

Natural Language Processing(NLP) for Machine Learning



Badreesh Shetty [Follow](#)

Nov 24, 2018 · 8 min read

In this article we'll be learning about Natural Language Processing(NLP) which can help computers analyze text easily i.e detect spam emails, autocorrect. We'll see how NLP tasks are carried out for understanding human language.

Natural Language Processing

NLP is a field in machine learning with the ability of a computer to understand, analyze, 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). Shashi Tharoor tweets could be used(pun intended).
- 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.)

(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`. This should work in most cases.

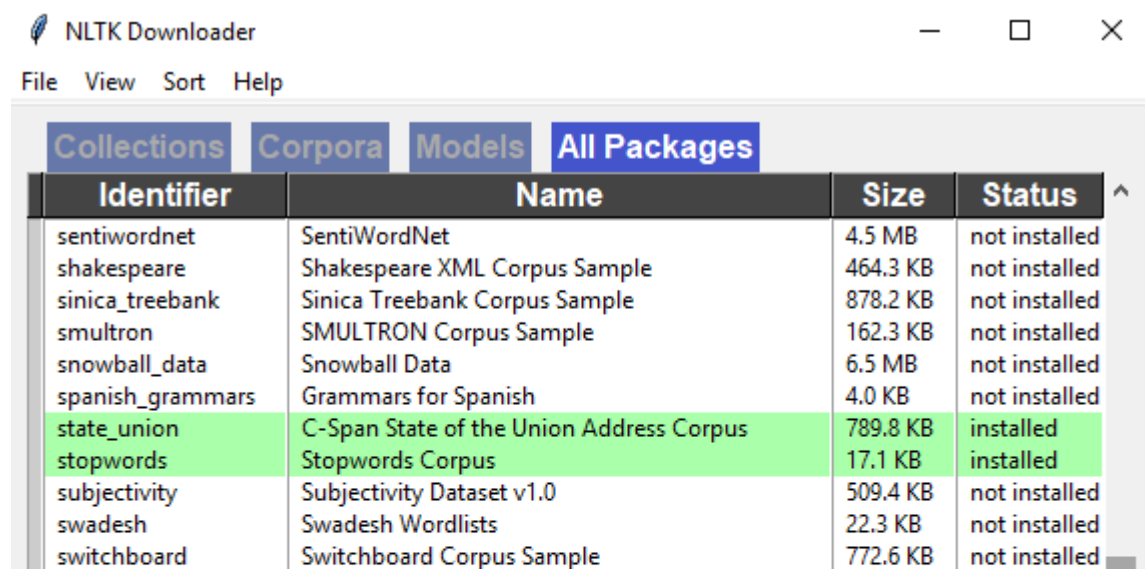
Install NLTK: <http://pypi.python.org/pypi/nltk>

Importing NLTK Library

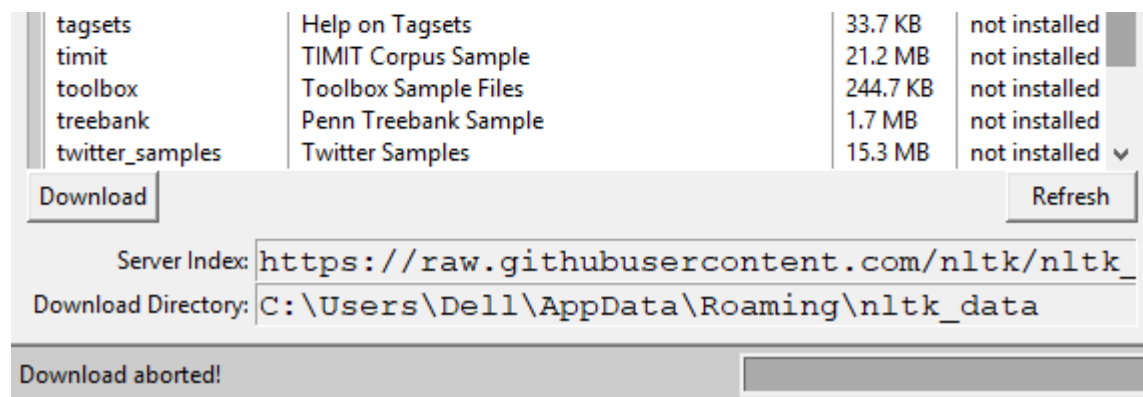
```
In [*]: 1 import nltk
        2 nltk.download()

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

After typing the above, we get an NLTK Downloader Application which is helpful in NLP Tasks



Identifier	Name	Size	Status
sentiwordnet	SentiWordNet	4.5 MB	not installed
shakespeare	Shakespeare XML Corpus Sample	464.3 KB	not installed
sinica_treebank	Sinica Treebank Corpus Sample	878.2 KB	not installed
smultron	SMULTRON Corpus Sample	162.3 KB	not installed
snowball_data	Snowball Data	6.5 MB	not installed
spanish_grammars	Grammars for Spanish	4.0 KB	not installed
state_union	C-Span State of the Union Address Corpus	789.8 KB	installed
stopwords	Stopwords Corpus	17.1 KB	installed
subjectivity	Subjectivity Dataset v1.0	509.4 KB	not installed
swadesh	Swadesh Wordlists	22.3 KB	not installed
switchboard	Switchboard Corpus Sample	772.6 KB	not installed



Stopwords Corpus is already installed in my system which helps in removing redundant repeated words. Similarly, we can install other useful packages.

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.

Reading and Exploring Dataset

```
In [2]: 1 # Read in the raw text
        2 rawData = open("SMSSpamCollection.tsv").read()
        3 # Print the raw data
        4 rawData[0:250]
```

```
Out[2]: "ham\tI've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and w
ill fulfil my promise. You have been wonderful and a blessing at all times.\nspam\tFree entry in 2 a wkly comp to win FA Cup f
i"
```

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.

```
In [3]: 1 import pandas as pd
        2 # Reading Tab separated Value
        3 data = pd.read_csv('SMSSpamCollection.tsv', sep='\t', names=['label', 'body_text'], header=None)
        4 # Print first 5 data
        5 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. ...

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:

1. 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]: 1 import string
        2 string.punctuation

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

In [8]: 1 #Function to remove Punctuation
        2 def remove_punct(text):
        3     text_nopunct = "".join([char for char in text if char not in string.punctuation])# It will discard all punctuations
        4     return text_nopunct
        5
        6 data['body_text_clean'] = data['body_text'].apply(lambda x: remove_punct(x))
        7
        8 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-> I've are omitted.

2.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]: 1 import re
        2
        3 # Function to Tokenize words
        4 def tokenize(text):
```

```

5 tokens = re.split('\W+', text) #\W+ means that either a word character (A-Za-z0-9_) or a dash (-) can go there.
6 return tokens
7
8 data['body_text_tokenized'] = data['body_text_clean'].apply(lambda x: tokenize(x.lower()))
9 #We convert to lower as Python is case-sensitive.
10
11 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...	I've 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.

3. 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]: 1 import nltk
2
3 stopword = nltk.corpus.stopwords.words('english')# All English Stopwords

In [11]: 1 # Function to remove Stopwords
2 def remove_stopwords(tokenized_list):
3     text = [word for word in tokenized_list if word not in stopword]# To remove all stopwords
4     return text
5
6 data['body_text_nostop'] = data['body_text_tokenized'].apply(lambda x: remove_stopwords(x))
7
8 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...	I've 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, 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, ...]	[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_nostop, all unnecessary words like been, for, the are removed.

Preprocessing 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

Note: Some search engines treat words with the same stem as synonyms.

Preprocessing Data: Using Stemming

```
In [12]: 1 ps = nltk.PorterStemmer()
2
3 def stemming(tokenized_text):
4     text = [ps.stem(word) for word in tokenized_text]
5     return text
6
7 data['body_text_stemmed'] = data['body_text_nostop'].apply(lambda x: stemming(x))
8
9 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.

Preprocessing 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]: 1 wn = nltk.WordNetLemmatizer()
2
3 def lemmatizing(tokenized_text):
4     text = [wn.lemmatize(word) for word in tokenized_text]
5     return text
6
7 data['body_text_lemmatized'] = data['body_text_nostop'].apply(lambda x: lemmatizing(x))
8
9 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.

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.

Apply CountVectorizer

```
In [17]: 1 from sklearn.feature_extraction.text import CountVectorizer
2
3 count_vect = CountVectorizer(analyzer=clean_text)
4 X_counts = count_vect.fit_transform(data['body_text'])
5 print(X_counts.shape)
6 print(count_vect.get_feature_names())
```

BOW is applied on the `body_text`, so the count of each word is stored in the document matrix. (Check the repo).

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"]

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. (Check the repo).

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).

Note: Used for search engine scoring, text summarization, document clustering.

Check my previous post — In the TF-IDF Section, I have elaborated on the working of TF-IDF.

Apply TfidfVectorizer

```
In [22]: 1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 tfidf_vect = TfidfVectorizer(analyzer=clean_text)
4 X_tfidf = tfidf_vect.fit_transform(data['body_text'])
5 print(X_tfidf.shape)
6 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. (Check the repo).

Note: 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.

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]: 1 import string
2
3 # Function to calculate length of message excluding space
4 data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(" "))
5
6 data.head()
7
8 def count_punct(text):
9     count = sum([1 for char in text if char in string.punctuation])
10    return round(count/(len(text) - text.count(" ")), 3)*100
11
12 data['punct%'] = data['body_text'].apply(lambda x: count_punct(x))
13
14 data.head()
```

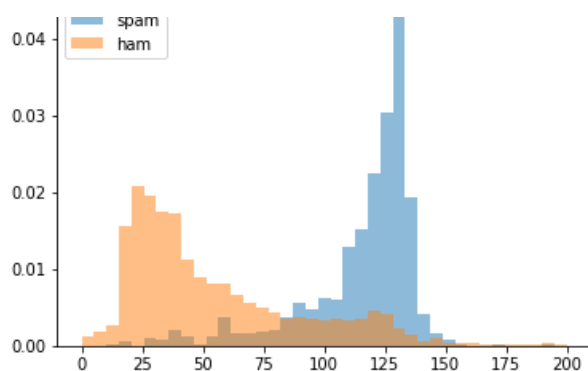
```
Out[27]:
```

	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.

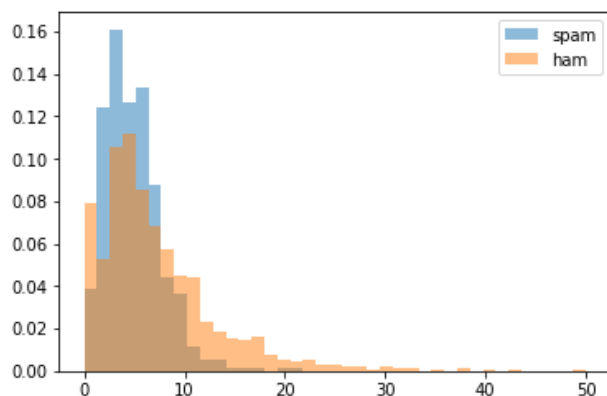
Check If Features are good or not

```
In [29]: 1 bins = np.linspace(0, 200, 40)
2
3 plt.hist(data[data['label']=='spam']['body_len'], bins, alpha=0.5, normed=True, label='spam')
4 plt.hist(data[data['label']=='ham']['body_len'], bins, alpha=0.5, normed=True, label='ham')
5 plt.legend(loc='upper left')
6 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.

```
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.

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.

For CountVectorizer

```
In [33]: 1 rf = RandomForestClassifier()
2 param = {'n_estimators': [10, 150, 300],
3         'max_depth': [30, 60, 90, None]}
4
5 gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)# n_jobs=-1 for parallelizing search
6 gs_fit = gs.fit(X_count_feat, data['label'])
7 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.

For TF-IDFVectorizer

```
In [34]: 1 rf = RandomForestClassifier()
2 param = {'n_estimators': [10, 150, 300],
3         'max_depth': [30, 60, 90, None]}
4
5 gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)# n_jobs=-1 for parallelizing search
6 gs_fit = gs.fit(X_tfidf_feat, data['label'])
7 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

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

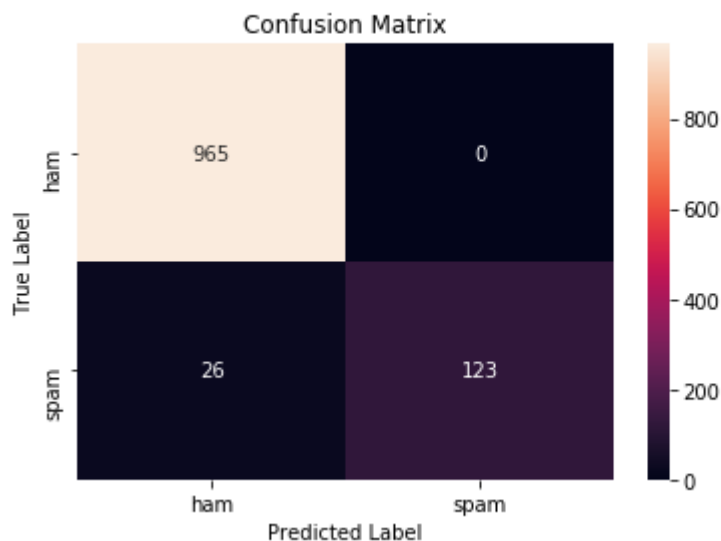
Check this out: Want to Win Competitions? Pay Attention to Your Ensembles.

Improvements: We can use GradientBoosting, XgBoost for classifying. I tried GridSearchCV on GradientBoosting, results were taking a lot of time so scrapped the idea of including here. It takes a lot of time as GradientBoosting takes an iterative approach of combining weak learners to create strong learner by focusing on mistakes of prior iteration. In short, compared to Random Forest it follows a sequential approach rather than random parallel approach.

Spam-Ham Classifier

All the above-discussed sections are combined to build a Spam-Ham Classifier.

Precision: 1.0 / Recall: 0.826 / F1-Score: 0.904 / Accuracy: 0.977



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.

Find the above code in this Github Repo.

Conclusion

In summary, we learned how to perform basic NLP tasks and used a machine learning classifier to predict whether the SMS is Spam or Ham.

[Machine Learning](#) [Data Science](#) [NLP](#) [Text Mining](#) [Data](#)

[About](#) [Help](#) [Legal](#)