# Feature Extraction

August 8, 2025

## 1 Feature Extraction from Text

Here's how this process typically works step-by-step

- Read in a collection of documents - a *corpus*
- Transform text into numerical vector data using a pipeline
- Create a classifier
- Fit/train the classifier
- Test the classifier on new data
- Evaluate performance

### 1.0.1 Perform imports and load the dataset

The dataset contains the text of 5572 e-mail messages. Where 4825 are legitimate correspondence messages, 747 are spam, and the text has been preprocessed as a csv file.

[3]:
```
# Load dataset
import pandas as pd
import numpy as np
df = pd.read_csv (r'C:
                                                \review.csv')
df.head(5)
```

[3]:

| | Category | 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… |

### 1.0.2 Check for missing values:

[7]:
```
df.isnull().sum()
```

[7]:
```
Category    0
Message     0
dtype: int64
```

### 1.0.3 Take a quick look at the *ham* and *spam* Category column:

[9]:
```
df['Category'].value_counts()
```

[9]:
```
Category
ham     4825
spam     747
Name: count, dtype: int64
```

4825 out of 5572 messages, or 86.6%, are ham. This means that any text classification model we create has to perform **better than 86.6%** to beat random chance.

### 1.0.4 Split the data into train & test

[12]:
```
from sklearn.model_selection import train_test_split

X = df['Message']  # this time we want to look at the text
y = df['Category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)
```

### 1.0.5 CountVectorizer

Convert a collection of text documents to a matrix of token counts. Text pre-processing, tokenizing and ability to filter out stop words are all included in count vectorizer. Which build a dictionary of features and transforms documents to feature vectors.

[16]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer() # instance of count vectorizer

X_train_counts = count_vect.fit_transform(X_train) # fit and transform training␣
↪data
X_train_counts.shape
```

[16]:
```
(3733, 7081)
```

### 1.0.6 TfidVectorizer

Term Frequency-Inverse Document Frequency vectorizer that transforms text data into numerical format and evaluate how important a word is to a document.

[17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()

X_train_tfidf = vectorizer.fit_transform(X_train)
X_train_tfidf.shape
```

[17]: (3733, 7081)

### 1.0.7 Train a Classifier

SVM classifier that's similar to SVC, called LinearSVC. LinearSVC handles sparse input better, and scales well to large numbers of samples.

[19]:
```
from sklearn.svm import LinearSVC
clf = LinearSVC()
clf.fit(X_train_tfidf,y_train)
```

[19]: LinearSVC()

### 1.0.8 Pipeline

This can perform both vectorization and classification.

[21]:
```
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('tfidf', TfidfVectorizer()),('clf', LinearSVC())])
```

[22]:
```
text_clf.fit(X_train, y_train)  # Feed the training data through the pipeline
```

[22]: Pipeline(steps=[('tfidf', TfidfVectorizer()), ('clf', LinearSVC())])

### 1.0.9 Test the classifier

[24]:
```
predictions = text_clf.predict(X_test) # Form a prediction set
```

[25]:
```
# Report the confusion matrix
from sklearn import metrics
print(metrics.confusion_matrix(y_test,predictions))
```

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

[26]:
```
# Print a classification report
print(metrics.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

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

[27]:
```
# Print the overall accuracy
print(metrics.accuracy_score(y_test,predictions))
```

```
0.989668297988037
```

Our model performed exceedingly well; it correctly predicted spam **98.97%** of the time!

### 1.0.10 Predict

[48]:
```
text_clf.predict(["This Best Mobile Casino Ever : +150 Free Spins!, And $13963.
 ↪99  Welcome Bonus You Will Win the Millionaire's Life ."])
```

[48]:
```
array(['spam'], dtype=object)
```

[55]:
```
text_clf.predict(["Hi, How are you?"])
```

[55]:
```
array(['ham'], dtype=object)
```