Tutorial for Preprocessing and Feature Selection (categorical data)

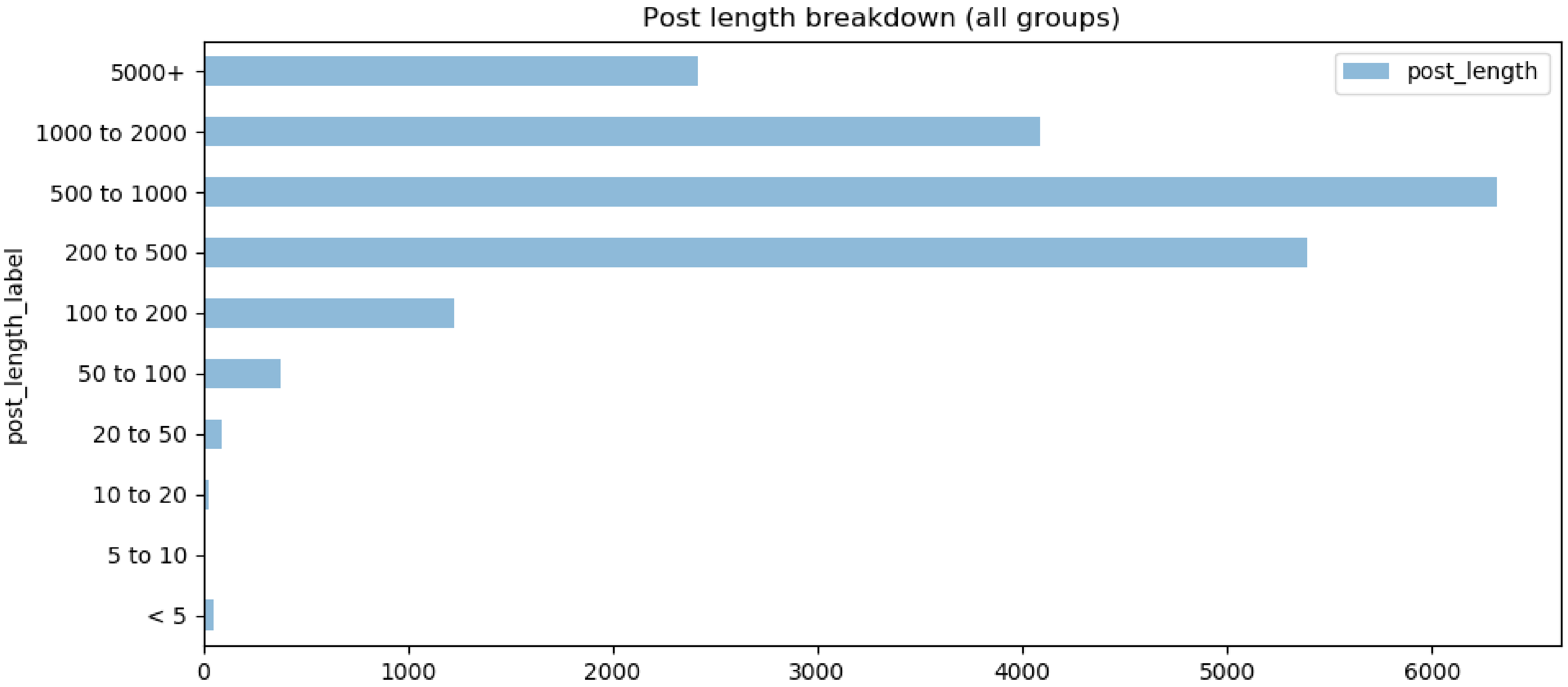
Credits: This material was originally created by Dr Stuart Middleton during previous years of the module.

Contents

- Preprocessing Textual Data
- Feature Selection using Categorical Data
- Classification using Categorical Data

Preprocessing Textual Data

- Example corpus - 20newsgroups
  - 20,000 posts from 20 newsgroups
  - Post text can be 5,000+ characters, but tends to be 500 to 1,000 characters
  - Posts contain UTF-8 encoded free text



Preprocessing Textual Data

- Data Cleaning for text
  - Metadata removal (e.g. HTML markers, formatting)
    Often datasets are not clean and need non-text removed
    BeautifulSoup (bs4) can parse HTML / XML if required
  - Whitespace removal (e.g. tabs, newlines)
    Careful - whitespace is often needed for tokenization later.
  - Spelling mistakes
    If you are working in a known language you can run a spell checker.
    Sometimes people use deliberate typos though!
  - Stemming (e.g. Porter stemming)
    Removes suffixes leaving base of word e.g. Leaving, Leave >> Leav

```
>>> stemmer = nltk.stem.porter.PorterStemmer()
>>> str_stemmed_word = stemmer.stem( 'leaving' )
>>> print( str_stemmed_word )

leav
```

Preprocessing Textual Data

- Data Cleaning for Text
  - Stopword removal
    - We know a-priori words like ('and', 'a', ...) don't add any discriminating value. Stoplists are lists of words we can safely remove.

Careful. We need to remove stopwords, but wait until we tokenize to avoid losing phrases with stopword in middle

e.g. "I'm going to Grab <u>a</u> Snack" != "I'm going to Grab Snack"

```
>>> list_stopwords = nltk.corpus.stopwords.words()
>>> list_stopwords.extend( [ ':', ';', '[', ']', '"', "'", '(', ')', '.', '?',
'#', '@'' ... ] )
```

Preprocessing Textual Data

- Categorical Features from Text
  - Sentence tokenization
    - Split on newlines >> Simple
    - Split on periods >> What about $45.56 ?
    - Use a pre-trained sent tokenizer to handle special cases
  - Word tokenization
    - Split on spaces >> Simple, but what about this;one ?
    - Use a pre-trained word tokenizer to handle special cases

```
>>> str_sent = 'This is a sent. Its got a simple second sent added with a tricky
$45.60 value.'
>>> list_sents = nltk.tokenize.sent_tokenize( text = str_sent )
>>> print( list_sents )
['This is a sent.', 'Its got a simple second sent added with a tricky $45.60
value.']

>>> list_tokens = nltk.tokenize.word_tokenize( text = list_sents[0] )
>>> print( list_tokens )
['This', 'is', 'a', 'sent', '.']
```

Preprocessing Textual Data

- Categorical Features
  - Creation of n-grams
    - Unigram >> Single word e.g. hello
    - Bigram >> Pair of words e.g. hello there
    - Trigram >> Triple of words e.g. hello there people
  - Why limit n-grams to words?
    - Mix in POS and NER tags e.g. I'm in LOCATION
    - Wildcards e.g. I'm in *

```
>>> from nltk.util import ngrams
>>> list_words = [ 'hello', 'there', 'people' ]
>>> list_ngram = list( ngrams( sequence = list_words, n = 2 ) )
>>> print(list_ngram )
[('hello', 'there'), ('there', 'people')]
```

Preprocessing Textual Data

- Transformers
  - Parts of speech (POS) tagging
    - Labels words according to the POS e.g. noun = NN, proper noun = NNP
    - NLTK has a simple POS tagger
    - Stanford CoreNLP has a better POS tagger, but is harder to use

```
>>> str_sent = 'I am in New York. Its cool.'
>>> list_sents = nltk.tokenize.sent_tokenize( text = str_sent )
>>> list_tokens = nltk.tokenize.word_tokenize( text = list_sents[0] )
>>> list_pos = nltk.tag.pos_tag( list_tokens )
>>> print( list_pos )
[('I', 'PRP'), ('am', 'VBP'), ('in', 'IN'), ('New', 'NNP'), ('York', 'NNP'),
('.', '.')]
```

Preprocessing Textual Data

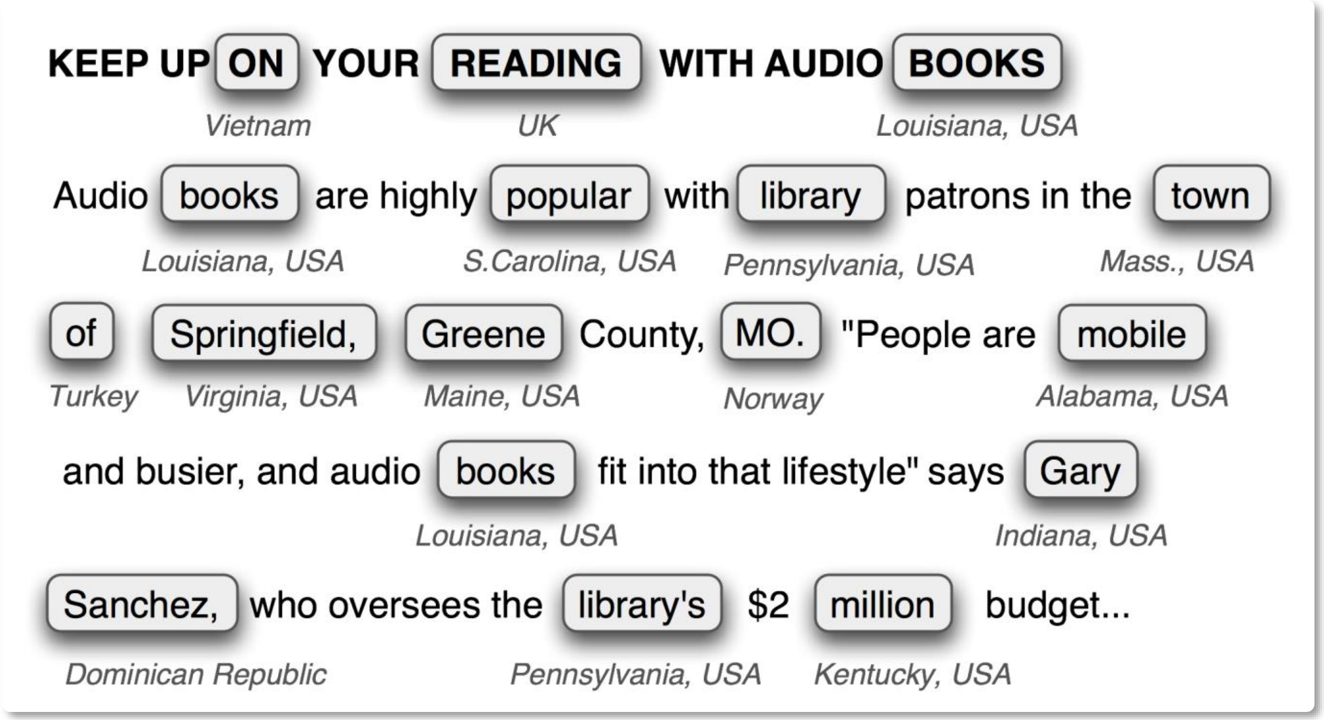
- Transformers
  - Named entity recognition (NER) tagging
    - Labels named entities e.g. New York = GPE (geo-political entity)
    - NLTK has a simple NER tagger
    - Stanford CoreNLP has a better NER tagger, but is harder to use

```
>>> str_sent = 'I am in New York. Its cool.'
>>> list_sents = nltk.tokenize.sent_tokenize( text = str_sent )
>>> list_tokens = nltk.tokenize.word_tokenize( text = list_sents[0] )
>>> list_pos = nltk.tag.pos_tag( list_tokens )
>>> print( list_pos )
[('I', 'PRP'), ('am', 'VBP'), ('in', 'IN'), ('New', 'NNP'), ('York', 'NNP'),
('.', '.')]

>>> tree_sent = nltk.ne_chunk( list_pos )
>>> print( tree_sent )
(S I/PRP am/VBP in/IN (GPE New/NNP York/NNP) ./.)
```

Preprocessing Textual Data

NE Type	Examples
ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75 %
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian



Source: NLTK website, https://www.nltk.org/book/ch07.html

Preprocessing Textual Data

1st post on 20newsgroupws corpus

Text

Archive-name: atheism/resources Alt-atheism-archive-name: resources Last-modified: 11 December 1992 Version: 1.0

Atheist Resources

Addresses of Atheist Organizations

USA

FREEDOM FROM RELIGION FOUNDATION

Darwin fish bumper stickers and assorted other atheist paraphernalia are available from the Freedom From Religion Foundation in the US.

...

Tokens

['Darwin', 'fish', 'bumper', 'stickers', 'and', 'assorted', 'other', 'atheist', 'paraphernalia', 'are', 'available', 'from', 'the', 'Freedom', 'From', 'Religion', 'Foundation', 'in', 'the', 'US', '.']

Preprocessing Textual Data

POS

```
[('Darwin', 'NNP'), ('fish', 'JJ'), ('bumper', 'NN'), ('stickers', 'NNS'),
('and', 'CC'), ('assorted', 'VBD'), ('other', 'JJ'), ('atheist', 'JJ'),
('paraphernalia', 'NNS'), ('are', 'VBP'), ('available', 'JJ'), ('from', 'IN'),
('the', 'DT'), ('Freedom', 'NN'), ('From', 'NNP'), ('Religion', 'NNP'),
('Foundation', 'NNP'), ('in', 'IN'), ('the', 'DT'), ('US', 'NNP'), ('.', '.')]
```

NER

```
{
        'GPE': ['Darwin'],
        'ORGANIZATION': ['Freedom From Religion Foundation'],
        'GSP': ['US']
}
```

Feature Selection using Categorical Data

- Indexing Categorical Labels
  - Convert enumerated text labels to integer indexes

    Why? Because many ML algorithms only work with a numeric vectors or numeric matrices
  - Create a feature index and inverted index containing tokens, POS, NER and n-gram versions

    Why? Allows lookup from feature text <--> feature index

Preprocessing Textual Data

Feature list

```
['Darwin fish', 'fish bumper', 'bumper stickers', 'stickers and', 'and
assorted', 'assorted other', 'other atheist', 'atheist paraphernalia',
'paraphernalia are', 'are available', 'available from', 'from the', 'the
Freedom', 'Freedom From', 'From Religion', 'Religion Foundation', 'Foundation
in', 'in the', 'the US', 'US .', 'Darwin fish bumper', 'fish bumper stickers',
'bumper stickers and', 'stickers and assorted', 'and assorted other', 'assorted
other atheist', 'other atheist paraphernalia', 'atheist paraphernalia are',
'paraphernalia are available', 'are available from', 'available from the', 'from
the Freedom', 'the Freedom From', 'Freedom From Religion', 'From Religion
Foundation', 'Religion Foundation in', 'Foundation in the', 'in the US', 'the US
.']
```

Feature inverted index

```
{ 'Darwin fish' : 0,
  'fish bumper' : 1,
  'bumper stickers' : 2,
  'stickers and' : 3,
  ...
}
```

Feature index

```
[ 'Darwin fish',
  'fish bumper',
  'bumper stickers',
  'stickers and',
  ...
]
```

Feature Selection using Categorical Data

- Count Vectors
  - Define what a <u>document</u> and <u>term</u> means
  - Count Vector = [[$Freq_{Term1Doc1}$,$Freq_{Term2Doc1}$, ...] x N_{term}, ...] x N_{doc}
  - Examples
    - Document = aggregation of sentences from forum topic = Class
    - Term = n-gram phrase feature = Feature

    - Document = online blog article from a author = Class
    - Term = n-gram feature = Feature

- Feature Selection
  - Metrics to score and rank features in terms of discriminating power

  - TF(term) = Term frequency = # occurrences of term in corpus
  - TF(doc,term) = # occurrences of term in document
  - DF(term) = Document frequency = # of documents the term appears in
  - IDF(term) = Inverse DF smoothed = log(N_{doc} / (DF(term) + 1))
  - TF-IDF = TF(doc,term) * IDF(term)

Feature Selection using Categorical Data

- Sklearn TfidfTransformer
    - note: IDF(term) = Inverse DF sklearn = log(N_{doc} / DF(term)) + 1

```
list_features = ['Darwin fish', 'fish bumper', 'bumper stickers', ... ]
array_count_vector = [
                  [ 0,5,0, ... ],
                  [ 1,0,0, ... ],
                  [ 0,5,0, ... ],
                  ...
         ]
```

← Term #2 (fish bumper) occurs 5 times in Doc # 1

```
transformer = sklearn.feature_extraction.text.TfidfTransformer()
transformer.fit( array_count_vector )
```

Learns the IDF vector

Feature Selection using Categorical Data

```
df_idf = pandas.DataFrame( data = transformer.idf_, index = list_features,
columns = ['idf_weights'] )
df_idf = df_idf.sort_values( by=['idf_weights'], ascending = False )

print( df_idf[0:20] )
```

```
                          idf_weights
wakefield , who           3.995732
disolve the silver        3.995732
silver off of             3.995732
of the ceramic            3.995732
the ceramic .             3.995732
silver was                3.995732
quite thin                3.995732
the silver was            3.995732
...
```

IDF on its own is not a great discriminator for large documents with many terms

Feature Selection using Categorical Data

```
tf_idf = transformer.transform( array_count_vector )
```

```
Computes the TF-IDF matrix
```

```
tf_idf_vector = tf_idf[0]
df_tf_idf = pandas.DataFrame( tf_idf_vector.T.todense(), index=list_features,
columns=['tfidf'] )
df_tf_idf = df_tf_idf.sort_values( by=["tfidf"], ascending=False )

print( df_tf_idf[:20], '\n' )
```

```
talk.politics.Mideast

                  tfidf
the               0.631129
of                0.318983
and               0.266981
to                0.253070
a                 0.162897
armenian          0.151492
that              0.140341
turkish           0.112215
armenians         0.102386
is                0.101752
israel            0.079450
armenia           0.077385
```

sign that a Stopword filter is needed

these features look more topic specific

Classification using Categorical Data

- Random Forest Classifier
  - Simple bagging classifier to show feature selection used for classification
  - Split the training and test dataset e.g. 90% train, 10% test
  - Select topN features according to TF-IDF >> Feature Subset
  - Calc post occurrence freq of features in Feature Subset

```
X = [
        [ 0,2,0,... ],
        [ 0,0,1,... ],
        ...
    ]
```

← Feature #2 occurs 2 times in post # 1

  - Label each document

```
Y = [ 0,1,1,2,0, ... ]
```

← Post #2 is a member of doc class #1

Classification using Categorical Data

- Random Forest Classifier
  - Training (fit)

```
rf = RandomForestClassifier( n_estimators = 500, max_leaf_nodes = 16 )
rf.fit( X_train, Y_train )
```

Build a forest of trees

  - Testing (predict)

```
Y_predict = rf.predict( X_test )
```

Predict class for X

Classification using Categorical Data

- Evaluating
  - Computing precision of predictions using ground truth labels from test set

```
TopN Features 1000
Post     Predict                    Ground truth
0        talk.politics.misc         talk.politics.misc
1        talk.politics.mideast      talk.politics.mideast
2        rec.sport.baseball         sci.space
3        comp.windows.x             comp.windows.x
4        soc.religion.christian     soc.religion.christian
5        talk.politics.guns         talk.politics.guns
6        rec.sport.baseball         rec.sport.baseball
...

TP =  2122  FP =  1878  P =  0.5305
```

Classification using Categorical Data

- Try the example code for yourself

- Ideas to learn more
  - Read the 'find out more' URI's provided in the code example
  - See how changes in feature selection strategy changes feature quality
  - Think about why this might be, test your own hypotheses out
  - Try other classifier types
  - Look at one-hot encoded vectors (Chapter 2, page 64) for instance-based classifiers with similarity distance measures