Statistics Foundation

Machine Learning – Text Analysis

Digital Vidya

Text Analysis

- **Natural Language Processing**

  NLP is a field in machine learning with the ability of a computer to understand, analyse, manipulate, and potentially generate human language

NLP in Real Life

- Information Retrieval(Google finds relevant and similar results)
- Information Extraction(Gmail structures events from emails)
- Machine Translation(Google Translate translates language from one language to another)
- Text Simplification(Rewordify simplifies the meaning of sentences)
- Sentiment Analysis(Hater News gives us the sentiment of the user)
- Text Summarization(Smmry or Reddit's autotldr gives a summary of sentences)
- Spam Filter(Gmail filters spam emails separately)
- Auto-Predict(Google Search predicts user search results)
- Auto-Correct(Google Keyboard and Grammarly correct words otherwise spelled wrong)
- Speech Recognition(Google WebSpeech or Vocalware)
- Question Answering(IBM Watson's answers to a query)
- Natural Language Generation(Generation of text from image or video data)

Text Analysis

- **(Natural Language Toolkit)NLTK:**
  - NLTK is a popular open-source package in Python. Rather than building all tools from scratch, NLTK provides all common NLP Tasks.

Installing NLTK

- Type !pip install nltk in the Jupyter Notebook or if it doesn't work in cmd type conda install -c conda-forge nltk

```
In [*]:
1 import nltk
2 nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Reading and Exploring Dataset

- **Reading in text data & why do we need to clean the text?**
  - While reading data, we get data in the structured or unstructured format. A structured format has a well-defined pattern whereas unstructured data has no proper structure. In between the 2 structures, we have a semi-structured format which is a comparably better structured than unstructured format.

  - As we can see from above when we read semi-structured data it is hard to interpret so we use pandas to easily understand our data.

Reading and Exploring Dataset

- **Reading in text data & why do we need to clean the text?**
  - While reading data, we get data in the structured or unstructured format. A structured format has a well-defined pattern whereas unstructured data has no proper structure. In between the 2 structures, we have a semi-structured format which is a comparably better structured than unstructured format.

In [3]:
```
import pandas as pd
# Reading Tab separated Value
data = pd.read_csv('SMSSpamCollection.tsv', sep='\t', names=['label', 'body_text'], header=None)
# Print first 5 data
data.head()
```

Out[3]:

	label	body_text
0	ham	I've been searching for the right words to tha...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	Nah I don't think he goes to usf, he lives aro...
3	ham	Even my brother is not like to speak with me. ...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!

Reading and Exploring Dataset

- **Pre-processing Data**
  - Cleaning up the text data is necessary to highlight attributes that we're going to want our machine learning system to pick up on.
  - Cleaning (or pre-processing) the data typically consists of a number of steps:
  - Remove punctuation
  - Tokenization
  - Remove stopwords
  - Stemming
  - Lemmatization
  - Vectorizing Data

Pre-processing Data

- **Remove punctuation**
- Punctuation can provide grammatical context to a sentence which supports our understanding. But for our vectorizer which counts the number of words and not the context, it does not add value, so we remove all special characters.

eg: How are you?->How are you

Remove punctuation

In [7]:
```
import string
string.punctuation
```

Out[7]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

In [8]:
```
#Function to remove Punctuation
def remove_punct(text):
    text_nopunct = "".join([char for char in text if char not in string.punctuation])# It will discard all punctuations
    return text_nopunct

data['body_text_clean'] = data['body_text'].apply(lambda x: remove_punct(x))

data.head()
```

Out[8]:

	label	body_text	body_text_clean
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL

In body_text_clean, we can see that all punctuations like I've-> Ive are omitted.

Digital Vidya

Pre-processing Data

- **Tokenization**
- Tokenizing separates text into units such as sentences or words.
- It gives structure to previously unstructured text. eg: Plata o Plomo-> ‘Plata’,’o’,’Plomo’.

Tokenization

In [9]:
```
import re

# Function to Tokenize words
def tokenize(text):
    tokens = re.split('\W+', text) #W+ means that either a word character (A-Za-z0-9_) or a dash (-) can go there.
    return tokens

data['body_text_tokenized'] = data['body_text_clean'].apply(lambda x: tokenize(x.lower()))
#We convert to lower as Python is case-sensitive.

data.head()
```

Out[9]:

	label	body_text	body_text_clean	body_text_tokenized
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]

In body_text_tokenized, we can see that all words are generated as tokens.

Pre-processing Data

- **Remove stopwords**
- Stopwords are common words that will likely appear in any text. They don't tell us much about our data so we remove them. eg: silver or lead is fine for me-> silver, lead, fine.

Remove stopwords

In [10]:
```
import nltk

stopword = nltk.corpus.stopwords.words('english')# All English Stopwords
```

In [11]:
```
# Function to remove Stopwords
def remove_stopwords(tokenized_list):
    text = [word for word in tokenized_list if word not in stopword]# To remove all stopwords
    return text

data['body_text_nostop'] = data['body_text_tokenized'].apply(lambda x: remove_stopwords(x))

data.head()
```

Out[11]:

	label	body_text	body_text_clean	body_text_tokenized	body_text_nostop
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...	[ive, searching, right, words, thank, breather...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]	[date, sunday]

In body_text_tokenized, we can see that all words are generated as tokens.

Digital Vidya

Pre-processing Data

- **Stemming**
- Stemming helps reduce a word to its stem form. It often makes sense to treat related words in the same way. It removes suffices, like “ing”, “ly”, “s”, etc. by a simple rule-based approach. It reduces the corpus of words but often the actual words get neglected.
- eg: Entitling,Entitled->Entitl

Preprocessing Data: Using Stemming

In [12]:
```
ps = nltk.PorterStemmer()

def stemming(tokenized_text):
    text = [ps.stem(word) for word in tokenized_text]
    return text

data['body_text_stemmed'] = data['body_text_nostop'].apply(lambda x: stemming(x))

data.head()
```

Out[12]:

	label	body_text	body_text_clean	body_text_tokenized	body_text_nostop	body_text_stemmed
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[Ive, been, searching, for, the, right, words,...	[Ive, searching, right, words, thank, breather...	[Ive, search, right, word, thank, breather, pr...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entri, 2, wkli, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...	[nah, dont, think, goe, usf, live, around, tho...
3	ham	Even my brother is not like to speak with me	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...	[even, brother, like, speak, treat, like, aid, ...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[I, have, a, date, on, sunday, with, will]	[date, sunday]	[date, sunday]

In body_text_stemmed, words like entry,wkly is stemmed to entri,wkli even though don’t mean anything

Pre-processing Data

- **Lemmatizing**
- Lemmatizing derives the canonical form ('lemma') of a word. i.e the root form. It is better than stemming as it uses a dictionary-based approach i.e a morphological analysis to the root word.eg: Entitling, Entitled->Entitle
- In Short, Stemming is typically faster as it simply chops off the end of the word, without understanding the context of the word. Lemmatizing is slower and more accurate as it takes an informed analysis with the context of the word in mind'

Preprocessing Data: Using a Lemmatizer

```
In [13]:
wn = nltk.WordNetLemmatizer()

def lemmatizing(tokenized_text):
    text = [wn.lemmatize(word) for word in tokenized_text]
    return text

data['body_text_lemmatized'] = data['body_text_nostop'].apply(lambda x: lemmatizing(x))

data.head(10)
```

Out[13]:

	label	body_text	body_text_clean	body_text_tokenized	body_text_nostop	body_text_stemmed	body_text_lemmatized
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[Ive, been, searching, for, the, right, words,...	[Ive, searching, right, words, thank, breather...	[Ive, search, right, word, thank, breather, pr...	[Ive, searching, right, word, thank, breather,...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entri, 2, wkli, comp, win, fa, cup, fin...	[free, entry, 2, wkly, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...	[nah, dont, think, goe, usf, live, around, tho...	[nah, dont, think, go, usf, life, around, though]
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...	[even, brother, like, speak, treat, like, aid,...	[even, brother, like, speak, treat, like, aid,...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]	[date, sunday]	[date, sunday]	[date, sunday]
5	ham	As per your request 'Melle Melle (Oru Minnamin...	As per your request Melle Melle Oru Minnaminun...	[as, per, your, request, melle, melle, oru, mi...	[per, request, melle, melle, oru, minnaminungi...	[per, request, mell, mell, oru, minnaminungint...	[per, request, melle, melle, oru, minnaminungi...

In body_text_stemmed, we can words like chances are lemmatized to chance whereas it is stemmed to chanc

Digital Vidya

Text Analysis

Vectorizing Data

- Vectorizing is the process of encoding text as integers i.e. numeric form to create feature vectors so that machine learning algorithms can understand our data

Vectorizing Data: Bag-Of-Words

- Bag of Words (BoW) or CountVectorizer describes the presence of words within the text data
- It gives a result of 1 if present in the sentence and 0 if not present
- It, therefore, creates a bag of words with a document-matrix count in each text document
- BOW is applied on the body_text, so the count of each word is stored in the document matrix.

Text Analysis

Vectorizing Data: N-Grams

- N-grams are simply all combinations of adjacent words or letters of length n that we can find in our source text
- Ngrams with n=1 are called unigrams
- Similarly, bigrams (n=2), trigrams (n=3) and so on can also be used

N-Grams

"plata o plomo means silver or lead"

n	Name	Tokens
2	bigram	["plata o", "o plomo", "plomo means", "means silver", "silver or","or lead"]
3	trigram	["plata o plomo", "o plomo means", "plomo means silver", "means silver or ", "silver or lead"]

Text Analysis

Vectorizing Data: N-Grams

- Unigrams usually don't contain much information as compared to bigrams and trigrams
- The basic principle behind n-grams is that they capture the letter or word is likely to follow the given word
- The longer the n-gram (higher n), the more context you have to work with

Apply CountVectorizer (N-Grams)

In [20]:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 ngram_vect = CountVectorizer(ngram_range=(2,2),analyzer=clean_text) # It applies only bigram vectorizer
4 X_counts = ngram_vect.fit_transform(data['body_text'])
5 print(X_counts.shape)
6 print(ngram_vect.get_feature_names())
```

- N-Gram is applied on the body_text, so the count of each group words in a sentence word is stored in the document matrix

Text Analysis

Vectorizing Data: N-Grams

- Unigrams usually don't contain much information as compared to bigrams and trigrams
- The basic principle behind n-grams is that they capture the letter or word is likely to follow the given word
- The longer the n-gram (higher n), the more context you have to work with

Apply CountVectorizer (N-Grams)

In [20]:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 ngram_vect = CountVectorizer(ngram_range=(2,2),analyzer=clean_text) # It applies only bigram vectorizer
4 X_counts = ngram_vect.fit_transform(data['body_text'])
5 print(X_counts.shape)
6 print(ngram_vect.get_feature_names())
```

- N-Gram is applied on the body_text, so the count of each group words in a sentence word is stored in the document matrix

Text Analysis

Vectorizing Data: N-Grams

- Unigrams usually don't contain much information as compared to bigrams and trigrams
- The basic principle behind n-grams is that they capture the letter or word is likely to follow the given word
- The longer the n-gram (higher n), the more context you have to work with

Apply CountVectorizer (N-Grams)

In [20]:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 ngram_vect = CountVectorizer(ngram_range=(2,2),analyzer=clean_text) # It applies only bigram vectorizer
4 X_counts = ngram_vect.fit_transform(data['body_text'])
5 print(X_counts.shape)
6 print(ngram_vect.get_feature_names())
```

- N-Gram is applied on the body_text, so the count of each group words in a sentence word is stored in the document matrix

Text Analysis

Vectorizing Data: TF-IDF

- It computes “relative frequency” that a word appears in a document compared to its frequency across all documents
- It is more useful than “term frequency” for identifying “important” words in each document (high frequency in that document, low frequency in other documents).

Apply TfidfVectorizer

In [22]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(analyzer=clean_text)
X_tfidf = tfidf_vect.fit_transform(data['body_text'])
print(X_tfidf.shape)
print(tfidf_vect.get_feature_names())
```

- TF-IDF is applied on the body_text, so the relative count of each word in the sentences is stored in the document matrix
- Vectorizers outputs sparse matrices. Sparse Matrix is a matrix in which most entries are 0.
- In the interest of efficient storage, a sparse matrix will be stored by only storing the locations of the non-zero elements

Text Analysis

- **Feature Engineering: Feature Creation**
- Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work
- It is like an art as it requires domain knowledge and it can tough to create features, but it can be fruitful for ML algorithm to predict results as they can be related to the prediction

Create feature for text message length and % of punctuation in text

In [27]:

```
import string

# Function to calculate length of message excluding space
data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(" "))

data.head()

def count_punct(text):
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(" ")), 3)*100

data['punct%'] = data['body_text'].apply(lambda x: count_punct(x))

data.head()
```

	label	body_text	body_len	punct%
0	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...	128	4.7
1	ham	Nah I don't think he goes to usf, he lives around here though	49	4.1
2	ham	Even my brother is not like to speak with me. They treat me like aids patent.	62	3.2
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	28	7.1
4	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your call...	135	4.4

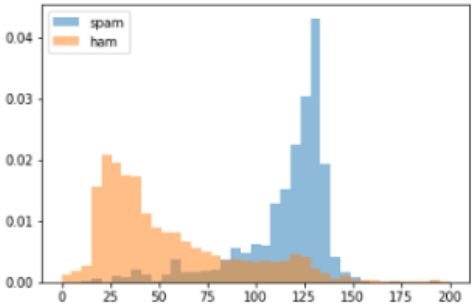
- body_len shows the length of words excluding whitespaces in a message body.
- punct% shows the percentage of punctuation marks in a message body.

Text Analysis

- **Check If Features are good or not**

In [29]:
```
bins = np.linspace(0, 200, 40)

plt.hist(data[data['label']=='spam']['body_len'], bins, alpha=0.5, normed=True, label='spam')
plt.hist(data[data['label']=='ham']['body_len'], bins, alpha=0.5, normed=True, label='ham')
plt.legend(loc='upper left')
plt.show()
```

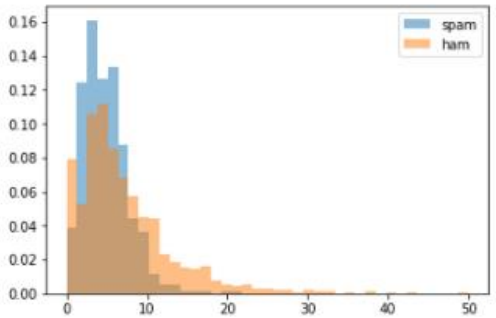


We can clearly see that Spams have a high number of words as compared to Hams. So it's a good feature to distinguish.

Text Analysis

- **Check If Features are good or not**

```
In [30]:
1 bins = np.linspace(0, 50, 40)
2
3 plt.hist(data[data['label']=='spam']['punct%'], bins, alpha=0.5, normed=True, label='spam')
4 plt.hist(data[data['label']=='ham']['punct%'], bins, alpha=0.5, normed=True, label='ham')
5 plt.legend(loc='upper right')
6 plt.show()
```



Spam has a percentage of punctuations but not that far away from Ham. Surprising as at times spam emails can contain a lot of punctuation marks. But still, it can be identified as a good feature.

Text Analysis

- **Building ML Classifiers: Model selection**
- We use an ensemble method of machine learning where multiple models are used and their combination produces better results than a single model(Support Vector Machine/Naive Bayes)
- Ensemble methods are the first choice for many Kaggle Competitions
- Random Forest i.e multiple random decision trees are constructed and the aggregates of each tree are used for the final prediction
- It can be used for classification as well as regression problems. It follows a bagging strategy where randomly

Grid-search:

It exhaustively searches overall parameter combinations in a given grid to determine the best model

Cross-validation:

It divides a data set into k subsets and repeat the method k times where a different subset is used as the test set i.e in each iteration

Text Analysis

- **Building ML Classifiers: Model selection**

For CountVectorizer

In [33]:
```
rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300],
        'max_depth': [30, 60, 90, None]}

gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)# n_jobs=-1 for parallelizing search
gs_fit = gs.fit(X_count_feat, data['label'])
pd.DataFrame(gs_fit.cv_results_).sort_values('mean_test_score', ascending=False).head()
```

mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	...	mean_test_score
0.522701	0.057746	90	150	{'max_depth': 90, 'n_estimators': 150}	0.978475	0.976640	0.973944	...	0.973774
0.621246	0.184220	None	300	{'max_depth': None, 'n_estimators': 300}	0.977578	0.973046	0.973944	...	0.972696
0.739778	0.078957	90	300	{'max_depth': 90, 'n_estimators': 300}	0.976682	0.975741	0.973944	...	0.972517
0.505711	0.076573	None	150	{'max_depth': None, 'n_estimators': 150}	0.977578	0.973046	0.974843	...	0.972337

- The mean_test_score for n_estimators =150 and max_depth gives the best result
- Where n_estimators is the number of trees in the forest.(group of decision trees) and
- max_depth is the max number of levels in each decision tree.

Text Analysis

- **Building ML Classifiers: Model selection**

For TF-IDFVectorizer

In [34]:

```
rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300],
        'max_depth': [30, 60, 90, None]}

gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)# n_jobs=-1 for parallelizing search
gs_fit = gs.fit(X_tfidf_feat, data['label'])
pd.DataFrame(gs_fit.cv_results_).sort_values('mean_test_score', ascending=False).head()
```

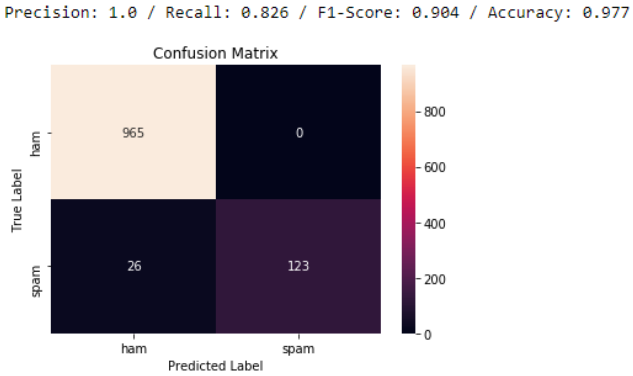
mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	...	mean_test_score
0.863907	0.499025	90	150	{'max_depth': 90, 'n_estimators': 150}	0.978475	0.977538	0.975741	...	0.975031
0.527299	0.081872	None	150	{'max_depth': None, 'n_estimators': 150}	0.978475	0.977538	0.973944	...	0.973594
0.590059	0.170319	None	300	{'max_depth': None, 'n_estimators': 300}	0.978475	0.974843	0.973046	...	0.973594
0.853713	0.067788	90	300	{'max_depth': 90, 'n_estimators': 300}	0.976682	0.975741	0.973046	...	0.973235

the mean_test_score for n_estimators =150 and max_depth=90 gives the best result

Text Analysis

Spam-Ham Classifier

- All the above-discussed sections are combined to build a Spam-Ham Classifier
- Random Forest gives an accuracy of 97.7%
- High-value F1-score is also obtained from the model
- Confusion Matrix tells us that we correctly predicted 965 hams and 123 spams
- 0 hams were incorrectly identified as spams and 26 spams were incorrectly predicted as hams
- Detecting spams as hams are justifiable as compared to hams as spams



Digital Vidya

SPACY

Advanced Text Mining – using spaCy

spaCy

- spaCy is a free, open-source library for NLP in Python
- It’s written in Cython and is designed to build information extraction or natural language understanding systems.
- It’s built for production use and provides a concise and user-friendly API

spaCy Models

- spaCy v2.0 features new neural models for tagging, parsing and entity recognition
- The models have been designed and implemented from scratch specifically for spaCy, to give you an unmatched balance of speed, size and accuracy
- A novel bloom embedding strategy with subword features is used to support huge vocabularies in tiny tables
- The default model for the English language is en_core_web_sm.

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
```

spaCy Models

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
```

- Here, the nlp object is a language model instance.
- nlp refers to the language model loaded by `en_core_web_sm`.
- Now spaCy can be used to read a string or a text file

Reading String

- spaCy to create a processed Doc object, which is a container for accessing linguistic annotations, for a given input string

```
>>> introduction_text = ('This tutorial is about Natural'
...     ' Language Processing in Spacy.')
>>> introduction_doc = nlp(introduction_text)
>>> # Extract tokens for the given doc
>>> print ([token.text for token in introduction_doc])
['This', 'tutorial', 'is', 'about', 'Natural', 'Language',
'Processing', 'in', 'Spacy', '.']
```

Reading File

- spaCy to create a processed Doc object, which is a container for accessing linguistic annotations, for a given input string

```
>>> file_name = 'introduction.txt'
>>> introduction_file_text = open(file_name).read()
>>> introduction_file_doc = nlp(introduction_file_text)
>>> # Extract tokens for the given doc
>>> print ([token.text for token in introduction_file_doc])
['This', 'tutorial', 'is', 'about', 'Natural', 'Language',
'Processing', 'in', 'Spacy', '.', '\n']
```

Sentence Detection

- Sentence Detection is the process of locating the start and end of sentences in a given text
- This allows you to you divide a text into linguistically meaningful units
- We can use these units when you're processing your text to perform tasks such as part of speech tagging and entity extraction

```
>>> about_text = ('Gus Proto is a Python developer currently'
...               ' working for a London-based Fintech'
...               ' company. He is interested in learning'
...               ' Natural Language Processing.')
>>> about_doc = nlp(about_text)
>>> sentences = list(about_doc.sents)
>>> len(sentences)
2
>>> for sentence in sentences:
...     print (sentence)
...
'Gus Proto is a Python developer currently working for a
London-based Fintech company.'
'He is interested in learning Natural Language Processing.'
```

Tokenization

- Tokenization is the next step after sentence detection.
- It allows you to identify the basic units in your text. These basic units are called tokens.
- Tokenization is useful because it breaks a text into meaningful units. These units are used for further analysis, like part of speech tagging.
- In spaCy, you can print tokens by iterating on the Doc object:

```
>>> for token in about_doc:
...     print (token, token.idx)
...
Gus 0
Proto 4
is 10
a 13
Python 15
```

Tokenization

- spaCy preserves the starting index of the tokens
- It's useful for in-place word replacement
- spaCy provides various attributes for the Token class:

```
>>> for token in about_doc:
...     print (token, token.idx, token.text_with_ws,
...            token.is_alpha, token.is_punct, token.is_space,
...            token.shape_, token.is_stop)
...
Gus 0 Gus  True False False Xxx False
Proto 4 Proto  True False False Xxxxx False
is 10 is  True False False xx True
a 13 a  True False False x True
Python 15 Python  True False False Xxxxx False
developer 22 developer  True False False xxxx False
```

Tokenization

- Some of the commonly required attributes are accessed:
  - `text_with_ws` prints token text with trailing space (if present).
  - `is_alpha` detects if the token consists of alphabetic characters or not.
  - `is_punct` detects if the token is a punctuation symbol or not.
  - `is_space` detects if the token is a space or not.
  - `shape_` prints out the shape of the word.
  - `is_stop` detects if the token is a stop word or not.

Stopwords

- Stop words are the most common words in a language. In the English language, some examples of stop words are the, are, but, and they. Most sentences need to contain stop words in order to be full sentences that make sense.

- Generally, stop words are removed because they aren't significant and distort the word frequency analysis.

```
>>> import spacy
>>> spacy_stopwords = spacy.lang.en.stop_words.STOP_WORDS
>>> len(spacy_stopwords)
326
>>> for stop_word in list(spacy_stopwords)[:10]:
...     print(stop_word)
...
using
becomes
had
itself
once
often
is
herein
who
too
```

Stopwords

- Remove stop words from the input text:

```
>>> for token in about_doc:
...     if not token.is_stop:
...         print (token)
...
Gus
Proto
Python
developer
currently
working
London
-
```

```
>>> about_no_stopword_doc = [token for token in about_doc if not token.is_stop]
>>> print (about_no_stopword_doc)
[Gus, Proto, Python, developer, currently, working, London,
-, based, Fintech, company, ., interested, learning, Natural,
Language, Processing, .]
```

Lemmatization

- Lemmatization is the process of reducing inflected forms of a word while still ensuring that the reduced form belongs to the language
- This reduced form or root word is called a lemma
- For example, organizes, organized and organizing are all forms of organize.
- Here, organize is the lemma. The inflection of a word allows you to express different grammatical categories like tense (organized vs organize), number (trains vs train), and so on.
- Lemmatization is necessary because it helps reduce the inflected forms of a word so that they can be analyzed as a single item.
- It also help normalize the text

Lemmatization

- spaCy has the attribute lemma on the Token class
- This attribute has the lemmatized form of a token:

```
>>> conference_help_text = ('Gus is helping organize a developer'
...     'conference on Applications of Natural Language'
...     ' Processing. He keeps organizing local Python meetups'
...     ' and several internal talks at his workplace.')
>>> conference_help_doc = nlp(conference_help_text)
>>> for token in conference_help_doc:
...     print (token, token.lemma_)
...
Gus Gus
is be
helping help
organize organize
a a
developer developer
conference conference
on on
Applications Applications
```

Lemmatization

- In this example, organizing reduces to its lemma form organize.
- If you do not lemmatize the text, then organize and organizing will be counted as different tokens, even though they both have a similar meaning.
- Lemmatization helps you avoid duplicate words that have similar meanings.

Word Frequency

- You can now convert a given text into tokens and perform statistical analysis over it.
- This analysis can give you various insights about word patterns, such as common words or unique words in the text:

```
>>> from collections import Counter
>>> complete_text = ('Gus Proto is a Python developer currently'
...     'working for a London-based Fintech company. He is'
...     ' interested in learning Natural Language Processing.'
...     ' There is a developer conference happening on 21 July'
...     ' 2019 in London. It is titled "Applications of Natural'
...     ' Language Processing". There is a helpline number '
...     ' available at +1-1234567891. Gus is helping organize it.'
...     ' He keeps organizing local Python meetups and several'
...     ' internal talks at his workplace. Gus is also presenting'
...     ' a talk. The talk will introduce the reader about "Use'
...     ' cases of Natural Language Processing in Fintech".'
...     ' Apart from his work, he is very passionate about music.'
...     ' Gus is learning to play the Piano. He has enrolled '
...     ' himself in the weekend batch of Great Piano Academy.'
...     ' Great Piano Academy is situated in Mayfair or the City'
...     ' of London and has world-class piano instructors.')
```

Word Frequency

- You can now convert a given text into tokens and perform statistical analysis over it.
- This analysis can give you various insights about word patterns, such as common words or unique words in the text:

```
>>> complete_doc = nlp(complete_text)
>>> # Remove stop words and punctuation symbols
>>> words = [token.text for token in complete_doc
...          if not token.is_stop and not token.is_punct]
>>> word_freq = Counter(words)
>>> # 5 commonly occurring words with their frequencies
>>> common_words = word_freq.most_common(5)
>>> print (common_words)
[('Gus', 4), ('London', 3), ('Natural', 3), ('Language', 3), ('Processing', 3)]
>>> # Unique words
>>> unique_words = [word for (word, freq) in word_freq.items() if freq == 1]
>>> print (unique_words)
['Proto', 'currently', 'working', 'based', 'company',
'interested', 'conference', 'happening', '21', 'July',
'2019', 'titled', 'Applications', 'helpline', 'number',
```

Part of Speech Tagging

Part of speech or POS is a grammatical role that explains how a particular word is used in a sentence.

There are eight parts of speech:

- Noun
- Pronoun
- Adjective
- Verb
- Adverb
- Preposition
- Conjunction
- Interjection
- Part of speech tagging is the process of assigning a POS tag to each token depending on its usage in the sentence. POS tags are useful for assigning a syntactic category like noun or verb to each word.

Part of Speech Tagging

- In spaCy, POS tags are available as an attribute on the Token object:

```
>>> for token in about_doc:
...     print (token, token.tag_, token.pos_, spacy.explain(token.tag_))
...
Gus NNP PROPN noun, proper singular
Proto NNP PROPN noun, proper singular
is VBZ VERB verb, 3rd person singular present
a DT DET determiner
Python NNP PROPN noun, proper singular
developer NN NOUN noun, singular or mass
currently RB ADV adverb
working VBG VERB verb, gerund or present participle
for IN ADP conjunction, subordinating or preposition
a DT DET determiner
London NNP PROPN noun, proper singular
- HYPH PUNCT punctuation mark, hyphen
based VBN VERB verb, past participle
Fintech NNP PROPN noun, proper singular
company NN NOUN noun, singular or mass
. . PUNCT punctuation mark, sentence closer
He PRP PRON pronoun, personal
is VBZ VERB verb, 3rd person singular present
interested JJ ADJ adjective
```

Part of Speech Tagging

Here, two attributes of the Token class are accessed:

- `tag_` lists the fine-grained part of speech
- `pos_` lists the coarse-grained part of speech
- spacy.explain gives descriptive details about a particular POS tag

Part of Speech Tagging

Using POS tags, you can extract a particular category of words:

```
>>> nouns = []
>>> adjectives = []
>>> for token in about_doc:
...     if token.pos_ == 'NOUN':
...         nouns.append(token)
...     if token.pos_ == 'ADJ':
...         adjectives.append(token)
...
>>> nouns
[developer, company]
>>> adjectives
[interested]
```

You can use this to derive insights, remove the most common nouns, or see which adjectives are used for a particular noun

Preprocessing Functions

- You can create a preprocessing function that takes text as input and applies the following operations:
  - Lowercases the text
  - Lemmatizes each token
  - Removes punctuation symbols
  - Removes stop words
- A preprocessing function converts text to an analyzable format. It's necessary for most NLP tasks.

Preprocessing Functions

- Preprocessing Function

```
>>> def is_token_allowed(token):
...     '''
...         Only allow valid tokens which are not stop words
...         and punctuation symbols.
...     '''
...     if (not token or not token.string.strip() or
...         token.is_stop or token.is_punct):
...         return False
...     return True
...
>>> def preprocess_token(token):
...     # Reduce token to its lowercase lemma form
...     return token.lemma_.strip().lower()
...
>>> complete_filtered_tokens = [preprocess_token(token)
...     for token in complete_doc if is_token_allowed(token)]
>>> complete_filtered_tokens
['gus', 'proto', 'python', 'developer', 'currently', 'work',
'london', 'base', 'fintech', 'company', 'interested', 'learn',
'natural', 'language', 'processing', 'developer', 'conference',
'happen', '21', 'july', '2019', 'london', 'title',
```

Rule-Based Matching

- Rule-based matching is one of the steps in extracting information from unstructured text
- It's used to identify and extract tokens and phrases according to patterns (such as lowercase) and grammatical features (such as part of speech)
- Rule-based matching can use regular expressions to extract entities (such as phone numbers) from an unstructured text
- It's different from extracting text using regular expressions only in the sense that regular expressions don't consider the lexical and grammatical attributes of the text

Rule-Based Matching

- With rule-based matching, you can extract a first name and a last name, which are always proper nouns:

```
>>> from spacy.matcher import Matcher
>>> matcher = Matcher(nlp.vocab)
>>> def extract_full_name(nlp_doc):
...     pattern = [{'POS': 'PROPN'}, {'POS': 'PROPN'}]
...     matcher.add('FULL_NAME', None, pattern)
...     matches = matcher(nlp_doc)
...     for match_id, start, end in matches:
...         span = nlp_doc[start:end]
...         return span.text
...
>>> extract_full_name(about_doc)
'Gus Proto'
```

pattern is a list of objects that defines the combination of tokens to be matched.

Both POS tags in it are PROPN (proper noun).

So, the pattern consists of two objects in which the POS tags for both tokens should be PROPN

Named Entity Recognition

- Named Entity Recognition (NER) is the process of locating named entities in unstructured text and then classifying them into pre-defined categories, such as person names, organizations, locations, monetary values, percentages, time expressions, and so on
- We can use NER to know more about the meaning of text
- For example, we could use it to populate tags for a set of documents in order to improve the keyword search
- We can also use it to categorize customer support tickets into relevant categories.

Named Entity Recognition

- spaCy has the property ents on Doc objects. We can use it to extract named entities:

```
>>> piano_class_text = ('Great Piano Academy is situated'
...     ' in Mayfair or the City of London and has'
...     ' world-class piano instructors.')
>>> piano_class_doc = nlp(piano_class_text)
>>> for ent in piano_class_doc.ents:
...     print(ent.text, ent.start_char, ent.end_char,
...           ent.label_, spacy.explain(ent.label_))
...
Great Piano Academy 0 19 ORG Companies, agencies, institutions, etc.
Mayfair 35 42 GPE Countries, cities, states
the City of London 46 64 GPE Countries, cities, states
```

Named Entity Recognition

- spaCy has the property ents on Doc objects. We can use it to extract named entities:

  - ent is a Span object with various attributes:
  - text gives the Unicode text representation of the entity
  - start_char denotes the character offset for the start of the entity
  - end_char denotes the character offset for the end of the entity.
  - label_ gives the label of the entity.

```
>>> piano_class_text = ('Great Piano Academy is situated'
...     ' in Mayfair or the City of London and has'
...     ' world-class piano instructors.')
>>> piano_class_doc = nlp(piano_class_text)
>>> for ent in piano_class_doc.ents:
...     print(ent.text, ent.start_char, ent.end_char,
...           ent.label_, spacy.explain(ent.label_))
...
Great Piano Academy 0 19 ORG Companies, agencies, institutions, etc.
Mayfair 35 42 GPE Countries, cities, states
the City of London 46 64 GPE Countries, cities, states
```

Named Entity Recognition

- We can use NER to redact people's names from a text. For example, we want to do this in order to hide personal information collected in a survey.

```
>>> survey_text = ('Out of 5 people surveyed, James Robert,'
...                ' Julie Fuller and Benjamin Brooks like'
...                ' apples. Kelly Cox and Matthew Evans'
...                ' like oranges.')
...
>>> def replace_person_names(token):
...     if token.ent_iob != 0 and token.ent_type_ == 'PERSON':
...         return '[REDACTED] '
...     return token.string
...
>>> def redact_names(nlp_doc):
...     for ent in nlp_doc.ents:
...         ent.merge()
...     tokens = map(replace_person_names, nlp_doc)
...     return ''.join(tokens)
...
>>> survey_doc = nlp(survey_text)
>>> redact_names(survey_doc)
'Out of 5 people surveyed, [REDACTED] , [REDACTED] and'
' [REDACTED] like apples. [REDACTED] and [REDACTED]'
' like oranges.'
```

Thank You

Digital Vidya