Machine Learning: Natural Language Processing

Speaker Qualifications

- Specialize in next-generation technologies
- Author of O'Reilly Videos on Hypermedia, Linking Data, Security and Encryption
- Author of 'Resource-Oriented Architecture Patterns for Webs of Data'
- Teaches and speaks internationally about REST, Semantic Web, Data Science, Machine Learning, GPU Computing, Security, Visualization, Architecture
- Worked in Defense, Finance, Retail, Hospitality, Video Game, Health Care, Telecommunications and Publishing Industries
- International Pop Recording Artist

Agenda

- Introduction
- Vector Space Model
- Word2Vec
- Naieve Bayes

Introduction

Natural Language Processing (NLP) Goals

- Search and Retrieval

Natural Language Processing (NLP) Goals

- Search and Retrieval
- Entity and Relationship Extraction

Natural Language Processing (NLP) Goals

- Search and Retrieval
- Entity and Relationship Extraction
- Linguistic structure

Natural Language Processing (NLP) Goals

- Search and Retrieval
- Entity and Relationship Extraction
- Linguistic structure
- Machine Translation

Natural Language Processing (NLP) Goals

- Search and Retrieval
- Entity and Relationship Extraction
- Linguistic structure
- Machine Translation
- Generative Content

Natural Language Processing (NLP) Goals

- Search and Retrieval
- Entity and Relationship Extraction
- Linguistic structure
- Machine Translation
- Generative Content
- Question Answering

NLP History

- Early successes in the 1950s in automatic translation

NLP History

- Early successes in the 1950s in automatic translation
- SHRDLU

NLP History

- Early successes in the 1950s in automatic translation
- SHRDLU
- ELIZA

NLP History

- Early successes in the 1950s in automatic translation
- SHRDLU
- ELIZA
- Shift to statistical models in the 1980s

NLP Difficulties

NLP Difficulties

- Character encoding

NLP Difficulties

- Character encoding
- Tokenization

NLP Difficulties

- Character encoding
- Tokenization
- Part of Speech labeling

NLP Difficulties

- Character encoding
- Tokenization
- Part of Speech labeling
- Stemming (e.g. "walking", "walked", "walks")

NLP Difficulties

- Character encoding
- Tokenization
- Part of Speech labeling
- Stemming (e.g. "walking", "walked", "walks")
- Lemmatization (e.g. "operating")

NLP Difficulties

- Character encoding
- Tokenization
- Part of Speech labeling
- Stemming (e.g. "walking", "walked", "walks")
- Lemmatization (e.g. "operating")
- Sentence/paragraph detection

NLP Difficulties

- Character encoding
- Tokenization
- Part of Speech labeling
- Stemming (e.g. "walking", "walked", "walks")
- Lemmatization (e.g. "operating")
- Sentence/paragraph detection
- Coreference resolution

Some Open Source NLP Frameworks

API	URL
GATE	http://gate.ac.uk
LingPipe	http://alias-i.com/lingpipe
Apache OpenNLP	http://opennlp.apache.org
UIMA	http://uima.apache.org
Stanford Parser	http://nlp.stanford.edu/software
Mallet	http://mallet.cs.umass.edu

Vector Space Model

Bag of Words

```
(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.
```

https://en.wikipedia.org/wiki/Bag-of-words_model

Bag of Words

```
(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.
```

```
[
    "John",
    "likes",
    "to",
    "watch",
    "movies",
    "also",
    "football",
    "games",
    "Mary",
    "too"
]
```

https://en.wikipedia.org/wiki/Bag-of-words_model

Bag of Words

```
(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.
```

```
[
    "John",
    "likes",
    "to",
    "watch",
    "movies",
    "also",
    "football",
    "games",
    "Mary",
    "too"
]
```

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

https://en.wikipedia.org/wiki/Bag-of-words_model

N-gram model

```
(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.
```

https://en.wikipedia.org/wiki/Bag-of-words_model

N-gram model

```
(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.
```

```
[
    "John likes",
    "likes to",
    "to watch",
    "watch movies",
    "Mary likes",
    "likes movies",
    "movies too",
]
```

https://en.wikipedia.org/wiki/Bag-of-words_model

Vector Space Model

$$d_j = (w_{1,j}, w_{2,j}, \ldots w_{t,j})$$

Vector Space Model

$$d_j = (w_{1,j}, w_{2,j}, \ldots w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, \ldots w_{t,q})$$

Euclidean Dot Product

$$a \cdot b = \|a\| \|b\| cos\theta$$

Euclidean Dot Product

$$a \cdot b = \|a\| \|b\| cos\theta$$

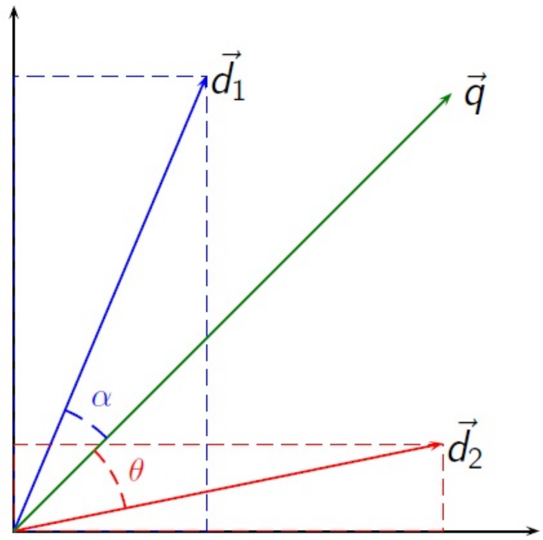
$$cos\theta = \frac{A \cdot B}{\|A\| \|B\|}$$

Euclidean Dot Product

$$a \cdot b = \|a\| \|b\| cos\theta$$

$$cos\theta = \frac{A \cdot B}{\|A\| \|B\|}$$

$$= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$



Cosine Similarity

$$cos\theta = \frac{d_2 \cdot q}{\|d_2\| \|q\|}$$

Cosine Similarity

$$cos\theta = \frac{d_2 \cdot q}{\|d_2\| \|q\|}$$

$$cos\theta = 1 \implies Identical$$

Cosine Similarity

$$cos\theta = \frac{d_2 \cdot q}{\|d_2\| \|q\|}$$

$$cos\theta = 1 \implies Identical$$

$$cos\theta = 0 \implies Orthogonal$$

Term Frequency

$$tf(t,d) = 1$$

https://en.wikipedia.org/wiki/Tf–idf

Term Frequency

$$tf(t,d) = 1$$

$$tf(t,d) = f_{t,d}$$

https://en.wikipedia.org/wiki/Tf–idf

Term Frequency

$$tf(t,d) = 1$$

$$tf(t,d) = f_{t,d}$$

$$tf(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}$$

https://en.wikipedia.org/wiki/Tf–idf

Inverse Document Frequency

$$idf(t,D) = log \frac{N}{|\{d \in D : t \in d\}|}$$

https://en.wikipedia.org/wiki/Tf–idf

Term Frequency-Inverse Document Frequency

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

https://en.wikipedia.org/wiki/Tf–idf

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf(\text{" } this \text{ "}, d_1) = \frac{1}{5} = 0.2$$

$$tf(\text{" } this \text{ "}, d_2) = \frac{1}{7} \approx 0.14$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf(\text{" } this \text{ "}, d_1) = \frac{1}{5} = 0.2$$

$$tf(\text{" } this \text{ "}, d_2) = \frac{1}{7} \approx 0.14$$

Inverse Document Frequency

$$idf(\text{" } this \text{ "}, D) = log\left(\frac{2}{2}\right) = 0$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf(\text{" } this \text{ "}, d_1) = \frac{1}{5} = 0.2$$

$$tf(\text{" } this \text{ "}, d_2) = \frac{1}{7} \approx 0.14$$

Inverse Document Frequency

$$idf(\text{" } this \text{ "}, D) = log\left(\frac{2}{2}\right) = 0$$

Term Frequency-Inverse Document Frequency

$$tfidf(\text{" } this \text{ "}, d_1) = 0.2 \times 0 = 0$$

$$tfidf(\text{" } this \text{ "}, d_2) = 0.4 \times 0 = 0$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf(\text{" } example \text{ "}, d_1) = \frac{0}{5} = 0$$

$$tf(\text{" } example \text{ "}, d_2) = \frac{3}{7} \approx 0.429$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf(\text{" } example \text{ "}, d_1) = \frac{0}{5} = 0$$

$$tf(\text{" } example \text{ "}, d_2) = \frac{3}{7} \approx 0.429$$

Inverse Document Frequency

$$idf(\text{" } example \text{ "}, D) = log\left(\frac{2}{1}\right) = 0.301$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Term Frequency

$$tf("\ example\ ", d_1) = \frac{0}{5} = 0$$

$$tf("\ example\ ", d_2) = \frac{3}{7} \approx 0.429$$

Inverse Document Frequency

$$idf("\ example\ ", D) = log\left(\frac{2}{1}\right) = 0.301$$

Term Frequency-Inverse Document Frequency

$$tfidf("\ example\ ", d_1) = 0 \times 0.301 = 0$$

$$tfidf("\ example\ ", d_2) = 0.429 \times 0.301 = 0.13$$

Lonely Words

- Many NLP systems treated words as isolated indices

Lonely Words

- Many NLP systems treated words as isolated indices
- Words are connected

Lonely Words

- Many NLP systems treated words as isolated indices
- Words are connected
- Similar words are used similarly

Types of Similarity

- Topical similarity based upon textual regions

Types of Similarity

- Topical similarity based upon textual regions
- Paradigmatic similarity based upon co-occurrence

Types of Similarity

- Topical similarity based upon textual regions
- Paradigmatic similarity based upon co-occurrence
- Syntagmatic similarity by examining the vectors

t-SNE Visualizations of Word Embeddings

http://tinyurl.com/hpf24ox

Advantages of VSM

- Simple models using linear algebra

https://en.wikipedia.org/wiki/Vector_space_model

Advantages of VSM

- Simple models using linear algebra
- Non-binary weights

https://en.wikipedia.org/wiki/Vector_space_model

Advantages of VSM

- Simple models using linear algebra
- Non-binary weights
- Continuous ranges of similarity between documents and queries

https://en.wikipedia.org/wiki/Vector_space_model

Advantages of VSM

- Simple models using linear algebra
- Non-binary weights
- Continuous ranges of similarity between documents and queries
- Ranking possibilities based on relevance

https://en.wikipedia.org/wiki/Vector_space_model

Advantages of VSM

- Simple models using linear algebra
- Non-binary weights
- Continuous ranges of similarity between documents and queries
- Ranking possibilities based on relevance
- Partial matching

https://en.wikipedia.org/wiki/Vector_space_model

Limitations of VSM

- Long documents are poorly represented

https://en.wikipedia.org/wiki/Vector_space_model

Limitations of VSM

- Long documents are poorly represented
- Search keywords must generally be exact

https://en.wikipedia.org/wiki/Vector_space_model

Limitations of VSM

- Long documents are poorly represented
- Search keywords must generally be exact
- Documents with similar contexts but different vocabularies yield missing results (synonymy)

https://en.wikipedia.org/wiki/Vector_space_model

Limitations of VSM

- Long documents are poorly represented
- Search keywords must generally be exact
- Documents with similar contexts but different vocabularies yield missing results (synonymy)
- Words with different meaning can yield bad results (polysemy)

https://en.wikipedia.org/wiki/Vector_space_model

Limitations of VSM

- Long documents are poorly represented
- Search keywords must generally be exact
- Documents with similar contexts but different vocabularies yield missing results (synonymy)
- Words with different meaning can yield bad results (polysemy)
- Term ordering is lost

https://en.wikipedia.org/wiki/Vector_space_model

Distributional Semantics

- **Distributional hypothesis** is based upon the semantic theory of language*.

*https://en.wikipedia.org/wiki/Distributional_semantics

Distributional Semantics

- **Distributional hypothesis** is based upon the semantic theory of language*.

- "i.e. words that are used and occur in the same contexts tend to purport similar meanings" (Harris, Z.)

*https://en.wikipedia.org/wiki/Distributional_semantics

Distributional Semantics

- **Distributional hypothesis** is based upon the semantic theory of language*.
- "i.e. words that are used and occur in the same contexts tend to purport similar meanings" (Harris, Z.)
- "a word is characterized by the company it keeps" (Firth, John R.)

*https://en.wikipedia.org/wiki/Distributional_semantics

Distributional Semantics

- **Distributional hypothesis** is based upon the semantic theory of language*.
- "i.e. words that are used and occur in the same contexts tend to purport similar meanings" (Harris, Z.)
- "a word is characterized by the company it keeps" (Firth, John R.)
- Uses linear algebra for computational and representational purposes

*https://en.wikipedia.org/wiki/Distributional_semantics

Distributional Semantics

- **Distributional hypothesis** is based upon the semantic theory of language*.
- "i.e. words that are used and occur in the same contexts tend to purport similar meanings" (Harris, Z.)
- "a word is characterized by the company it keeps" (Firth, John R.)
- Uses linear algebra for computational and representational purposes
- High-dimensional vectors are built from a corpus

*https://en.wikipedia.org/wiki/Distributional_semantics

Latent Semantic Analysis

Latent Semantic Analysis

- Construct a term-document matrix

Latent Semantic Analysis

- Construct a term-document matrix
- Find a low-rank approximation

Latent Semantic Analysis

- Construct a term-document matrix
- Find a low-rank approximation
  - Matrix is too big

Latent Semantic Analysis

- Construct a term-document matrix
- Find a low-rank approximation
  - Matrix is too big
  - Matrix is too noisy

Latent Semantic Analysis

- Construct a term-document matrix
- Find a low-rank approximation
  - Matrix is too big
  - Matrix is too noisy
  - Matrix is too sparse

LSA Uses

LSA Uses

- Document classification

LSA Uses

- Document classification
- Cross language retrieval

LSA Uses

- Document classification
- Cross language retrieval
- Solutions to synonymy and polysemy

LSA Uses

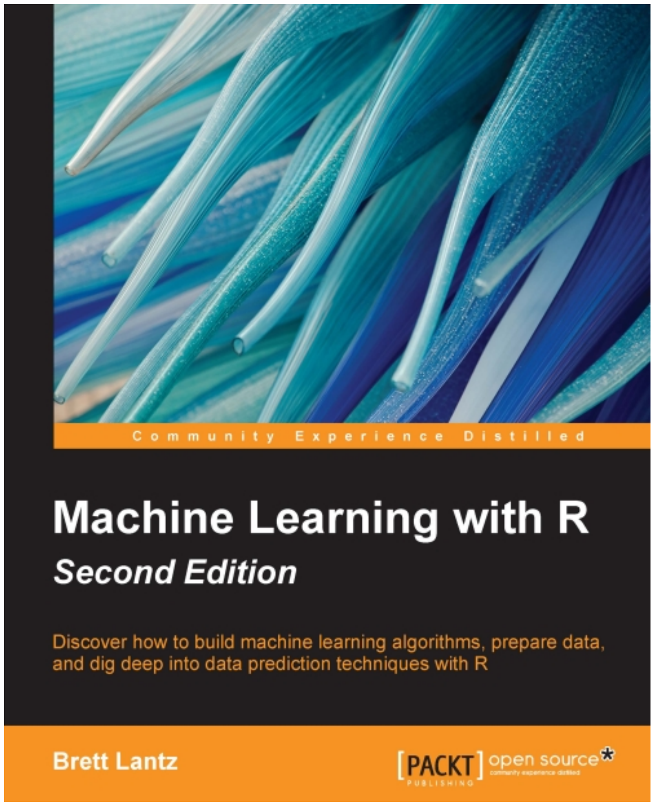
- Document classification
- Cross language retrieval
- Solutions to synonymy and polysemy
- Convert queries into the low-dimensional matrix to find documents

Word2Vec

http://mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_I
of-Words_Model.pdf

https://deeplearning4j.org/word2vec.html

NaiveBayes



https://github.com/dataspelunking/MLwR/

```r
# read the sms data into the sms data frame
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```



```r
# read the sms data into the sms data frame
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```


```r
# examine the structure of the sms data
> str(sms_raw)
'data.frame':   5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in"
 "K..give back my thanks."
 "Am also doing in cbe only. But have to pay."
 "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT
  collection. 09066364349 NOW from Landline not to lose out!"
  | __truncated__ ...
```

```r
# read the sms data into the sms data frame
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```

```r
# examine the structure of the sms data
> str(sms_raw)
'data.frame':	5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in"
 "K..give back my thanks."
 "Am also doing in cbe only. But have to pay."
 "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT
  collection. 09066364349 NOW from Landline not to lose out!"
  | __truncated__ ...
```

```r
# convert spam/ham to factor.
> sms_raw$type <- factor(sms_raw$type)
```



```r
# read the sms data into the sms data frame
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```

```r
# examine the structure of the sms data
> str(sms_raw)
'data.frame':	5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in"
 "K..give back my thanks."
 "Am also doing in cbe only. But have to pay."
 "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT
  collection. 09066364349 NOW from Landline not to lose out!"
  | __truncated__ ...
```

```r
# convert spam/ham to factor.
> sms_raw$type <- factor(sms_raw$type)
```


```r
# examine the type variable more carefully
> str(sms_raw$type)
 Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
```

```
# read the sms data into the sms data frame
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```

```
# examine the structure of the sms data
> str(sms_raw)
'data.frame':	5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in"
 "K..give back my thanks."
 "Am also doing in cbe only. But have to pay."
 "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT
  collection. 09066364349 NOW from Landline not to lose out!"
  | __truncated__ ...
```

```
# convert spam/ham to factor.
> sms_raw$type <- factor(sms_raw$type)
```

```
# examine the type variable more carefully
> str(sms_raw$type)
 Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
```

```
> table(sms_raw$type)

 ham spam
4812  747
```



```
# build a corpus using the text mining (tm) package
> library(tm)
Loading required package: NLP
```

```
# build a corpus using the text mining (tm) package
> library(tm)
Loading required package: NLP
```

```
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```



```
# build a corpus using the text mining (tm) package
> library(tm)
Loading required package: NLP
```

```
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```


```
# examine the sms corpus
> print(sms_corpus)
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5559
```

```
# build a corpus using the text mining (tm) package
> library(tm)
Loading required package: NLP
```

```
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```

```
# examine the sms corpus
> print(sms_corpus)
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5559
```

```
> inspect(sms_corpus[1:2])
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 2

[[1]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 49

[[2]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 23
```



```
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
```

```
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
```

```
> lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."
```



```
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
```

```
> lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."
```


```
> sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
```

```
> lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."
```

```
> sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```
# show the difference between sms_corpus and corpus_clean
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
> as.character(sms_corpus_clean[[1]])
[1] "hope you are having a good week. just checking in"
```



```
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
```

```
> lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."
```

```
> sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```
# show the difference between sms_corpus and corpus_clean
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
> as.character(sms_corpus_clean[[1]])
[1] "hope you are having a good week. just checking in"
```


```
# remove numbers
> sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
# remove stop words
> sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
# remove punctuation
> sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

```r
# tip: create a custom function to replace (rather than remove) punctuation
> removePunctuation("hello...world")
[1] "helloworld"
```



```r
# tip: create a custom function to replace (rather than remove) punctuation
> removePunctuation("hello...world")
[1] "helloworld"
```


```r
> replacePunctuation <- function(x) { gsub("[[:punct:]]+", " ", x) }
> replacePunctuation("hello...world")
[1] "hello world"
```

```
# tip: create a custom function to replace (rather than remove) punctuation
> removePunctuation("hello...world")
[1] "helloworld"
```

```
> replacePunctuation <- function(x) { gsub("[[:punct:]]+", " ", x) }
> replacePunctuation("hello...world")
[1] "hello world"
```

```
# illustration of word stemming
> library(SnowballC)
> wordStem(c("learn", "learned", "learning", "learns"))
[1] "learn" "learn" "learn" "learn"
```



```
# tip: create a custom function to replace (rather than remove) punctuation
> removePunctuation("hello...world")
[1] "helloworld"
```

```
> replacePunctuation <- function(x) { gsub("[[:punct:]]+", " ", x) }
> replacePunctuation("hello...world")
[1] "hello world"
```

```
# illustration of word stemming
> library(SnowballC)
> wordStem(c("learn", "learned", "learning", "learns"))
[1] "learn" "learn" "learn" "learn"
```


```
> sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
# eliminate unneeded whitespace
> sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

```
# examine the final clean corpus
> lapply(sms_corpus[1:3], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."

$`3`
[1] "Am also doing in cbe only. But have to pay."
```



```
# examine the final clean corpus
> lapply(sms_corpus[1:3], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."

$`3`
[1] "Am also doing in cbe only. But have to pay."
```


```
> lapply(sms_corpus_clean[1:3], as.character)
$`1`
[1] "hope you are have a good week. just check in"

$`2`
[1] "k..give back my thanks."

$`3`
[1] "am also do in cbe only. but have to pay."
```

```
# create a document-term sparse matrix
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5559, terms: 10856)>>
Non-/sparse entries: 59144/60289360
Sparsity           : 100%
Maximal term length: 48
Weighting          : term frequency (tf)
```



```
# create a document-term sparse matrix
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5559, terms: 10856)>>
Non-/sparse entries: 59144/60289360
Sparsity           : 100%
Maximal term length: 48
Weighting          : term frequency (tf)
```


```
# creating training and test datasets
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test  <- sms_dtm[4170:5559, ]
```

```
# create a document-term sparse matrix
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5559, terms: 10856)>>
Non-/sparse entries: 59144/60289360
Sparsity           : 100%
Maximal term length: 48
Weighting          : term frequency (tf)
```

```
# creating training and test datasets
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test  <- sms_dtm[4170:5559, ]
```

```
# also save the labels
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type
```



```
# create a document-term sparse matrix
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5559, terms: 10856)>>
Non-/sparse entries: 59144/60289360
Sparsity           : 100%
Maximal term length: 48
Weighting          : term frequency (tf)
```

```
# creating training and test datasets
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test  <- sms_dtm[4170:5559, ]
```

```
# also save the labels
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type
```


```
# check that the proportion of spam is similar
> prop.table(table(sms_train_labels))
sms_train_labels
      ham      spam
0.8647158 0.1352842

> prop.table(table(sms_test_labels))
sms_test_labels
      ham      spam
0.8683453 0.1316547
```

```
> # word cloud visualization
> library(wordcloud)
Loading required package: RColorBrewer
```

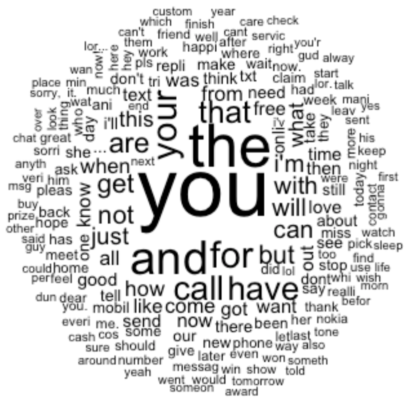


```
> # word cloud visualization
> library(wordcloud)
Loading required package: RColorBrewer
```


```
> wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```

```
> # word cloud visualization
> library(wordcloud)
Loading required package: RColorBrewer
```

```
> wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```




```
# subset the training data into spam and ham groups
> spam <- subset(sms_raw, type == "spam")
> ham  <- subset(sms_raw, type == "ham")
```

```
# subset the training data into spam and ham groups
> spam <- subset(sms_raw, type == "spam")
> ham  <- subset(sms_raw, type == "ham")
```

```
> wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```



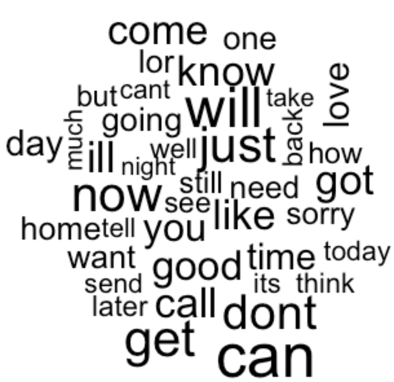
```
# subset the training data into spam and ham groups
> spam <- subset(sms_raw, type == "spam")
> ham  <- subset(sms_raw, type == "ham")
```

```
> wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```


```
> wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



```
> wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```

```
> sms_dtm_freq_train <- removeSparseTerms(sms_dtm_train, 0.999)
```



```
> sms_dtm_freq_train <- removeSparseTerms(sms_dtm_train, 0.999)
```


```
> sms_dtm_freq_train
<<DocumentTermMatrix (documents: 4169, terms: 1306)>>
Non-/sparse entries: 33250/5411464
Sparsity           : 99%
Maximal term length: 24
Weighting          : term frequency (tf)
```

```
> sms_dtm_freq_train <- removeSparseTerms(sms_dtm_train, 0.999)
```

```
> sms_dtm_freq_train
<<DocumentTermMatrix (documents: 4169, terms: 1306)>>
Non-/sparse entries: 33250/5411464
Sparsity           : 99%
Maximal term length: 24
Weighting          : term frequency (tf)
```

```
# save frequently-appearing terms to a character vector
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
```



```
> sms_dtm_freq_train <- removeSparseTerms(sms_dtm_train, 0.999)
```

```
> sms_dtm_freq_train
<<DocumentTermMatrix (documents: 4169, terms: 1306)>>
Non-/sparse entries: 33250/5411464
Sparsity           : 99%
Maximal term length: 24
Weighting          : term frequency (tf)
```

```
# save frequently-appearing terms to a character vector
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
```


```
> str(sms_freq_words)
 chr [1:1332] ":-(" ":-)" "!!''." "..." "....." ...
```

```
> sms_dtm_freq_train <- removeSparseTerms(sms_dtm_train, 0.999)
```

```
> sms_dtm_freq_train
<<DocumentTermMatrix (documents: 4169, terms: 1306)>>
Non-/sparse entries: 33250/5411464
Sparsity           : 99%
Maximal term length: 24
Weighting          : term frequency (tf)
```

```
# save frequently-appearing terms to a character vector
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
```

```
> str(sms_freq_words)
 chr [1:1332] ":-(" ":-)" "!!''." "..." "....." ...
```

```
# create DTMs with only the frequent terms
> sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
> sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```



```
> # convert counts to a factor
> convert_counts <- function(x) {
+     x <- ifelse(x > 0, "Yes", "No")
+ }
```

```r
> # convert counts to a factor
> convert_counts <- function(x) {
+     x <- ifelse(x > 0, "Yes", "No")
+ }
```

```r
# apply() convert_counts() to columns of train/test data
> sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
> sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```



```r
> # convert counts to a factor
> convert_counts <- function(x) {
+     x <- ifelse(x > 0, "Yes", "No")
+ }
```

```r
# apply() convert_counts() to columns of train/test data
> sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
> sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```


```r
## Step 3: Training a model on the data ----
> library(e1071)
> sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```r
> # convert counts to a factor
> convert_counts <- function(x) {
+     x <- ifelse(x > 0, "Yes", "No")
+ }
```

```r
# apply() convert_counts() to columns of train/test data
> sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
> sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

```r
## Step 3: Training a model on the data ----
> library(e1071)
> sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```r
## Step 4: Evaluating model performance ----
> sms_test_pred <- predict(sms_classifier, sms_test)
```



```r
> library(gmodels)
> CrossTable(sms_test_pred, sms_test_labels,
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
+            dnn = c('predicted', 'actual'))


   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1390


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1201 |        31 |      1232 |
             |     0.995 |     0.169 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       152 |       158 |
             |     0.005 |     0.831 |           |
-------------|-----------|-----------|-----------|
Column Total |      1207 |       183 |      1390 |
             |     0.868 |     0.132 |           |
-------------|-----------|-----------|-----------|
```


Questions
brian@bosatsu.net
@bsletten
bsletten
133 / 133