Using crowdsourcing to get representations based on regular expressions

Anders Søgaard and Hector Martinez and Jakob Elming and Anders Johannsen
Center for Language Technology
University of Copenhagen
DK-2300 Copenhagen S
{soegaard|alonso|zmk867|ajohannsen}@hum.ku.dk

Abstract

Often the bottleneck in document classification is finding good representations that zoom in on the most important aspects of the documents. Most research uses n-gram representations, but relevant features often occur discontinuously, e.g., *not... good* in sentiment analysis. In this paper we present experiments getting experts to provide regular expressions, as well as crowdsourced annotation tasks from which regular expressions can be derived. Somewhat surprisingly, it turns out that these crowdsourced feature combinations outperform automatic feature combination methods, as well as expert features, by a very large margin and reduce error by 24-41% over n-gram representations.

1 Introduction

Finding good representations of classification problems is often glossed over in the literature. Several authors have emphasized the need to pay more attention to finding such representations (Wagstaff, 2012; Domingos, 2012), but in document classification most research still uses n-gram representations.

This paper considers two document classification problems where such representations seem inadequate. The problems are answer scoring (Burstein et al., 1998), on data from stackoverflow.com, and multi-attribute sentiment analysis (McAuley et al., 2012). We argue that in order to adequately represent such problems we need *discontinuous* features, i.e., regular expressions.

The problem with using regular expressions as features is of course that even with a finite vocabulary we can generate infinitely many regular expressions that match our documents. We suggest to use expert knowledge or crowdsourcing in the loop. In particular we present experiments where standard representations are augmented with features from a few hours of manual work, by machine learning experts or by turkers.

Somewhat surprisingly, we find that features derived from crowdsourced annotation tasks lead to the best results across the three datasets. While crowdsourcing of annotation tasks has become increasing popular in NLP, this is, to the best of our knowledge, the first attempt to crowdsource the problem of finding good representations.

1.1 Related work

Musat et al. (2012) design a collaborative two-player game for sentiment annotation and collecting a sentiment lexicon. One player guesses the sentiment of a text and picks a word from it that is representative of its sentiment. The other player also provides a guess observing only this word. If the two guesses agree, both players get a point. The idea of gamifying the problem of finding good representations goes beyond crowdsourcing, but is not considered here. Boyd-Graber et al. (2012) crowdsource the feature weighting problem, but using standard representations. The work most similar to ours is probably Tamuz et al. (2011), who learn a 'crowd kernel' by asking annotators to rate examples by similarity, providing an embedding that promotes feature combinations deemed relative when measuring similarity.

			BoW		*Exp*		*AMT*	
	n	$P(1)$	m	μ_x	m	μ_x	m	μ_x
STACKOVERFLOW	97,519	0.5013	30,716	0.00131	1,156	0.1380	172,691	0.00331
TASTE	152,390	0.5003	38,227	0.00095	666	0.10631	114,588	0.00285
APPEARANCE	152,331	0.5009	37,901	0.00097	650	0.14629	102,734	0.00289

Table 1: Characteristics of the $n \times m$ data sets

2 Experiments

Data The three datasets used in our experiments come from two sources, namely stackoverflow.com and ratebeer.com. The two beer review datasets (TASTE and APPEARANCE) are described in McAuley et al. (2012) and available for download.[1] Each input example is an unstructured review text, and the associated label is the score assigned to taste or appearance by the reviewer. We randomly sample about 152k data points, as well as 500 examples for experiments with experts and turks.

We extracted the STACKOVERFLOW dataset from a publicly available data dump,[2], and we briefly describe our sampling process here. We select pairs of answers, where one is ranked higher than the other by stackoverflow.com users. Obviously the answers submitted first have a better chance of being ranked highly, so we also require that the highest ranked answer was submitted last. From this set of answer pairs, we randomly sample 97,519 pairs, as well as 500 examples for our experiments with experts and turks.

Our experiments are classification experiments using the same learning algorithm in all experiments, namely L_1-regularized logistic regression. We don't set any parameters The only differences between our systems are in the feature sets. Results are from 5-fold cross-validation. The four feature sets are described below: *BoW*, *HI*, *Exp* and *AMT*.

For motivating using regular expressions, consider the following sentence from a review of John Harvard's Grand Cru:

(1) Could have been more flavorful.

The only word carrying direct sentiment in this sentence is *flavorful*, which is positive, but the sentence is a negative evaluation of the Grand Cru's taste. The trigram *been more flavorful* seems negative at first, but in the context of negation or in a comparative, it can become positive again. However, note that this trigram may occur discontinuously, e.g., in *been less watery and more flavorful*. In order to match such occurrences, we need simple regular expressions, e.g.,:

```
been.*more.*flavorful
```

This is exactly the kind of regular expressions we asked experts to submit, and that we derived from the crowdsourced annotation tasks. Note that the sentence says nothing about the beer's appearance, so this feature is only relevant in TASTE, not in APPEARANCE.

BoW **and** *BoW+HI* Our most simple baseline approach is a bag-of-words model of unigram features (*BoW*). We lower-case our data, but leave in stop words. We also introduce a semantically enriched unigram model *(BoW)+HI*, where in addition to representing what words occur in a text, we also represent what Harvard Inquirer (HI)[3] word classes occur in it. The HI classes are used to generate features from the crowdsourced annotation tasks, so the semantically enriched unigram model is an important baseline in our experiments below.

BoW+Exp In order to collect regular expressions from experts, we set up a web interface for querying held-out portions of the datasets with regular expressions that reports how occurrences of the submitted regular expressions correlate with class. We used the Python `re` syntax for regular expressions after augmenting word forms with POS and semantic classes from the HI. Few of the experts made use of the POS tags, but many regular expressions included references to HI classes.

[1]http://snap.stanford.edu/data/web-RateBeer.html

[2]http://www.clearbits.net/torrents/2076-aug-2012

[3]http://www.wjh.harvard.edu/ inquirer/homecat.htm

Regular expressions submitted by participants were visible to other participants during the experiment, and participants were allowed to work together. Participants had 15 minutes to familiarize themselves with the syntax used in the experiments. Each query was executed in 2-30 seconds.

Seven researchers and graduate students spent five effective hours querying the datasets with regular expressions. In particular, they spent three hours on the Stack Exchange dataset, and one hour on each of the two RateBeer datasets. One had to leave an hour early. So, in total, we spent 20 person hours on Stack Exchange, and seven person hours on each of the RateBeer datasets. In the five hours, we collected 1,156 regular expressions for the STACKOVERFLOW dataset, and about 650 regular expressions for each of the two RateBeer datasets. *Exp* refers to these sets of regular expressions. In our experiments below we concatenate these with the *BoW* features to form *BoW+Exp*.

BoW+AMT For each dataset, we also had 500 held-out examples annotated by three turkers each, using Amazon Mechanical Turk,[4] obtaining 1,500 HITs for each dataset. The annotators were presented with each text, a review or an answer, twice: once as running text, once word-by-word with bullets to tick off words. The annotators were instructed to tick off words or phrases that they found predictive of the text's sentiment or answer quality. They were not informed about the class of the text. We chose this annotation task, because it is relatively easy for annotators to mark spans of text with a particular attribute. This set-up has been used in other applications, including NER (Finin et al., 2010) and error detection (Dahlmeier et al., 2013). The annotators were constrained to tick off at least three words, including one closed class item (closed class items were colored differently). Finally, we only used annotators with a track record of providing high-quality annotations in previous tasks. It was clear from the average time spent by annotators that annotating STACKOVERFLOW was harder than annotating the Ratebeer datasets. The average time spent on a Ratebeer HIT was 44s, while for STACKOVERFLOW it was 3m:8s. The mean number of words ticked off was between 5.6 and 7, with more words ticked off in STACKOVERFLOW. The maximum number of words ticked off by an annotator was 41. We spent $292.5 on the annotations, including a trial round. This was supposed to match, roughly, the cost of the experts consulted for *BoW+Exp*.

	BoW	*HI*	*Exp*	*AMT*
STACKOVERF	0.655	0.654	0.683	**0.739**
TASTE	0.798	0.797	0.798	**0.867**
APPEARANCE	0.758	0.760	0.761	**0.859**

Table 2: Results using all features

The features generated from the annotations were constructed as follows: We use a sliding window of size 3 to extract trigrams over the possibly discontinuous words ticked off by the annotators. These trigrams were converted into regular expressions by placing Kleene stars between the words. This gives us a manually selected subset of skip trigrams. For each skip trigram, we add copies with one or more words replaced by one of their HI classes.

Feature combinations This subsection introduces some harder baselines for our experiments, considered in Experiment #2. The simplest possible way of combining unigram features is by considering n-gram models. An n-gram extracts features from a sliding window (of size n) over the text. We call this model *BoW*$(N = n)$. Our *BoW*$(N = 1)$ model takes word forms as features, and there are obviously more advanced ways of automatically combining such features.

Kernel representations We experimented with applying an approximate feature map for the additive χ^2-kernel. We used two sample steps, resulting in $4N + 1$ features. See Vedaldi and Zimmerman (2011) for details.

Deep features We also ran denoising autoencoders (Pascal et al., 2008), previously applied to a wide range of NLP tasks (Ranganath et al., 2009; Socher et al., 2011; Chen et al., 2012), with $2N$ nodes in the middle layer to obtain a deep representation of our datasets from χ^2-*BoW* input. The network was trained for 15 epochs. We set the drop-out rate to 0.0 and 0.3.

Summary of feature sets The feature sets – *BoW*,

[4]www.mturk.com

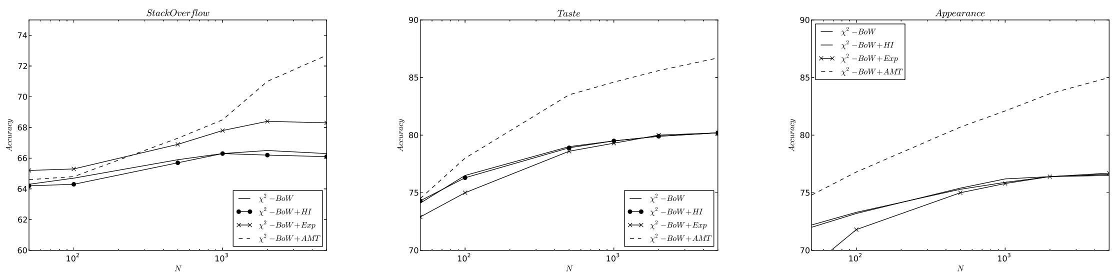

Figure 1: Results selecting N features using χ^2 (left to right): STACKOVERFLOW, TASTE, and APPEARANCE. The x-axis is logarithmic scale.

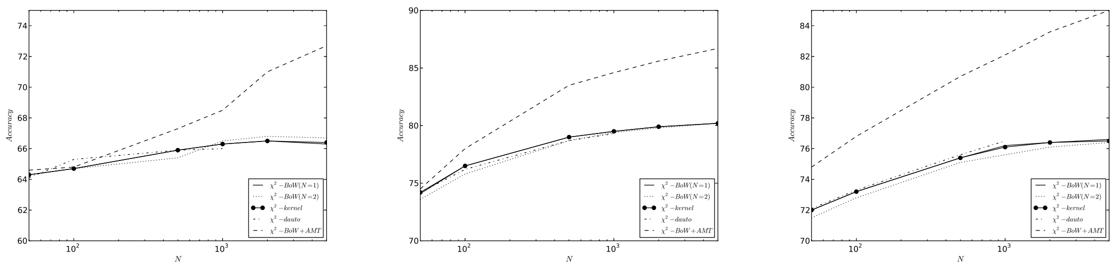

Figure 2: Results using different feature combination techniques (left to right): STACKOVERFLOW, TASTE, and APPEARANCE. The x-axis is logarithmic scale.

Exp and *AMT* – are very different. Their characteristics are presented in Table 1. $P(1)$ is the class distribution, e.g., the prior probability of positive class. n is the number of data points, m the number of features. Finally, μ_x is the average density of data points. One observation is of course that the expert feature set *Exp* is much smaller than *BoW* and *AMT*, but note also that the expert features fire about 150 times more often on average than the BoW features. *HI* is only a small set of additional features.

3 Results

Experiment #1: ***BoW*** **vs.** ***Exp*** **and** *AMT* We present results using all features, as well as results obtained after selecting k features as ranked by a simple χ^2 test. The results using all collected features are presented in Table 2. The error reduction on STACKOVERFLOW when adding crowdsourced features to our baseline model (*BoW+AMT*), is 24.3%. On TASTE, it is 34.2%. On APPEARANCE, it is 41.0%.

The *BoW+AMT* feature set is bigger than those of the other models. We therefore report results using the top-k features as ranked by a simple χ^2 test. The result curves are presented in the three plots in Fig. 1. With +500 features, *BoW+AMT* outperforms the other models by a large margin.

Experiment #2: ***AMT*** **vs. more baselines** The *BoW* baseline uses a standard representation that, while widely used, is usually thought of as a weak baseline. *BoW+HIT* did not provide a stronger baseline. We also show that bigram features, kernel-based decomposition and deep features do not provide much stronger baselines either. The result curves are presented in the three plots in Fig. 2. *BoW+AMT* is still significantly better than all other models with +500 features. Since autoencoders are consistently worse than denoising autoencoders (drop-out 0.3), we only plot denoising autoencoders.

4 Conclusion

We presented a new method for deriving feature representations from crowdsourced annotation tasks and showed how it leads to 24%-41% error reductions on answer scoring and multi-aspect sentiment analysis problems. We saw no significant improvements using features contributed by experts, kernel representations or learned deep representations.

References

Jordan Boyd-Graber, Brianna Satinoff, He He, and Hal Daume. 2012. Besting the quiz master: Crowdsourcing incremental classification games. In *NAACL*.

Jill Burstein, Karen Kukich, Susanne Wolff, Chi Lu, Martin Chodorow, Lisa Braden-Harder, and Mary Dee Harris. 1998. Automated scoring using a hybrid feature identification technique. In *ACL*.

Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. 2012. Marginalized denoising autoencoders for domain adaptation. In *ICML*.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner English. In *Workshop on Innovative Use of NLP for Building Educational Applications, NAACL*.

Pedro Domingos. 2012. A few useful things to know about machine learning. In *CACM*.

Tim Finin, Will Murnane, Anand Karandikar, Nicholas Keller, Justin Martineau, and Mark Dredze. 2010. Annotating named entities in Twitter data with crowdsourcing. In *NAACL Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*.

Julian McAuley, Jure Leskovec, and Dan Jurafsky. 2012. Learning attitudes and attributes from multi-aspect reviews. In *ICDM*.

Claudiu-Christian Musat, Alireza Ghasemi, and Boi Faltings. 2012. Sentiment analysis using a novel human computation game. In *Workshop on the People's Web Meets NLP, ACL*.

Vincent Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML*.

Rajesh Ranganath, Dan Jurafsky, and Dan McFarland. 2009. It's not you, it's me: detecting flirting and its misperception in speed-dates. In *NAACL*.

Richard Socher, Eric Huan, Jeffrey Pennington, Andrew Ng, and Christopher Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*.

Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. 2011. Adaptively learning the crowd kernel. In *ICML*.

Andrea Vedaldi and Andrew Zisserman. 2011. Efficient additive kernels via explicit feature maps. In *CVPR*.

Kiri Wagstaff. 2012. Machine learning that matters. In *ICML*.