Supervised sentiment analysis: Feature representation

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding
N-gram feature functions

- Unigrams: the basis for “bag-of-words” models
- Easily generalized to “bag of-ngrams”
- Highly dependent on the tokenization scheme
- Can be combined with preprocessing steps like ‘_NEG’ marking
- Creates very large, very sparse feature representations
- Generally fails to directly model relationships between features
# Feature functions vs. features

```python
[1]: from collections import Counter
    import numpy as np
    import pandas as pd
    from sklearn.feature_extraction import DictVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.utils.extmath import softmax
    import sst

[2]: def unigrams_phi(text):
    return Counter(text.lower().split())

[3]: example_texts = ["a a a", "a a b", "a b b", "b b b"]

[4]: feats = [unigrams_phi(text) for text in example_texts]

[5]: vec = DictVectorizer(sparse=False)

[6]: X = vec.fit_transform(feats)

[7]: pd.DataFrame(X, columns=vec.get_feature_names())

[7]:
   a   b
  0  3.0  0.0
  1  2.0  1.0
  2  1.0  2.0
  3  0.0  3.0
```
Feature functions vs. features

```
[7]: pd.DataFrame(X, columns=vec.get_feature_names())

[7]:
   a   b
    0  3.0  0.0
    1  2.0  1.0
    2  1.0  2.0
    3  0.0  3.0

[8]: y = ['C1', 'C1', 'C2', 'C3']

[9]: mod = LogisticRegression()

[10]: mod.fit(X, y)

[10]: LogisticRegression()

[11]: pd.DataFrame(mod.coef_, index=mod.classes_, columns=vec.get_feature_names())

[11]:
   a   b
  C1  0.567932 -0.567932
  C2 -0.071105  0.071103
  C3 -0.496827  0.496829

[12]: softmax(X.dot(mod.coef_.T) + mod.intercept_)

[12]: array([[0.90606849, 0.08182458, 0.01210693],
           [0.69610577, 0.22566175, 0.07823248],
           [0.32165061, 0.37430625, 0.30404314],
           [0.07617433, 0.31820816, 0.60561751]])

[13]: mod.predict_proba(X)
```
Other ideas for hand-built feature functions

- Lexicon-derived features
- Negation marking
- Modal adverbs:
  - “It is quite possibly a masterpiece.”
  - “It is totally amazing.”
- Length based features
- Thwarted expectations: ratio of positive to negative words
  - “Many consider the movie bewildering, boring, slow-moving or annoying.”
  - “It was hailed as a brilliant, unprecedented artistic achievement worthy of multiple Oscars.”
- Non-literal language:
  - “Not exactly a masterpiece.”
  - “Like 50 hours long.”
  - “The best movie in the history of the universe.”
Assessing individual feature functions

1. `sklearn.feature_selection` offers functions to assess how much information your feature functions contain with respect to your labels.

2. Take care when assessing feature functions individually; correlations between them will make these assessments hard to interpret:

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F</td>
</tr>
</tbody>
</table>

   Chi-squared scores:
   - $\chi^2(X_1, y) = 3$
   - $\chi^2(X_2, y) = 0.33$
   - $\chi^2(X_3, y) = 0.2$

3. Consider more holistic assessment methods: systematically removing or disrupting features in the context of a full model and comparing performance before and after.

What do the scores tell us about the best model? In truth, a linear model performs best with just $X_1$, and including $X_2$ hurts.
Distributed representations as features

Classifier prediction

Lexical vectors combined via a function like sum or mean. These are the inputs to the classifier.

Embedding look-up
Distributed representations as features

```python
[1]: import numpy as np
    import os
    from sklearn.linear_model import LogisticRegression
    import sst
    import utils

[2]: GLOVE_HOME = os.path.join('data', 'glove.6B')
    SST_HOME = os.path.join('data', 'sentiment')

[3]: glove_lookup = utils.glove2dict(os.path.join(GLOVE_HOME, 'glove.6B.300d.txt'))

[4]: def vsm_leaves_phi(text, lookup, np_func=np.mean):
    allvecs = np.array([lookup[w] for w in text.lower().split() if w in lookup])
    if len(allvecs) == 0:
        dim = len(next(iter(lookup.values())))
        feats = np.zeros(dim)
    else:
        feats = np_func(allvecs, axis=0)
    return feats

[5]: def glove_leaves_phi(text, np_func=np.mean):
    return vsm_leaves_phi(text, glove_lookup, np_func=np_func)

[6]: def fit_softmax(X, y):
    mod = LogisticRegression(
        fit_intercept=True, solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod

[7]: glove_sum_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    glove_leaves_phi,
    fit_softmax,
    vectorize=False)  # Tell 'experiment' it needn't use a DictVectorizer.
```