A general-purpose sentence-level nonsense detector

CS229 Final Project Report

December 12, 2014

Ian Tenney
iftenney@stanford.edu

1 Introduction and Background

I have constructed a sentence-level nonsense detector, with the goal of discriminating well-formed English sentences from the large volume of fragments, headlines, incoherent drivel, and meaningless snippets present in internet text. For many NLP tasks, the availability of large volumes of internet text is enormously helpful in combating the sparsity problem inherent in modeling language. However, the derived models can be easily polluted by ungrammatical text and spam, and automated means are necessary to filter this and provide high-quality training data.

There is scarce precedent in the literature for a direct nonsense-detection system, but similar problems exist in the context of spam filtering and computational sentence completion. For spam filtering, recent research has focused on the problem of "Bayesian poisoning" (Hayes, 2007), whereby Naive Bayes-based filters are defeated by the inclusion of random words or text snippets that make the spam vocabulary appear similar to normal text. Solutions to this problem propose combining multiple sources of information, such as higher n-grams, in order to introduce greater context (Upasana and Chakravarty, 2010).

Sentence completion is an example of a standard NLP task where the system must consider a variety of possible solutions and discriminate between "sensical" and "nonsensical" answers. While generally operating on a more restricted domain (e.g. SAT questions), the fundamental goal is similar enough that similar techniques can be applied. The Microsoft Research Sentence Completion Challenge (Zweig and Burges, 2011) highlights several approaches to this task, including neural networks and dependency parsing but also showing strong performance with lighter-weight n-gram and distributional models (Zweig et al., 2012).

I take a lightweight approach, using a mix of heuristics, lightweight token-based features, part-of-speech features, and language model scores as features to avoid computationally-intensive parsing or neural networks. I structure my system as a binary classification task on a heterogeneous feature space, consisting of to produce a final answer of "sentence" or "nonsense" for a given line of text.

I implement this project in a mix of Java and Python, using Java to interface with Stanford CoreNLP for feature extraction, and Python (with the excellent pandas and scikit-learn libraries) for data management, classifier implementation, and analysis.

2 Data

2.1 Source and Labeling Schema

The base dataset is a 1 TB corpus of sentence-tokenized internet text derived from Common Crawl (http://commoncrawl.org/) and provided by the Stanford NLP group. This is a relatively indiscriminate crawl, and contains text from a variety of web pages, ranging from internet forums to pornography (it's the internet) to shopping sites and news articles. In addition to coherent sentences, this contains a large number of link titles, headlines, and other text fragments. A typical few lines would be:

- `p 49, panel 2-4 (Makede & Ntshonge)`
- `If so, an Authenticator may have been maliciously associated with your account by an unauthorized party.`
- `No New Messages`
- `Forum Software © ASPPlayground.NET Advanced Edition 2.4.5`
- `Living Area`

I annotate each line with one of the following labels:

- `-SENTENCE-` complete English sentences, not necessarily with perfect grammar
- `-FRAGMENT-` sentence fragments, headlines, or captions, constituting coherent but non-sentential text
- `-NONSENSE-` small fragments, noun phrases, gibberish, spam, and anything else not counted above
- `-NONE-` no label, or foreign-language text

This scheme is designed to roughly align with downstream NLP tasks: nonsense is unwanted for any task, while fragments are inappropriate for parsing but, as they are subsequences of real sentence, are still useful to build language models. I retain these labels for visualization purposes and future use, but for our supervised learning models we treat “-SENTENCE-” as a positive example and all other labels as negative.

2.2 Datasets

I compile two labeled datasets from this corpus, one by manual annotation and one by crowdsourcing. The crowdsourced dataset consists of 12,000 lines of text, each labeled by two distinct workers on Amazon Mechanical Turk. Of these, 7099 lines are labeled unambiguously as a single category by both annotators[1], and thus can be used for training and evaluation. This means that we drop the most ambiguous examples, since Turkers are more likely to disagree on lines that are difficult to classify by any algorithm; as we see in Section 6, this tends to give results that are inflated relative to real-world performance.

To remedy this, I also compiled a manual dataset, consisting of 5,000 lines of text, each labeled by a single highly-trained user[2]. All of these labels are assumed to be unambiguous, and the entire set is usable.

The class distribution for each dataset is:

Dataset (size)	**Manual (4997)**	**MTurk (7099)**
-SENTENCE-	17.47%	35.79%
-FRAGMENT-	31.50%	26.02%
-NONSENSE-	50.29%	38.16%
-NONE-	0.74%	0.03%

where the discrepancy can be attributed difficulty by Mechanical Turk workers in correctly identifying fragments and nonsense, leaving relatively more sentenced behind when ambiguous data is discarded.

3 Features and Preprocessing

I implement five groups of features, for a maximum of a 148-dimensional feature space:

0. Baseline sentence heuristic: first letter is Capitalized, and line ends with one of `. ? !` (1 feature).
1. Number of characters, words, punctuation, digits, and named entities (from Stanford CoreNLP NER tagger), and normalized versions by text length (10 features).
2. Part-of-speech distributional tags: $\frac{\#\texttt{<tag>}}{\#\text{ words}}$ for each Penn treebank tag (45 features).
3. Indicators for the part of speech tag of the first and last token in the text (45x2 = 90 features).
4. Language model raw score ($s_{lm} = \log p(\text{text})$) and normalized score ($\bar{s}_{lm} = \frac{s_{lm}}{\#\text{ words}}$) (2 features).

[1]Unfortunately, budget and time constraints prevented us from using a third annotator, which would have increased the fraction of usable labels.

[2]myself, with some help from my roommate.

For part-of-speech tags, I use the default English tagger included with Stanford CoreNLP (Manning et al., 2014). This is a sequence-based MaxEnt tagger trained on a mixed corpus of newswire, technical, and colloquial text. For language model scores, I build a bigram model using the KenLM package, trained on the entire English Gigaword 5 corpus (Parker et al., 2011) of newswire text.

Before evaluating the model, all features are independently scaled to zero mean and unit variance (across the training set) in order to improve performance of regularized and non-linear models and allow extraction of meaningful feature weights from logistic regression.

4 t-SNE Visualization

I used t-distributed Stochastic Neighbor Embedding (t-SNE) to visualize our dataset and understanding the difficulty of the classification task. t-SNE attempts to non-linearly map high-dimensional data into a low-dimensional (usually 2D) space by preserving local similarity: formally, it generates a gaussian distribution ($p_{ij} = \frac{p_{j|i}+p_{i|j}}{2N}$ where $p_{j|i} \propto \exp\left(-\|x_i - x_j\|^2/2\sigma_i^2\right)$) over pairwise distances in the original space, then finds an embedding that (locally) minimizes the KL divergence between this and a heavy-tailed t-distributed similarity q_{ij} in the low-dimensional space (van der Maaten, 2014). I use the Barnes-Hut implementation of t-SNE, using the Python implementation available at http://lvdmaaten.github.io/tsne/, and retain our multi-class labels for visualization purposes.

I generate visualizations using the “level 3” feature set (everything but language models), with scaling normalization applied.

On the MTurk dataset, we find that restricting to unambiguous labels gives us a well-separated space: regions of mostly sentences (blue) have a well-defined boundary from nonsense regions (red), and to a lesser extend, fragments (green). On the manual dataset (Figure 4) this separation is much weaker, and sentences are interspersed with fragments and nonsense in many clusters, reflecting the inclusion of more ambiguous data points.

Because the t-SNE algorithm uses L2 distances in the feature space, it is sensitive to scaling: strong contributors to distance in the original space may be relatively uninformative for our classification task. To better visualize the difficulty of the classification prob-

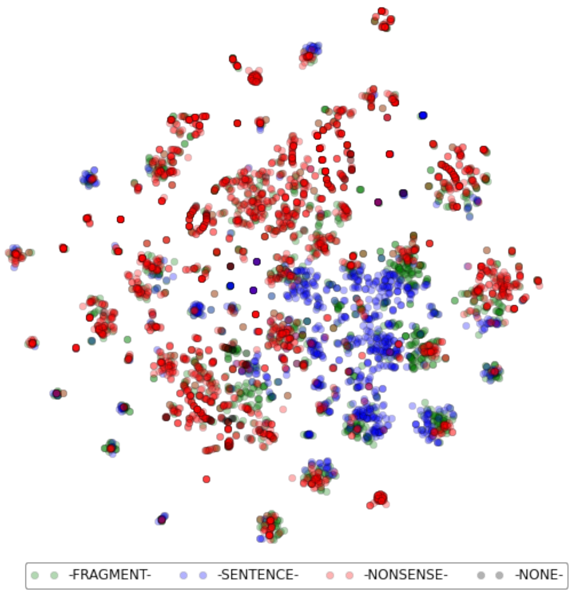

Figure 1: t-SNE embedding of manual dataset. MTurk dataset visualization available online.

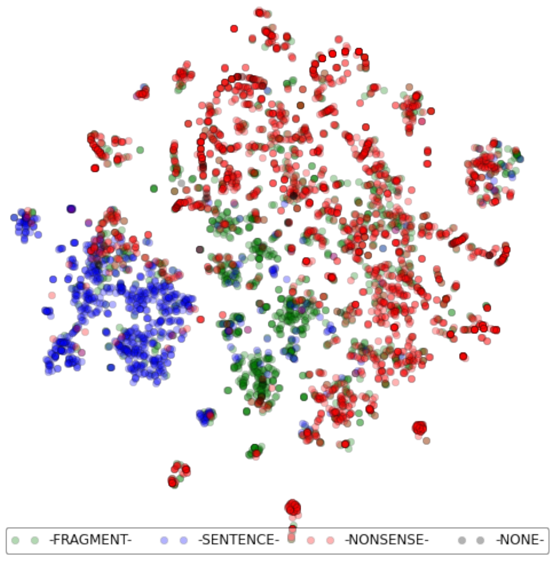

Figure 2: t-SNE embedding of manual dataset, scaled features. MTurk dataset visualization available online.

lem, we train a logistic classifier (L1, C = 100) on the full dataset and use the resulting weight vector (97 nonzero elements, out of 146) to scale the feature space: $x_j^{(i)} \to w_j x_j^{(i)}$ before running t-SNE. This emphasizes the dimensions most relevant to our task, and shows a clean separating margin in both datasets (Figure 1).

I have posted interactive visualizations of the dataset at `http://stanford.edu/~iftenney/nonsense/`, in which you can mouse-over data points to see the original text and label. In the scaled space, the resulting clusters group text that share common attributes or grammatical patterns, such as beginning with a wh-word or consisting of two nouns.

5 Supervised Models

I implement five models for the binary classification task: a baseline heuristic, logistic regression, linear SVM, RBF (Gaussian) SVM, and random forest. The baseline system uses feature 0 (Section 3) directly: if a line of text matches the heuristic, it is classified as `"-SENTENCE-"`, otherwise negative. This achieves a precision of 79% and recall of 89% on the manual dataset, reflecting that a large number of nonsense fragments conform to this naive pattern.

All models are tuned using 5-fold cross validation on the training set, using a grid search to choose the best set of hyperparameters. For logistic regression and linear SVM, we tune the regularization strength C and use either an L1 or L2 regularization penalty. For the RBF SVM, we use a Gaussian kernel, and tune the regularization strength C and the kernel width γ. For the random forest, we use the scikit-learn implementation with probabilistic prediction (Pedregosa et al., 2011) and tune the number of decision trees and the maximum tree depth.

I compute accuracy, precision, and recall for all our models, and optimize for F1 score ($F_1 = 2 \cdot \frac{P \cdot R}{P+R}$). The F1 score provides a balance between precision and recall that is less sensitive than either to the choice of threshold, and so provides a good summary of overall model performance.

6 Results and Discussion

I present F1 scores for all our models in Tables 2, 1, 3, and 4. The MTurk dataset is split 5000/2099 train/test, while the manual dataset is split 4000/1000. Train and dev set results are averaged over 5-fold cross-validation, while test set is from a single evaluation on unseen data.

All results are presented with the best hyperparameters from the cross-validation runs, which vary somewhat depending on the feature set. Most notably, the linear models (logistic and linear SVM) perform best with L2 regularization and C = 0.1 on features 0,1, while L1 regularization (with C = 1) performs better on the larger feature sets (0,1,2, and higher) that introduce more correlated or less-informative features. As mentioned in Section 4, L1 regularized logistic regression with C = 100 yields 97 (out of 146) nonzero features, with most of the zeros mapping to group 3 indicator features.

Table 1: F1 scores for each model, on manual dataset, all features.

	Train	Dev	Test
Number of elements	**3200**	**800**	**997**
Baseline	78.33%	*N/A*	83.73%
Logistic	83.84%	80.15%	84.46%
Linear SVM	83.81%	80.67%	84.02%
RBF SVM	89.42%	81.41%	84.40%
Random Forest	99.93%	86.11%	**89.04%**

Table 2: F1 scores on MTurk dataset, all features.

	Train	Dev	Test
Number of elements	**4000**	**1000**	**2099**
Baseline	90.07%	*N/A*	91.12%
Logistic	93.40%	92.11%	92.15%
Linear SVM	93.31%	92.43%	92.56%
RBF SVM	95.83%	92.63%	92.60%
Random Forest	99.98%	93.33%	94.07%

The more expressive non-linear models also have the best performance, although RBF SVM only slightly outperforms the logistic and linear SVM classifiers at the expense of much longer training time. Random Forest yields the strongest performance overall: despite nearly memorizing the training set (F1 scores of $>$ 99.8% on all tests) it still performs significantly better (2-5%) than other models on unseen data. Notably, the model also trains quickly (within a factor of 5 of logistic regression), and is relatively insensitive to hyperparameters: performance with 100 trees of depth 10 is within 1% of 400 trees of depth 40, allowing for faster training and a more compact model.

Table 3: Test-set F1 scores across all feature sets, on manual dataset. Feature groups as described in Section 3.

Feature Grp.	**0,1**	**0,1,2**	**0,1,2,3**	**all**
Baseline	83.73%	83.73%	83.73%	83.73%
Logistic	83.04%	83.63%	84.71%	84.46%
Linear SVM	82.94%	83.38%	84.27%	84.02%
RBF SVM	83.33%	83.64%	85.19%	84.40%
Random Forest	87.21%	88.82%	**89.40%**	**89.04%**

Table 4: Test-set F1 scores, on MTurk dataset.

Feature Grp.	**0,1**	**0,1,2**	**0,1,2,3**	**all**
Baseline	83.73%	83.73%	83.73%	83.73%
Logistic	91.77%	92.06%	92.02%	92.15%
Linear SVM	91.17%	92.18%	92.32%	92.56%
RBF SVM	92.81%	92.68%	92.59%	92.60%
Random Forest	93.65%	94.05%	93.82%	**94.07%**

Performance increases with additional features, as shown in Table 3. Group 1 features outperform the baseline (under the random forest model), and all models improve slightly when adding group 2 (part-of-speech distribution) and group 3 (begin- and end-tag) features. Interestingly, adding the language model does not seem to help: performance is stagnant, or slightly lower when these features are included. This is likely due to the low-order (bigram) of the model, which makes it unable to capture sophisticated relationships across strings of words, and due to the many high-probability sequences found in nonsense and fragments.

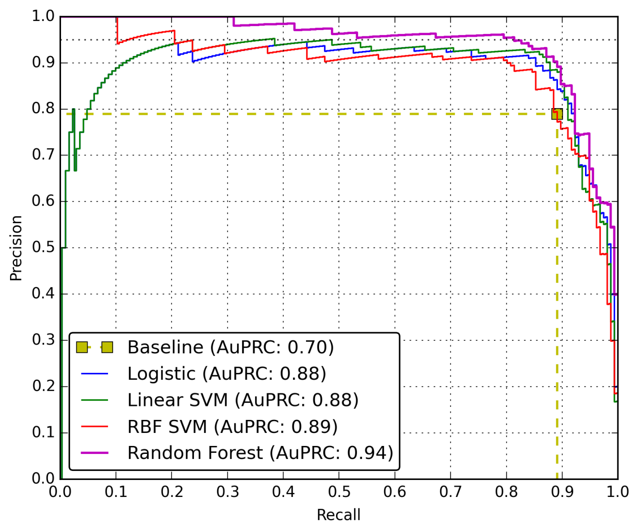

Figure 3: Precision-recall curves for all models, on the manually-annotated dataset using features 0,1,2,3. Yellow dot shows performance of baseline heuristic. AuPRC is the area under the precision recall curve, as an aggregate metric of performance.

As expected from the labeling ambiguity, performance of both the baseline heuristic and our models is significantly higher on the MTurk dataset than on the manually-annotated set. Nonetheless, with all "marginal" examples labeled, we are still able to achieve nearly 90% F1 using the random forest classifier. Analysis of the precision-recall curve in Figure 6 shows that the random forest clearly outperforms other models in precision, reaching a precision of 96% at 80% recall - performance which should be sufficient for many practical applications.

6.1 Feature Importance

I can extract feature importances from the random forest by counting splits: for a given decision tree, a feature's importance is given by the number of training examples that are decided using that feature. Averaging this across the ensemble gives us the following as our top 10 features:

1. `f_sentence_pattern` - baseline heuristic
2. `f_pos_.` - fraction of "." in POS tags
3. `f_nchars` - number of characters
4. `f_nwords` - number of words
5. `f_rupper` - ratio of # uppercase / # words

6. `f_rpunct` - ratio of # punctuation / # letters
7. `f_pos_IN` - fraction of certain prepositions
8. `f_pos_DT` - fraction of determiners (“the”, “those”, “an”, etc.)
9. `f_pos_PRP` - fraction of personal pronouns
10. `f_nupper` - number of uppercase letters

The importance of the sentence heuristic is expected, given the strong performance of the baseline, and the importance of many of the group 1 features is also not surprising, given that group 1 is a dense set with all features active for a given line of text.

Unfortunately, unlike with linear models, it is not possible to tell directly whether each of these features is indicative of positive or negative examples (indeed, this is not necessarily well-defined).

6.2 Error Analysis

While the learned model significantly outperforms baseline, it is still weak in many cases where the sentence heuristic is incorrect. This can go both ways: many sentences from internet forums are not properly capitalized (such as `"co-produced by Danger Mouse ... that's cool!"`), while in other cases the heuristic applies but is outweighed by other features, such as `"Label is resistant to these solvents: Xylene, IPA, DMSO, Ethanol, Toluene."` which is treated as `-OTHER-` presumably due to the large number of capital letters and abnormally-high noun fraction. In other cases, we see the limitations of the distributional model, such as `"Microsoft's network certifications for another article."`, which is erroneously called a sentence as its lack of a verb is overlooked.

Additionally, a large number of errors on the MTurk dataset are related to labeling: in many cases, the model actually predicts correctly while the labels are wrong. This highlights the difficulty of obtaining good crowdsourced labels: there is a trade-off between precision and time, and providing longer instructions and other quality-control measures can quickly ramp up the cost of an annotation project.

7 Conclusion and Future Work

I present a model that successfully discriminates complete sentences from fragments and nonsense, and is capable of reaching 96% precision while maintaining 80% recall. This is realized by a random forest classifier, which strongly overfits the training set with only a modest model size, yet still generalizes well to unseen data.

This model performs well enough for many practical purposes, where 4-5% noise is tolerable or can be filtered by more intensive means, but could certainly be improved. Most notably, additional data would permit the use of a richer feature set, such as an n-gram model over part-of-speech sequences. Alternatively, language modeling may be effective if trigrams or higher are used to span over more syntactic constructs; however, this also risks biasing the model, as it may learn to pick out text that resembles the training corpus, regardless of grammaticality.

8 Collaboration Statement

This project was based on an idea proposed by Gabor Angeli (angeli@stanford.edu) for CS224N, although I am submitting this project to CS229 only. Gabor provided me with access to the dataset, a budget for crowdsourcing, and some advice on feature engineering. All work for the project, including data annotation, pipeline implementation, and analysis, was performed by me alone.

References

Brian Hayes. 2007. Computing science: How many ways can you spell vagra? *American Scientist*, 95(4):298–302, July.

Christopher D. Manning, Mihai Surdeanu, John Bauer, et al. 2014. The stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition, June.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, October.

Upasana and S. Chakravarty. 2010. A survey on text classification techniques for e-mail filtering. In *2010 Second International Conference on Machine Learning and Computing (ICMLC)*, pages 32–36, February.

Laurens van der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245.

Geoffrey Zweig and Christopher J. C. Burges. 2011. The microsoft research sentence completion challenge. Technical Report MSR-TR-2011-129, December.

Geoffrey Zweig, John C. Platt, Christopher Meek, et al. 2012. Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 601–610, Stroudsburg, PA, USA. Association for Computational Linguistics.