Homework 03 - Feature Engineering

James Folberth   Kaggle username: cusymplectic   Kaggle display name: jafo5560

September 17, 2015

Remark: *During development, I mistakenly used sklearn's* `GridSearchCV` *and adjusted the* `n_iter` *and* `alpha` *parameters to* `SGDClassifie` *I misunderstood the limits of "don't change the underlying algorithm". This has been removed from my code, and I am using exactly* `lr = SGDClassifier(loss='log', penalty='l2', shuffle=True)` *as of 17 September morning. I brought this to JBG's attention on Piazza; he said to just change my code appropriately, and only use my updated submissions on Kaggle. I apologize again for this mistake. Please contact me with any questions.*

I started off by making an argument to easily split the training data (`train.csv`) into two parts: a development and a test set. Note that when I refer to the "test set" below, I'm referring to my development test set, not the public Kaggle test set coming from `test.csv`. I usually used 80% to 90% of `train.csv` to make the development set, as that resembled the ratios for `train.csv` and `test.csv`. Before the data were split, the indices were shuffled. I modified the provided code to print the top 20 features for each class, and my features for a few misclassified examples.

When I got the original code to run, I noticed words like `kill`, `killing`, `killed` were highly weighted features. Following what was suggested in class, I stemmed these words with the help of a part of speech tagger. This improved slightly the accuracy on the test set. Similar to the script provided and used in the feature engineering lecture, I implemented an analyzer for sklearn's `CountVectorizer` to use to generate features. I tagged each stemmed word and made n-grams out of the words that were not in a list of English stop words (e.g. `the`, `of`, `i`). In order to reduce overfitting, I set the maximal number of features coming out of `CountVectorizer` to about 100000; the exact number varied across development.

```
Using (1,2,3,4)-grams:
conf mat on test set:
Accuracy: 0.684246
False True
475   259
208   537
```

```
(1,2,3,4)-grams and Tf-Idf weighting:
conf mat on test set:
Accuracy: 0.688303
False True
428   306
155   590
```

Features chosen by `CountVectorizer` are sorted by frequency, so very frequent features (even if not "informative") are included in feature space. This is a bit misleading for the classifier, and additionally wastes computational resources. I used sklearn's `TfidfTransformer` to weight the features. This improved classification accuracy marginally, but significantly reduced overfitting as measured by the difference between the accuracy on the development set and test set (by almost 10% in difference!). That's good news, as it should reduce generalization error.

I noticed quite a few names (e.g. `sherlock`, `olivia`, `jim`) that were heavily weighted. I downloaded a list of common first names and removed the names, but this reduced my test set error by a few percent. I removed the name removing part of my code. Even though I though it would work well (and I'm not entirely sure why it didn't; perhaps my implementation isn't good.), my tests indicated that it didn't work well.

Since I was estimating the part of speech for stemming, I thought I'd also add the verb tense for every verb I found. This improves test set accuracy by almost a percent! In class we mentioned adding features of the form `this *`, where `*` is some other word; I added these features, and noticed a slight improvement in test accuracy, but I haven't seen a `this *` feature heavily weighted.

I scanned through `train.csv`, looking for anything that might pop out in the sentence column. What popped out, however, is that I could be using the page and trope columns, as those are also included in the given Kaggle test data in `test.csv`! This made a *huge* difference in the test set accuracy, but at the cost of increased overfitting (training error was 90%!).

```
(1,2,3,4)-grams, Tf-Idf weighting,\
tense, page, trope:
conf mat on test set:
Accuracy: 0.750507
False True
488   246
123   622
```

```
(1,2,3)-grams, Tf-Idf weighting, IG scoring\
tense, page, trope, this *, genre
Accuracy: 0.751183
False True
481   253
115   630
```

While perusing the intrawebs, I found a blog post discussing using χ^2 statistics to score features and reduce the amount of "noisy", low-weight features that might be corrupting my accuracy. It turns out that sklear has this functionality builtin! I later found a wonderful paper by JBG et al. describing their approach to a *very similar problem*; they used an information theoretic measure (information gain [IG]), which I found to give better performance (about 75% on test set).

Finally, I added show genre pulled via a Python API and added that to my feature list. I didn't see a noticeable improvement in test accuracy ("within the noise"), but I think this helps with generalization to new shows. The genre doesn't show up in my list if highly weighted features.

Longer than one page (sorry!), but I would like cite the following.

- sklearn: `http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html`
- NLTK: `http://www.nltk.org/`
- χ^2 feature score: `http://streamhacker.com/2010/06/16/text-classification-sentiment-analysis-eliminate-low-informa`
- JBG et al. paper: `http://www.umiacs.umd.edu/~jbg/docs/2013_spoiler.pdf`
- The TV DB: `http://thetvdb.com/` and `https://github.com/fuzzycode/pytvdbapi`