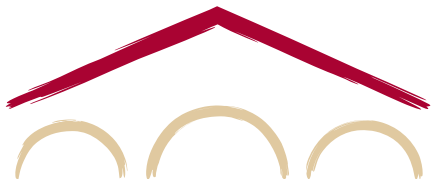
Supervised sentiment analysis: sst.py

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding



Readers

[1]:
```
import os
import sst
```

[2]:
```
SST_HOME = os.path.join('data', 'sentiment')
```

[3]:
```
train_df = sst.train_reader(SST_HOME, include_subtrees=False, dedup=False)
```

[4]:
```
train_df.sample(1, random_state=1).to_dict(orient="records")
```

[4]:
```
[{'example_id': '04162-00001',
  'sentence': "One can only assume that the jury who bestowed star Hoffman 's
brother Gordy with the Waldo Salt Screenwriting award at 2002 's Sundance
Festival were honoring an attempt to do something different over actually
pulling it off",
  'label': 'negative',
  'is_subtree': 0}]
```

[5]:
```
train_df.label.value_counts()
```

[5]:
```
positive    3610
negative    3310
neutral     1624
Name: label, dtype: int64
```

[6]:
```
dev_df = sst.dev_reader(SST_HOME)
```

[7]:
```
dev_df.label.value_counts()
```

[7]:
```
positive    444
negative    428
neutral     229
Name: label, dtype: int64
```

Feature functions

[1]:
```
from collections import Counter
import sst
```

[2]:
```
def unigrams_phi(text):
    """The basis for a unigrams feature function. Downcases all tokens.

    Parameters
    ----------
    text : str
        The example to represent.

    Returns
    -------
    defaultdict
        A map from strings to their counts in `text`. (Counter maps a
        list to a dict of counts of the elements in that list.)

    """
    return Counter(text.lower().split())
```

[3]:
```
example_text = "NLU is enlightening !"
```

[4]:
```
unigrams_phi(example_text)
```

[4]:
```
Counter({'nlu': 1, 'is': 1, 'enlightening': 1, '!': 1})
```

Model wrappers

[5]:
```
from sklearn.linear_model import LogisticRegression
```

[6]:
```
def fit_softmax_classifier(X, y):
    """Wrapper for `sklearn.linear.model.LogisticRegression`. This is
    also called a Maximum Entropy (MaxEnt) Classifier, which is more
    fitting for the multiclass case.

    Parameters
    ----------
    X : 2d np.array
        The matrix of features, one example per row.
    y : list
        The list of labels for rows in `X`.

    Returns
    -------
    sklearn.linear.model.LogisticRegression
        A trained `LogisticRegression` instance.

    """
    mod = LogisticRegression(
        fit_intercept=True,  solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod
```

sst.experiment

[7]:
```
import os
import utils
```

[8]:
```
SST_HOME = os.path.join('data', 'sentiment')
```

[9]:
```
unigrams_softmax_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    unigrams_phi,
    fit_softmax_classifier,
    assess_dataframes=None,              # The default
    train_size=0.7,                      # The default
    score_func=utils.safe_macro_f1,      # The default
    vectorize=True,                      # The default
    verbose=True)                        # The default
```

```
                precision    recall  f1-score   support

      negative      0.634     0.662     0.648      1010
       neutral      0.289     0.144     0.192       479
      positive      0.646     0.764     0.700      1075

      accuracy                          0.608      2564
     macro avg      0.523     0.523     0.513      2564
  weighted avg      0.575     0.608     0.585      2564
```

sst.experiment

[7]:
```
import os
import utils
```

[8]:
```
SST_HOME = os.path.join('data', 'sentiment')
```

[9]:
```
unigrams_softmax_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    unigrams_phi,
    fit_softmax_classifier,
    assess_dataframes=None,              # The default
    train_size=0.7,                      # The default
    score_func=utils.safe_macro_f1,      # The default
    vectorize=True,                      # The default
    verbose=True)                        # The default
```

	precision	recall	f1-score	support
negative	0.634	0.662	0.648	1010
neutral	0.289	0.144	0.192	479
positive	0.646	0.764	0.700	1075
accuracy			0.608	2564
macro avg	0.523	0.523	0.513	2564
weighted avg	0.575	0.608	0.585	2564

Our default metric for almost all our work: gives equal weight to all classes regardless of size, while balancing precision and recall.

sst.experiment

The return value of `sst.experiment` is a `dict` packaging up the objects and info needed to test this model in new settings and conduct deep error analysis:

```
[10]: list(unigrams_softmax_experiment.keys())

[10]: ['model',
       'phi',
       'train_dataset',
       'assess_datasets',
       'predictions',
       'metric',
       'scores']

[11]: list(unigrams_softmax_experiment['train_dataset'].keys())

[11]: ['X', 'y', 'vectorizer', 'raw_examples']
```

Bringing it all together

[1]:
```
from collections import Counter
import os
from sklearn.linear_model import LogisticRegression
import sst
```

[2]:
```
SST_HOME = os.path.join('data', 'sentiment')
```

[3]:
```
def phi(text):
    return Counter(text.lower().split())
```

[4]:
```
def fit_model(X, y):
    # X, y to a model a fitted model with a predict method.
    mod = LogisticRegression(
        fit_intercept=True,  solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod
```

[5]:
```
experiment = sst.experiment(sst.train_reader(SST_HOME), phi, fit_model)
```

sklearn.feature_extraction.DictVectorizer

[1]:
```
import pandas as pd
from sklearn.feature_extraction import DictVectorizer
```

[2]:
```
train_feats = [
    {'a': 1, 'b': 1},
    {'b': 1, 'c': 2}]
```

[3]:
```
vec = DictVectorizer(sparse=False) # Use `sparse=True` for real problems!
```

[4]:
```
X_train = vec.fit_transform(train_feats)
```

[5]:
```
pd.DataFrame(X_train, columns=vec.get_feature_names())
```

[5]:

	a	b	c
0	1.0	1.0	0.0
1	0.0	1.0	2.0

[6]:
```
test_feats = [
    {'a': 2},
    {'a': 4, 'b': 2, 'd': 1}]
```

[7]:
```
X_test = vec.transform(test_feats) # Not `fit_transform`!
```

[8]:
```
pd.DataFrame(X_test, columns=vec.get_feature_names())
```

[8]:

	a	b	c
0	2.0	0.0	0.0
1	4.0	2.0	0.0