Introduction to Text Encoding

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science,
Ordergroove

Standardizing your text

Example of free text:

> Fellow-Citizens of the Senate and of the House of Representatives: AMONG the vicissitudes incident to life no event could have filled me with greater anxieties than that of which the notification was transmitted by your order, and received on the th day of the present month.

Dataset

```
print(speech_df.head())
```

```
                  Name         Inaugural Address    \
0    George Washington     First Inaugural Address
1    George Washington    Second Inaugural Address
2    John Adams                  Inaugural Address
3    Thomas Jefferson      First Inaugural Address
4    Thomas Jefferson     Second Inaugural Address


                        Date                           text
0    Thursday, April 30, 1789    Fellow-Citizens of the Sena...
1       Monday, March 4, 1793    Fellow Citizens: I AM again...
2     Saturday, March 4, 1797    WHEN it was first perceived...
3    Wednesday, March 4, 1801    Friends and Fellow-Citizens...
4       Monday, March 4, 1805    PROCEEDING, fellow-citizens...
```

Removing unwanted characters

- `[a-zA-Z]` : All letter characters
- `[^a-zA-Z]` : All non letter characters

```
speech_df['text'] = speech_df['text']\
                    .str.replace('[^a-zA-Z]', ' ')
```

Removing unwanted characters

Before:

```
"Fellow-Citizens of the Senate and of the House of
Representatives: AMONG the vicissitudes incident to
life no event could have filled me with greater" ...
```

After:

```
"Fellow Citizens of the Senate and of the House of
Representatives AMONG the vicissitudes incident to
life no event could have filled me with greater" ...
```

Standardize the case

```
speech_df['text'] = speech_df['text'].str.lower()
print(speech_df['text'][0])
```

```
"fellow citizens of the senate and of the house of
representatives among the vicissitudes incident to
life no event could have filled me with greater"...
```

Length of text

```
speech_df['char_cnt'] = speech_df['text'].str.len()
print(speech_df['char_cnt'].head())
```

```
0    1889
1     806
2    2408
3    1495
4    2465
Name: char_cnt, dtype: int64
```

Word counts

```
speech_df['word_cnt'] =
    speech_df['text'].str.split()
speech_df['word_cnt'].head(1)
```

```
['fellow', 'citizens', 'of', 'the', 'senate', 'and',...
```

Word counts

```
speech_df['word_counts'] =
    speech_df['text'].str.split().str.len()
print(speech_df['word_splits'].head())
```

```
0    1432
1     135
2    2323
3    1736
4    2169
Name: word_cnt, dtype: int64
```

Average length of word

```
speech_df['avg_word_len'] =
          speech_df['char_cnt'] / speech_df['word_cnt']
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Word Count Representation

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Text to columns

“citizens of the senate and of the house of representatives”

Index	citizens	of	the	senate	and	house	representatives
1	1	3	2	1	1	1	1

Initializing the vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
print(cv)
```

```
CountVectorizer(analyzer=u'word', binary=False,
        decode_error=u'strict',
        dtype=<type 'numpy.int64'>,
        encoding=u'utf-8', input=u'content',
        lowercase=True, max_df=1.0, max_features=None,
        min_df=1,ngram_range=(1, 1), preprocessor=None,
        stop_words=None, strip_accents=None,
        token_pattern=u'(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None
```

Specifying the vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0.1, max_df=0.9)
```

`min_df` : minimum fraction of documents the word must occur in

`max_df` : maximum fraction of documents the word can occur in

Fit the vectorizer

```
cv.fit(speech_df['text_clean'])
```

Transforming your text

```
cv_transformed = cv.transform(speech_df['text_clean'])
print(cv_transformed)
```

```
<58x8839 sparse matrix of type '<type 'numpy.int64'>'
```

Transforming your text

```
cv_transformed.to_array()
```

Getting the features

```
feature_names = cv.get_feature_names()
print(feature_names)
```

```
[u'abandon', u'abandoned', u'abandonment', u'abate',
u'abdicated', u'abeyance', u'abhorring', u'abide',
u'abiding', u'abilities', u'ability', u'abject'...
```

Fitting and transforming

```
cv_transformed = cv.fit_transform(speech_df['text_clean'])
print(cv_transformed)
```

```
<58x8839 sparse matrix of type '<type 'numpy.int64'>'
```

Putting it all together

```
cv_df = pd.DataFrame(cv_transformed.toarray(),
                     columns=cv.get_feature_names())\
                              .add_prefix('Counts_')
print(cv_df.head())
```

```
     Counts_aback    Counts_abandoned    Counts_a...
0               1                   0         ...
1               0                   0         ...
2               0                   1         ...
3               0                   1         ...
```

[1] ```out Counts_aback Counts_abandon Counts_abandonment 0 1 0 0 1 0 0 1 2 0 1 0 3 0 1 0 4 0 0 0 ```

Updating your DataFrame

```
speech_df = pd.concat([speech_df, cv_df],
                      axis=1, sort=False)
print(speech_df.shape)
```

```
(58, 8845)
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Tf-Idf Representation

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Introducing TF-IDF

```
print(speech_df['Counts_the'].head())
```

```
0     21
1     13
2     29
3     22
4     20
```

TF-IDF

$$\text{TF-IDF} = \frac{\dfrac{\textit{Count of word occurances}}{\textit{Total words in document}}}{\log\left(\dfrac{\textit{Number of docs word is in}}{\textit{Total number of docs}}\right)}$$

Importing the vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer()
print(tv)
```

```
TfidfVectorizer(analyzer=u'word', binary=False, decode_erro
        dtype=<type 'numpy.float64'>, encoding=u'utf-8', in
        lowercase=True, max_df=1.0, max_features=None, min_
        ngram_range=(1, 1), norm=u'l2', preprocessor=None,
        stop_words=None, strip_accents=None, sublinear_tf=F
        token_pattern=u'(?u)\\b\\w\\w+\\b', tokenizer=None,
        vocabulary=None)
```

Max features and stopwords

```
tv = TfidfVectorizer(max_features=100,
                     stop_words='english')
```

`max_features` : Maximum number of columns created from TF-IDF

`stop_words` : List of common words to omit e.g. "and", "the" etc.

Fitting your text

```
tv.fit(train_speech_df['text'])
train_tv_transformed = tv.transform(train_speech_df['text']
```

Putting it all together

```
train_tv_df = pd.DataFrame(train_tv_transformed.toarray(),
                           columns=tv.get_feature_names())\
                                .add_prefix('TFIDF_')

train_speech_df = pd.concat([train_speech_df, train_tv_df],
                            axis=1, sort=False)
```

Inspecting your transforms

```
examine_row = train_tv_df.iloc[0]
```

```
print(examine_row.sort_values(ascending=False))
```

```
TFIDF_government    0.367430
TFIDF_public        0.333237
TFIDF_present       0.315182
TFIDF_duty          0.238637
TFIDF_citizens      0.229644
Name: 0, dtype: float64
```

Applying the vectorizer to new data

```
test_tv_transformed = tv.transform(test_df['text_clean'])

test_tv_df = pd.DataFrame(test_tv_transformed.toarray(),
                          columns=tv.get_feature_names())\
                            .add_prefix('TFIDF_')

test_speech_df = pd.concat([test_speech_df, test_tv_df],
                           axis=1, sort=False)
```

Let's practice!

Bag of words and N-grams

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Issues with bag of words

Positive meaning

Single word: *happy*

Negative meaning

Bi-gram : *not happy*

Positive meaning

Trigram : *never not happy*

Using N-grams

```
tv_bi_gram_vec = TfidfVectorizer(ngram_range = (2,2))

# Fit and apply bigram vectorizer
tv_bi_gram = tv_bi_gram_vec\
               .fit_transform(speech_df['text'])

# Print the bigram features
print(tv_bi_gram_vec.get_feature_names())
```

```
[u'american people', u'best ability ',
 u'beloved country', u'best interests' ... ]
```

Finding common words

```
# Create a DataFrame with the Counts features
tv_df = pd.DataFrame(tv_bi_gram.toarray(),
                     columns=tv_bi_gram_vec.get_feature_names())\
                        .add_prefix('Counts_')

tv_sums = tv_df.sum()
print(tv_sums.head())
```

```
Counts_administration government    12
Counts_almighty god                 15
Counts_american people              36
Counts_beloved country               8
Counts_best ability                  8
dtype: int64
```

Finding common words

```
print(tv_sums.sort_values(ascending=False)).head()
```

```
Counts_united states          152
Counts_fellow citizens         97
Counts_american people         36
Counts_federal government      35
Counts_self government         30
dtype: int64
```

Let's practice!

Wrap-up

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Chapter 1

- How to understand your data types
- Efficient encoding or categorical features
- Different ways to work with continuous variables

Chapter 2

- How to locate gaps in your data
- Best practices in dealing with the incomplete rows
- Methods to find and deal with unwanted characters

Chapter 3

- How to observe your data's distribution
- Why and how to modify this distribution
- Best practices of finding outliers and their removal

Chapter 4

- The foundations of word embeddings
- Usage of Term Frequency Inverse Document Frequency (Tf-idf)
- N-grams and its advantages over bag of words

Next steps

- Kaggle competitions
- More DataCamp courses
- Your own project

Thank You!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON