Supervised sentiment analysis: Hyperparameter search and classifier comparison

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding
Hyperparameter search
Hyperparameter search: Rationale

1. The **parameters** of a model are those whose values are learned as part of optimizing the model itself.

2. The **hyperparameters** of a model are any settings that are set outside of this optimization. Examples:
   a. GloVe or LSA dimensionality
   b. GloVe $x_{\text{max}}$ and $\alpha$
   c. Regularization terms, hidden dimensionalities, learning rates, activation functions
   d. Optimization methods

3. Hyperparameter optimization is crucial to building a persuasive argument: every model must be put in its best light!

4. All hyperparameter tuning must be done only on train and development data.
Hyperparameter search in sst.py

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

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

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

[4]: def fit_softmax_with_search(X, y):
    basemod = LogisticRegression(solver='liblinear', multi_class='auto')
    cv = 5
    param_grid = {'fit_intercept': [True, False],
                  'C': [0.4, 0.6, 0.8, 1.0, 2.0, 3.0],
                  'penalty': ['l1', 'l2']}
    best_mod = utils.fit_classifier_with_hyperparameter_search(
                X, y, basemod, cv, param_grid)
    return best_mod

[5]: xval = sst.experiment(sst.train_reader(SST_HOME), phi, fit_softmax_with_search)

Best params: {'C': 2.0, 'fit_intercept': False, 'penalty': 'l2'}
Best score: 0.513

precision  recall  f1-score  support
```
Classifier comparison
Classifier comparison: Rationale

1. Suppose you’ve assessed a baseline model $B$ and your favored model $M$, and your chosen assessment metric favors $M$. Is $M$ really better?

2. If the difference between $B$ and $M$ is clearly of practical significance, then you might not need to do anything beyond presenting the numbers. Still, is there variation in how $B$ or $M$ performs?

3. Demšar (2006) advises the Wilcoxon signed-rank test for situations in which you can afford to repeatedly assess $B$ and $M$ on different train/test splits. We’ll talk later in the term about the rationale for this.

4. For situations where you can’t repeatedly assess $B$ and $M$, McNemar’s test is a reasonable alternative. It operates on the confusion matrices produced by the two models, testing the null hypothesis that the two models have the same error rate.
Classifier comparison in sst.py

```python
[1]: from collections import Counter
    import os
    import scipy.stats
    from sklearn.linear_model import LogisticRegression
    from sklearn.naive_bayes import MultinomialNB
    import sst
    import utils

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

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

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

[5]: def fit_naivebayes(X, y):
    mod = MultinomialNB(fit_prior=True)
    mod.fit(X, y)
    return mod
```
Classifier comparison in sst.py

Wilcoxon signed rank test

```
[6]: mod1_scores, mod2_scores, p = sst.compare_models(
    sst.train_reader(SST_HOME),
    phi1=phi,
    phi2=\textbf{None}, \textcolor{red}{# Defaults to `phi1`}
    train_func1=fit_softmax,
    train_func2=fit_naivebayes, \textcolor{red}{# Defaults to `train_func1`}
    stats_test=scipy.stats.wilcoxon, \textcolor{red}{# Default}
    trials=10, \textcolor{red}{# Default}
    train_size=0.7, \textcolor{red}{# Default}
    score_func=utils.safe_macro_f1) \textcolor{red}{# Default}

Model 1 mean: 0.521
Model 2 mean: 0.493
p = 0.002
```
Classifier comparison in sst.py

McNemar’s test

[7]: softmax_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    phi,
    fit_softmax,
    verbose=False)

[8]: naivebayes_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    phi,
    fit_naivebayes,
    verbose=False)

[9]: stat, p = utils.mcnemar(
    softmax_experiment['assess_datasets'][0]['y'],
    naivebayes_experiment['predictions'][0],
    softmax_experiment['predictions'][0])