AM 221 Milestone 3

Arjun Mody & Michael Zochowski

April 2, 2014

For this milestone, we focused on three aspects of our project: feature selection for the SVMs, building a one-vs-all SVM classifier, and establishing a benchmark.

1 Feature Selection

There are many naive approaches to feature selection which include taking simply the most frequent words or those that do not occur in the intersection of the two categories. We use an approach described by Olena Kummer and Jacques Savoy (2012) in which they use the normalized Z-score of different features within a specific category to determine how "confident" they are at predicting the sentiment of a sentence. We adapt a basic version of their method, which is used for binary classification, to our multi-classification problem.

Let C be the entire corpus of reviews that we are examining for feature selection. Let S be the subset of reviews with a particular label, such as 5-star reviews. Then $\neg S$ is the rest of the reviews, and $C = S \cup \neg S$. Let f be some feature of interest, and $\neg f$ contains all other features. Consider the following table:

	S	$\neg S$	C
f	a	b	$a+b$
$\neg f$	c	d	$c+d$
	$a+c$	$b+d$	n

Here, a is the number of times the feature appears in the subset. b is the number of times the feature appears in the rest of the reviews, and $a+b$ is the number of times it appears in the entire corpus. Thus, $P(f) = (a+b)/n$ is the frequency that f appears at all. Let n' be the number of words in the subset, or $n' = a+c$. Then, we can treat the distribution of f as a binomial distribution where $P(f)$ is the probability of drawing it and n' is the number of trials. Thus, $P(f) \cdot n'$ gives us the expected number of times that f should appear in the subset when using the frequency over the entire corpus. If the value $a - P(f) \cdot n'$, or the difference between the actual number of times that f is observed in S and the expected number of times, is large, then we can be confident that the distribution of f

in S differs from its distribution in C. We use the following normalized Z-score for a feature:

$$\text{Z-score}(f) = \frac{a - n' \cdot P(f)}{\sqrt{n' \cdot P(f) \cdot (1 - P(f)}}$$

2 One-vs-all SVM

The classifier works in the following way. By solving an LP to build each SVM, we are able to construct five SVMs, each of which represent the binary classification of a specific rating versus every othe rating. In other words, each SVM defines a hyperplane which tries to separate one class of ratings (i.e. one-star or two-star etc...) from all the other ratings. Then, to make a prediction, we generate a ”confidence score” given by each SVM, which is the distance (which can be negative) above the hyperplane that the review lies. Then, we assign the review the rating of the SVM that garners the largest confidence score.

3 Benchmark

We will use the following two benchmarks in our final project in assessing the accuracy of our classifier:

(i) Percentage Correct

$$\frac{\text{Num. Correct}}{\text{Num. Predicted}}$$

(ii) Average Square Error: Let Y_i be the true value of review i, and let X_i be the predicted value. Let there be N predictions. Then we compute

$$\frac{1}{N}\sum_{i=1}^{N}(X_i - Y_i)^2$$

When we ran the classifier on 1,000 reviews, it predicted the ratings of 44.5% of the reviews correctly, with an average square error of 2.863, meaning it was off by an average of 1-2 stars. Of course, for a binary classifier this is quite poor, but for a multi-class classifier, this may not be so bad, depending on how separable the data truly are. We can see that there is a lot of room for improvement, however. Most of this will happen in selecting our features. We may decide to exclude more than the 100 most common English language words, and we will also expand our feature set to include 2-grams and 3-grams, or phrases that appear often in specific types of reviews. Finally, we will do an analysis of how well the Z-score is in generating reasonable feature lists of subjective, rather than objective, terms. There are also more sophisticated methods for suplementing the Z-score in feature selection which we might explore from the original paper.

For the final project, we hope to do more rigorous testing in terms of separating our training and test sets, as well as using methods from machine learning such as k-fold cross-validation.

4 The Code

The relevant code is in the following files:

- `constants.py`: This file contains relevant constants to run our various scripts (such as the locations of various files, the number of common English language words to remove from the feature set, and the number of reviews in the learning set).
- `svm.py`: This file contains our main script to build the SVMs and make predictions. The file has several functions, and can be run with various options. The function `load_text()` loads in the reviews from the raw data file, and structures them. The function `build_SVMs` extracts the features, writes several dictionaries to file which map the features to a vector, writes an AMPL `.dat` file, and runs the AMPL solver to build the actual SVM. The function `predict()` makes predictions on N random reviews, and computes the percentage of reviews that were correct as well as the average square error.

  Usage: `~$ .\svm.py [-h] [-s] [-d] [-p] <numpredictions> [-v]`:
  - -h (help): Output usage instructions
  - -s (svm): Build the SVMs (takes some time)
  - -d (dictionaries): Re-write the dictionaries
  - -p numpredictions (predict): Make an integer number of predictions.
  - -v (verbose): Output information about each individual prediction, including the predicted rating, the actual rating, and the review text.
- AMPL Data files: Contains the feature vectors for building the SVMs.
- AMPL Model file: Contains the actual LP model and optimization problem that we are solving to build each individual SVM.
- Dictionaries: These map the features to indices in order to convert reviews into feature vectors.