocab = {}
i = 1

with open('1.txt') as f:
    x = f.read().lower().split()

for word in x:
    if word not in vocab:
        vocab[word] = i
        i += 1

print(vocab)
```

```
{'this': 1, 'is': 2, 'a': 3, 'story': 4, 'about': 5, 'cats': 6, 'our':
7, 'feline': 8, 'pets': 9, 'are': 10, 'furry': 11, 'animals': 12}
```

In [8]:
```
with open('2.txt') as f:
    x = f.read().lower().split()

for word in x:
    if word not in vocab:
        vocab[word] = i
        i += 1

print(vocab)
```

```
{'this': 1, 'is': 2, 'a': 3, 'story': 4, 'about': 5, 'cats': 6, 'our':
7, 'feline': 8, 'pets': 9, 'are': 10, 'furry': 11, 'animals': 12, 'sur
fing': 13, 'catching': 14, 'waves': 15, 'fun': 16, 'popular': 17, 'wat
er': 18, 'sport': 19}
```

Feature Extraction

In [9]:
```
# Create an empty vector with space for each word in the vocabulary
one = ['1.txt'] + [0] * len(vocab)

with open('1.txt') as f:
    x = f.read().lower().split()

# Map the frequency of each word in 1.txt to the vector one
for word in x:
    one[vocab[word]] += 1

print(one)
```

```
['1.txt', 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

In [10]:
```
# create an empty vector with space for each word in the vocabulary:
two = ['2.txt'] + [0]*len(vocab)

with open('2.txt') as f:
    x = f.read().lower().split()

# Map the frequency of word in 2.txt to vector two
for word in x:
    two[vocab[word]] += 1

print(two)
```

```
['2.txt', 1, 3, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1]
```

In [13]:
```
# Compare the two vectors
print(f'{one}\n{two}')
```

```
['1.txt', 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
['2.txt', 1, 3, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1]
```

Feature Extraction From Text

In [16]:
```
import pandas as pd
import numpy as np

df = pd.read_csv('../TextFiles/smsspamcollection.tsv', sep='\t')
df.head()
```

Out[16]:

	label	message	length	punct
0	ham	Go until jurong point, crazy.. Available only ...	111	9
1	ham	Ok lar... Joking wif u oni...	29	6
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	6
3	ham	U dun say so early hor... U c already then say...	49	6
4	ham	Nah I don't think he goes to usf, he lives aro...	61	2

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
label      0
message    0
length     0
punct      0
dtype: int64
```

In [18]:
```
df['label'].value_counts()
```

Out[18]:
```
ham     4825
spam     747
Name: label, dtype: int64
```

In [19]:
```
df['label'].value_counts(normalize = True)
```

Out[19]:
```
ham     0.865937
spam    0.134063
Name: label, dtype: float64
```

In [20]:
```
df['label'].value_counts(normalize = True)*100
```

Out[20]:
```
ham     86.593683
spam    13.406317
Name: label, dtype: float64
```

In [21]:
```
from sklearn.model_selection import train_test_split

X = df['message']
y = df['label']
```

In [22]:
```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
```

In [23]:
```
# Scikit-learn -- CountVectorizer

from sklearn.feature_extraction.text import CountVectorizer
```

In [24]:
```
count_vect = CountVectorizer()
```

In [25]:
```
# Fit vectorizer to the data(build vocab and count the number of words)
# count_vect.fit(X_train)
# Transform the fitted data
# X_train_counts = count_vect.transform(X_train)

# Fit and transform can be done in one step
X_train_counts = count_vect.fit_transform(X_train)
```

In [26]:
```
X_train_counts.shape
```

Out[26]:
```
(3733, 7082)
```

In [27]:
```
X_train.shape
```

Out[27]:
```
(3733,)
```

In [28]:
```
X_train_counts
```

Out[28]:
```
<3733x7082 sparse matrix of type '<class 'numpy.int64'>'
        with 49992 stored elements in Compressed Sparse Row format>
```

Transform Counts to Frequencies with Tf-idf

In [29]:
```
# Term Frequency (Tf) = Divide number of occurrences of each word in the
# by the total number of words in the document.

# Inverse Document Frequency (idf) = log(n/(df(t) + 1)) where n = number
# df(t) = document frequency of t
```

In [30]:
```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()
```

In [31]:
```
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

In [32]:
```
X_train_tfidf.shape
```

Out[32]:
```
(3733, 7082)
```

In [33]:
```
# We can combine CountVectorizer and TfidfTransformer into one step as g
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
```

In [34]:
```
X_train_tfidf.shape
```

Out[34]:
```
(3733, 7082)
```

Train a classifier

In [36]:
```
from sklearn.svm import LinearSVC
```

In [37]:
```
clf = LinearSVC()
```

In [38]:
```
clf.fit(X_train_tfidf, y_train)
```

Out[38]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

Build a Pipeline

In [40]:
```
# We combine Tfidftransformer and LinearSVC into one step usong Pipeline
from sklearn.pipeline import Pipeline
```

In [43]:
```
text_clf = Pipeline([('tfidf', TfidfVectorizer()), ('clf', LinearSVC())]
```

In [44]:
```
text_clf.fit(X_train, y_train)
```

Out[44]:
```
Pipeline(memory=None,
     steps=[('tfidf', TfidfVectorizer(analyzer='word', binary=False, d
ecode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=T
rue,...ax_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0))])
```

In [45]:
```
predictions = text_clf.predict(X_test)
```

In [46]:
```
from sklearn.metrics import confusion_matrix, classification_report, acc
print(confusion_matrix(y_test, predictions))
```

```
[[1586    7]
 [  12  234]]
```

In [47]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

         ham       0.99      1.00      0.99      1593
        spam       0.97      0.95      0.96       246

   micro avg       0.99      0.99      0.99      1839
   macro avg       0.98      0.97      0.98      1839
weighted avg       0.99      0.99      0.99      1839
```

In [49]:
```
print(accuracy_score(y_test, predictions))
```

```
0.989668297988037
```

In [51]:
```
# Predictions for text messages
text_clf.predict(["Hi, how are you?"])
```

Out[51]:
```
array(['ham'], dtype=object)
```

In [52]:
```
text_clf.predict(["Congratulations, you have been selected the winner fo
```

Out[52]:
```
array(['spam'], dtype=object)
```

In [ ]:
```

```