Comparing Extant Story Classifiers: Results & New Directions*

Joshua D. Eisenberg[1], W. Victor H. Yarlott[2], and Mark A. Finlayson[3]

1 **Florida International University, School of Computer and Information Sciences, Miami, FL, USA**
jeise003@fiu.edu
2 **Florida International University, School of Computer and Information Sciences, Miami, FL, USA**
wvyar@cs.fiu.edu
3 **Florida International University, School of Computer and Information Sciences, Miami, FL, USA**
markaf@fiu.edu

Abstract

Having access to a large set of stories is a necessary first step for robust and wide-ranging computational narrative modeling; happily, language data–including stories–are increasingly available in electronic form. Unhappily, the process of automatically separating stories from other forms of written discourse is not straightforward, and has resulted in a data collection bottleneck. Therefore researchers have sought to develop reliable, robust automatic algorithms for identifying story text mixed with other non-story text. In this paper we report on the reimplementation and experimental comparison of the two approaches to this task: Gordon's unigram classifier, and Corman's semantic triplet classifier. We cross-analyze their performance on both Gordon's and Corman's corpora, and discuss similarities, differences, and gaps in the performance of these classifiers, and point the way forward to improving their approaches.

1998 ACM Subject Classification I.2.7 Natural Language Processing: Language Models

Keywords and phrases Story Detection, Machine Learning, Natural Language Processing, Perceptron Learning

Digital Object Identifier 10.4230/OASIcs.CMN.2016.6

1 Motivation

Computational narrative researchers often rely on textual story data to inform and support their analyses. There is a wealth of textual story content that can be harvested from electronic sources, including online newspapers, journals, blogs, e-books, emails, electronic records, and so forth. Nevertheless, what text does or does not constitute a story is not *prima facie* obvious: text is rarely labeled explicitly as story. Thus, to take advantage of the vast potential sources of narrative data available in electronic form, we need to be able to automatically find stories in a sea of otherwise non-story text.

* This work was partially supported by National Institutes of Health (NIH) grant number 5R01GM105033-02. Implementation work on this project was supported by Nicholas Zessoules, who was funded under Research Experience for Undergraduates (REU) grant number 1263124 from the National Science Foundation (NSF) and Department of Defense (DoD).

There has been prior work on automatic story identification, in particular, Gordon's unigram-based story classifier [9] and Corman's semantic triplet story classifier [2, 3] each tested on their own data sets. In this work we re-implement and cross-evaluate both these classifiers on both datasets, and thereby gain insight into the strengths and weaknesses of both approaches. This comparison also points the way forward to more accurate and effective story classifiers.

1.1 What is a Story?

In this study we are trying to classify the linguistic artifact known as the story, but what is a story? Let's start with the definitions of story that Gordon and Corman employ. Gordon is searching for *personal stories*: "textual discourse that describes a specific series of causally related events in the past, spanning a period of time of minutes, hours, or days, where the author or a close associate is among the participants" [9]. This definition implies knowledge of who the narrator is in relation to the *dramatis personae*, and the diegesis or narrative distance of the story teller. Corman says a story has three components: actors, the actions they perform, and the resolution which they "must result in resolution" [2]. But, not all stories have resolution. We propose a more succinct definition of story, a definition from author E. M. Forster, "A story is a narrative of events arranged in their time sequence" [8]. Typically these events are related for the story to be intelligble, but not all stories are composed this way. We feel that this definition is a beacon towards constructing a more accurate story classifier.

1.2 Outline of the Paper

To evaluate the classifiers on both sets of data, we needed access to implementations of both classifiers as well as to their training and testing corpora. Gordon's paper [9] provided a high level description of his classifier, which we used as a blueprint for implementing it in Java. He also provided a Python implementation of a story classifier[1], but it is not the same algorithm described in the paper. Our re-implementation of the Gordon classifier is close to the original design, which is discussed in §2.1. During reimplementation we noted that replacing the original classifier with a Perceptron implementation improved the results somewhat, and the details of this mildly improved version of Gordon's classifier are given in §2.2.

Corman et al. did not provide an open-source implementation of their code, and so we reimplemented that classifier from descriptions in their papers[2]. They provided a high level description of their preprocessing and feature extraction pipeline in [2, 3], described in §2.3. We used this as a guide for building our reimplementation, but there were aspects that performed significantly differently due to specific, difficult-to-reimplement design choices.

To train and test the classifiers we obtained the datasets used in the original papers: the ICSWM Spinn3r 2009 Weblog dataset [1], which Gordon used to test his classifier, and the CSC Islamic Extremist corpus, which Corman used for his study. We report the results of our reproductions of the original experiments in §3.1, and report on our new cross-test experiments in §3.2. We discuss our observations of the results of the experiments in §4, and outline some potential future work in §4.1. We conclude with a list of our contributions in §5.

---

[1] `https://github.com/asgordon/StoryNonstory`

[2] We intend to release all of our code as supplementary material associated with a future article.

2 Extant Story classifiers

The computational identification of stories is relatively a new problem, and there are only two extant approaches: one developed by Gordon et al. at USC [9], the other by Corman et al. at ASU [2, 3]. Both the Gordon and Corman classifiers leverage supervised machine learning algorithms trained on large annotated datasets, and both use linguistic features to separate story from non-story text. Gordon's classifier uses unigram frequencies to classify stories. This classifier was originally tested on the ICWSM 2009 Spinn3r blog dataset, which contains personal stories that were posted to blogs in 2009 [1]. Corman's classifier, on the other hand, focuses on verbs and their patient and agent arguments (semantic triplets). It also considers the unigram frequencies and density of various features such as part of speech tags, named entities, and stative verbs. Corman's classifier was originally tested on the CSC corpus of Islamic Extremist texts, in which each paragraph was annotated as either *story*, *exposition*, *supplication*, *religious verse*, or *other*.

2.1 Reimplemented Gordon Story Classifier

The philosophy of Gordon's classifier is stories are made up of words, and so to find stories one should look for story-relevant words. Gordon's story classifier uses unigram features [9] and so makes associations based on what words appear in stories vs. non-stories (a "bag of words" approach). The features (words) extracted from each text in the training set are used to train a confidence weighted linear classifier [5]. This is similar to a perceptron [13, 16], but it has augmentations which can improve how it learns NLP features. Although the confidence weighted linear classifier can be better suited than the perceptron for classifying certain NLP phenomena, we did not find that to be true in our story classification experiments. To train the confidence weighted linear classifier each word is run through the classifier one time (one epoch); it is uncertain how many epochs of training Gordon used, but we found that performance did not improve significantly by training this classifier for more epochs. After training, the confidence weighted linear classifier has assigned each individual word a weight that represents how relevant it is in classifying text as a story. Weights closer to zero imply that the word occurs in both stories and non-stories with similar frequency, while weights far from zero imply that the relevant word appears correlates with one of the two classes (stories or non-stories). To classify a document, its words are extracted in the same manner as they are in the training set. Then the feature counts are capped: all counts above 7 are brought down to 7. Finally, the feature values are normalized to values between 0 and 1.

There are two parts to our reimplementation: the feature extractor and the confidence weighted linear classifier. The feature extraction pipeline is built in Java, and the classifier is written in the Go programming language, in order to make use of the gonline library[3], a library of online machine learning algorithms written in Go (we could not find a usable version of the confidence weighted linear classifier in Java). We modified the gonline library in order to produce more fine-grained error statistics, and to suppress false parser errors. For the feature extractor, we use some of the same text preprocessing that Gordon provided[4]. We use the same regular expressions from his Python code to break up clitics, punctuation, and irregular characters. Then we feed the filtered data through the Stanford Tokenizer [12] to turn each document into a stream of tokens. This stream of tokens is used for the unigram counting.

[3] `https://github.com/tma15/gonline`
[4] `https://github.com/asgordon/StoryNonstory`

2.2 Improved Gordon Story Classifier

During reimplementation we experimented with replacing the confidence-weighted linear classifier with traditional single layered biased perceptron network [13, 16]. This "Perceptron" version of the Gordon classifier scored an F_1 measure 5 points higher than our implementation of the confidence weighted Gordon classifier. This is interesting because the confidence weighted linear classifier has been reported as better at classifying data from the NLP domain [5]. In this case, not only was the Perceptron classifier easier to develop than the confidence weighted one, it also had better performance. These findings are expanded upon in §3.2. In our final implementation of this improved classifier, we trained the Perceptron for 10 epochs and used a learning rate of 0.005. We settled on these hyperparameters via tuning and experimentation. Additionally, for each epoch, we randomized the order that the training vectors are shown to the Perceptron, because Perceptrons are known to be quite sensitive to the order that the training examples are learned.

The final difference between the original implementation and our improved implementation is that our encoding scheme for the frequencies is slightly different. Our augmentation is a smoothing of all the feature values: 0.07 (roughly 1/14) is added to each feature value. We did this to guarantee that the weights for each feature will be updated during the Perceptron learning, as features with value 0 do not contribute to the learning.

2.3 Reimplemented Corman Classifier

Corman's semantic triplet based classifier is trained on a wide variety of features "of varying semantic complexity" so it requires a robust linguistic pipeline to facilitate feature extraction [2, 3]. Some of the features are based on lexical properties: the densities of the 30 Penn Tree Bank part of speech tags [18], stative verbs, and person, location, and organization named entities. The term frequency inverse document frequency (tf-idf) [17] measure is calculated for each word in each document of the training corpus and, in this implementation, the words with the highest 20,000 tf-idf measures are features. The final feature is the semantic triplet. These are triplets of each verb with their respective patient and agent arguments. Sometimes this feature is actually a four-tuple where the fourth term is a locational preposition. The lexical features and triplets are extracted from each document in the training set and used to train a support vector machine (SVM) with an RBF function [10].

Corman's 2012 paper gives a high-level description of how these features are extracted from each document. Our classifier differs in a few non-trivial ways:

- We do not use the Illinois Semantic Role Labeler [14]. We use a semantic role labeler built from scratch in our lab, which is included in the Story Workbench linguistic annotation tool [7]. We used our own tool because the Illinois SRL is quite heavyweight: it requires installation of the Illinois Curator Server [4] and MongoDB[5].
- We do not do coreference resolution to replace the pronouns with their corresponding referent entity.
- We do no alias standardization. It was unclear how this should be carried out, since the named entity tagging is run on each token, and there was no explanation for how multi-word named entity boundaries were determined.
- We performed no spell checking on our named entities for the same reason as the alias standardization.

[5] `https://www.mongodb.org/`

- We only use the Stanford named entity recognizer [6] and the Illinois named entity tagger [15]. We do not use the Open Calais named entity recognizer[6] because we wanted the classifier to run without needing to query a limited resource on the internet.

We removed three portions of the text preprocessing, used a different SRL, and one fewer NER than Corman's original implementation. Even though we built a less complicated version of Corman's classifier, it performed approximately the same as (or even slightly better than) the original.

To extract semantic triplets, we first extract the parse tree for a sentence. We pass this parse tree to the SRL, which extracts all the predicates and their arguments. We take the lemmatized predicates and search for a VerbNet [19] category based on the number of arguments in Propbank [11]; we take an exact match if there is one, but take the closest match otherwise. Failing this, we return the lemma of the word from Wordnet. We follow the following rules, set forth by Corman et al. in their paper:

- If our object has multiple verbs, it is complex. We create new triplets for the object recursively and assign a pointer to the new triplets as the object for this triplet.
- If the SRL doesn't return Arg1, we substitute Arg2 for the object.
- If Arg2 is a location preposition, we include it as a fourth element.
- If Arg2 is a preposition, we use Arg1 as the subject and Arg2 as the object.

Due to some shortcomings in our SRL we may find that some or all of the arguments for a given predicate are null; in the event that some of our predicates are null, we simply tag the remaining slot (either subject or object) with a "−1" indicating a lack of an argument. If all of our arguments are null we attempt to find the closest noun behind the predicate, which we assign as the subject, and the closet noun ahead of the predicate, which we assign as the object. As above if we result in a null argument from finding the closet nouns we tag those slots with "−1".

To use the triplets as a feature, we extract all the possible subjects, objects, and location prepositions for a given verb or verb category from across the entire corpus. Then we assign each verb and verb category a specific index. These indices are determined by a simple alphabetical sort: We check every document's triplets and assign it a "1" at a given index if it has that verb or that argument for the verb and a "0" otherwise. These features are used to train a SVM that uses a radial bias function (RBF) kernel function and a soft margin C of 10,000, which is a relatively standard setting (we are not sure what soft margin parameter was used by Corman). This produces a model that can classify whether a text contains a story. To test text on the model, the same types of features extracted in training, are extracted from the test document. The feature values are used with the model to obtain a classification value.

3 Experimental Results

3.1 Reproduction of Original Experiments

To show that the reimplemented classifiers behave the same way as the originals, they were trained and tested on the same data sets as in the original studies. The Gordon confidence weighted (CW) linear classifier was trained and tested on the Spinn3r Weblog Corpus. We used the same texts that Gordon used in his study. As can be seen in Table 1, in terms of

[6] http://www.opencalais.com/

Table 1 Results for all three Gordon classifiers tested and trained on the Weblog corpus.

System	Training	Testing	Precision	Recall	F_1
Gordon Reported	**Web**	**Web**	0.66	0.48	0.55
Gordon CW	**Web**	**Web**	0.475	0.5	0.471
Gordon Perceptron	**Web**	**Web**	0.648	0.457	0.522

Table 2 Results for Corman's classifier- tested and trained on the Extremist corpus.

System	Training	Testing	Precision	Recall	F_1
Corman Reported	**Ext**	**Ext**	0.731	0.559	0.634
Corman	**Ext**	**Ext**	0.773	0.573	0.658

F_1 our Gordon CW reimplementation performs almost 8 points worse than what Gordon reported. Our Gordon CW classifier has a precision of 0.475, recall of 0.5, and an F_1 of 0.471. This could be because Gordon used a different version of the CW algorithm than we did, but it is unclear why our implementation performs so differently. Although our modified Gordon Perceptron is arguably simpler than our Gordon CW, it has a higher F_1 by almost 5 points. The Gordon Perceptron's F_1 is 0.522, which is almost 3 points less than the performance Gordon reported. We cannot say that the Gordon CW is a good reimplementation of the original Gordon classifier since the F_1 measures are significantly different. On the other hand, the performance of our Perceptron and Gordon's original classifier are quite similar, which is encouraging.

Corman et al. trained and tested their classifier on the CSC Islamic Extremist Corpus. We used the same texts during this experiment on the reimplementation. As can be seen in Table 2, our reimplementation of Corman's classifier performs similarly to the original version. The reimplementation scored a precision of 0.773, recall of 0.573, and F_1 of 0.658, which compares favorably with the originally reported results.

In terms of F_1 our performance scored slightly higher than that of the original system. This may be due to the differences in the preprocessing pipeline and the triplet extraction, as discussed in §2.3. Nevertheless, we take the similarity of the results as evidence that our reimplementation is roughly faithful to the original.

3.2 Cross-Testing the Classifiers

Because we have both classifiers and both datasets, we performed experiments to compare how each classifier performs on the other data, as well as on both datasets simultaneously. We trained and then tested both our reimplemented and improved classifiers on all combinations of the three corpora. For the cross-tests of the Gordon Classifier we use the Gordon Perceptron. As shown in the previous section, the Gordon Perceptron performs more close to the original Gordon Classifier than our reimplementation of the Gordon CW. Using the Gordon Perceptron allows for a more accurate comparison of the classifier than our Gordon CW implementation. For each of the cross-tests, the corpora go through 10-fold cross validation to generate the training and testing sets, as follows:

- If the training and testing corpora are the same, divide up the stories into ten subsets of equal size, and the not stories into ten sets of equal size. For each fold of cross validation a different story set and the not story set (of the same index) are used as the testing set and the remaining 9 are used for training.
- If the training is done on the combined corpus, and the test corpus is either the weblog or extremist corpus, which we will refer to as the single corpus, first divide the stories

Table 3 Results for Gordon Classifiers across all experiments.

System	Training	Testing	Precision	Recall	F_1
Gordon Reported	**Web**	**Web**	0.66	0.48	**0.55**
Gordon CW	**Web**	**Web**	0.475	0.5	0.471
Gordon Perceptron	**Web**	**Web**	0.648	0.457	**0.522**
Gordon Perceptron	**Web**	**Ext**	0.238	0.354	0.285
Gordon Perceptron	**Web**	**Comb**	0.594	0.008	0.014
Gordon Perceptron	**Ext**	**Web**	0.25	0.413	0.311
Gordon Perceptron	**Ext**	**Ext**	0.65	0.545	0.472
Gordon Perceptron	**Ext**	**Comb**	0.433	0.407	0.322
Gordon Perceptron	**Comb**	**Web**	0.363	0.488	0.299
Gordon Perceptron	**Comb**	**Ext**	0.62	0.508	0.426
Gordon Perceptron	**Comb**	**Comb**	0.639	0.471	0.464

Table 4 Results for Corman classifier across all experiments.

System	Training	Testing	Precision	Recall	F_1
Corman Reported	**Ext**	**Ext**	0.731	0.559	**0.634**
Corman	**Ext**	**Ext**	0.773	0.573	**0.658**
Corman	**Ext**	**Web**	0.229	0.372	0.283
Corman	**Ext**	**Comb**	0.46	0.495	0.477
Corman	**Web**	**Ext**	0.591	0.003	0.007
Corman	**Web**	**Web**	0.656	0.314	0.425
Corman	**Web**	**Comb**	0.612	0.005	0.01
Corman	**Comb**	**Ext**	0.779	0.554	**0.647**
Corman	**Comb**	**Web**	0.412	0.34	0.365
Corman	**Comb**	**Comb**	0.756	0.543	**0.632**

into ten equal sized sets, and then divide up that corpus' not stories into ten equal sets. First divide the stories in the single corpus into ten equal sets, and the not stories of the same corpus into ten equal sets. These can be split into the training and testing sets the same as in the previous situation. Additionally, the whole other corpus, the one that is not the single corpus, is added to the training set.

- If training is done on a single corpus, and the test corpus is the combined corpus, first break up the stories and not stories of the single corpus each into ten equal subsets. Assign them to the training and testing set as in the first situation. Then add the whole other corpus, the one that is not the single corpus, to the testing set.

The results can be seen in Tables 3 and 4. Macro averages for precision, recall and F_1 are reported for each experiment. We chose to report macro averaging since it was less sensitive to outliers and atypical models. Unless stated otherwise, the F_1 measures reported in this study are relative to the story class. Using story as the label to classify correctly is a good way to measure performance for this task, since the overall goal is to produce a system that can accurately identify stories.

4 Discussion

The Corman classifier has it's best performance when tested and trained on the extremist corpus, while the Gordon Perceptron does best on the weblog corpus. Both classifiers perform best when tested and trained on the corpus that it was tested on in their original studies. The Corman classifier scores best across all the experiments: 0.658 F_1 for when it is trained

and tested on the extremist corpus. Yet, the Corman classifier F_1 is 10 points worse than the Gordon Classifier when tested and trained on the weblog corpus. This suggests that the Corman classifier has a harder time learning from the weblog corpus than the extremist corpus.

The Gordon classifier performed best when trained and tested on the Webblog corpus. The F_1 measure for this experiment was 0.522, which is about 3 percentage points lower than the best result Gordon reported, and we hypothesize that this is because differences in our CW classifier implementations.

When the Corman classifier, with the Extremist model, is tested on the Weblog corpus it only has an F_1 of 0.283. This poor performance is because the model has not been trained to recognize the type of stories in the Weblog corpus. The stories in the Extremist corpus are typically 3rd person, second hand accounts, not 1st person personal accounts. Even though the Weblog corpus has significant syntactic irregularity, it contains a different type of story than that present in the Extremist corpus, so it is still useful for model training.

None of the cross-tests perform as well as the tests on the original corpus. The only cross test that comes close to the original performance is when the Corman classifier is trained on the combined corpus and tested on either the extremist or combined corpora, respectively their F_1 measures are 0.647 and 0.632. This makes sense, because the training set mostly contains examples from the extremist corpus, thus the model is more heavily influenced by that corpus. The extremist corpus has about five times more texts than the weblog corpus, hence the training set will have five times more texts from the extremist corpus. In effect, the combined model is quite similar to the extremist model.

The two weakest of all the cross tests are when the models are trained using the Weblog data and tested on the combined corpus. When trained on the Weblog corpus and tested on the combined corpus the F_1 measure Gordon's F_1 is 0.014, and Corman's F_1 is 0.007. Another particular weak experiment was when the Corman classifier is tested on the webblog corpus but tested on the extremist corpus. The F_1 for this experiment is 0.007. This suggests that the Weblog data does not generalize well to the Extremist set. So although the Corman classifier has experiments with the highest F_1 measures, it also produces the weakest models when trained on only the Weblog corpus.

We can draw a few additional conclusions from these results. The Corman classifier has better performance when trained on the extremist corpus while the Gordon classifier has better performance with the weblog corpus. This is interesting because from the stories, the Extremist corpus mainly contains second hand accounts of events, often with a 3rd person narrator. On the other hand, the Weblog corpus mainly contains person stories and 1st person narrators. So it is possible that the Gordon classifier is better at finding personal stories while the Corman classifier is better at finding second hand accounts of events. It naturally follows that the Corman classifier has better performance when trained on the combined corpus than Gordon. This is due to the extremist corpus comprising 80% of the combined corpus.

4.1 Future Work

There are a number of features not considered by Corman or Gordon that we hypothesize could lead to a boost in performance.

First, the temporal nature of stories is not considered in either system. Forster said "A story is a narrative of events arranged in their time sequence" [8]. At a minimum a story narrates time. Both classifiers are aware of whether certain lexical and verb-based features occur but not when they occurred. Implementing features that reflect the passage of time may help improve performance.

Second, stories are collections of events, and thus it may help to consider detection of events rather than verbs *per se*. The semantic triplets reflect each verb and their arguments, but these verbs do not always refer to an event. As of now, Corman's classifier generates many, many triplets, and a large number of them have don't directly have to do with the progression of events in a story.

Third, referential cohesion of the characters carrying out the events could also help with the classification. Usually events in stories are carried about by a group of characters. Incorporating features reflecting whether certain characters are talked about more, or less, or across the timeline of the story, may also improve the classifier.

5 Contributions

The experiments in this paper move the study of story classification forward. First, we reimplemented both the Gordon and Corman story classifiers. Our reimplementations are more or less faithful to the behavior of the originals, as shown by the results in Tables 1 and 2. Additionally, both classifiers were tested across the Weblog, Extremist, and combined corpora, yielding 16 new experiments (Tables 3 and 4). This is the first time these different story classifiers have been tested on the same corpora, and so could be directly compared. We also detailed the process of constructing these classifiers: the nuances and difficulties associated with their construction and how to test them. Through cross-testing both classifiers we were able to find strengths and weakness in both methods.

Acknowledgements. We thank Reid Swanson especially, as well as Andrew Gordon, for working to retrieve, format, and providing their original annotations of the Weblog corpus. Thanks to Steve Corman for facilitating the transmission of the Extremist story data, which is covered by special U.S. government FOUO rules. Also thanks to Betul Ceran and Ravi Karad for providing the list of stative verbs and examples of the extracted semantic triplets.

References

1 Kevin Burton, Akshay Java, and Ian Soboroff. The icwsm 2009 spinn3r dataset. In *Proceedings of the Third Annual Conference on Weblogs and Social Media (ICWSM 2009), San Jose, CA*, 2009.

2 B. Ceran, R. Karad, S. Corman, and H. Davulcu. A hybrid model and memory based story classifier. In *The Third Workshop on Computational Models of Narrative (CMN). Istanbul, Turkey*, 2012.

3 Betul Ceran, Ravi Karad, Ajay Mandvekar, Steven R. Corman, and Hasan Davulcu. A semantic triplet based story classifier. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pages 573–580. IEEE, 2012.

4 James Clarke, Vivek Srikumar, Mark Sammons, and Dan Roth. An NLP curator (or: How I learned to stop worrying and love NLP pipelines). In *LREC*, pages 3276–3283, 2012.

5 Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.

6 Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

7 Mark A. Finlayson. The story workbench: An extensible semi-automatic text annotation tool. In *Intelligent Narrative Technologies*, 2011.

8 Edward Morgan Forster. *Aspects of the Novel*. RosettaBooks, 2010.

9 Andrew Gordon and Reid Swanson. Identifying personal stories in millions of weblog entries. In *Third International Conference on Weblogs and Social Media, Data Challenge Workshop, San Jose, CA*, 2009.

10 S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

11 Paul Kingsbury and Martha Palmer. Propbank: the next level of treebank. In *Proceedings of Treebanks and lexical Theories*, volume 3. Citeseer, 2003.

12 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

13 Hwee Tou Ng, Wei Boon Goh, and Kok Leong Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *ACM SIGIR Forum*, volume 31, pages 67–73. ACM, 1997.

14 Vasin Punyakanok, Dan Roth, and Wen-tau Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287, 2008.

15 Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

16 Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

17 Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

18 Beatrice Santorini. Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision). Technical report, University of Pennsylvania, 1990.

19 Karin Kipper Schuler. *VerbNet: A broad-coverage, comprehensive verb lexicon*. PhD thesis, University of Pennsylvania, 2005.