RoCKET: *Ro*bust Classification and Knowledge Extraction from Text

Glenn Fung
American Family Insurance
gfung@amfam.com

Devin Conathan
American Family Insurance
dconatha@amfam.com

Sukrat Gupta
American Family Insurance
sgupta@amfam.com

Shi Yu
shi.yu@hotmail.com

Luisa Polania
American Family Insurance
lpolania@amfam.com

Abstract

In this paper we describe **RoCKET** (Robust Classification and Knowledge Extraction from Text), a machine learning platform and interface that allows users to extract information and answer non-trivial questions about a large corpus of unstructured documents. Our approach leverages state-of-the-art text representations, active learning and crowd-sourcing to efficiently label concepts and train algorithms to classify documents without requiring extensive domain knowledge expertise from the users. We claim and show empirical evidence that (1) our implementation of active learning algorithms provides a more efficient labeling experience than passive learning, (2) our text representations improve performance over baseline bag-of-word models when the number of labeled examples is small, and (3) **RoCKET** can be applied in industry settings and more specifically the insurance domain where it is a valuable tool to extract relevant customer-related information during claims processes.

1 Motivation

Humans regularly extract information from small amounts of unstructured text. For example, a person can quickly glance at a magazine's table of contents and select the articles that relate to a particular subject of interest. However, humans cannot ingest and analyze large amounts of data in the same way; there is a need to emulate the human level of understanding that can scale to large amounts of unstructured text data.

According to an estimate from IBM, 2.5 exabytes (billion GB) of data was generated every day in 2012, "About 75% of data is unstructured, coming from sources such as text, voice and video" [29]. A good portion of this information comes in the form of unstructured text. Much of the key information needed for understanding and optimizing decision-making processes in industry settings resides in human-composed, unstructured text such as messages, reviews, logs and notes. In order to leverage the power of large-scale machine learning and extract the value encoded in these data, they must be processed and transformed into structured data.

In any industry, most existing input-systems that capture structured data also capture additional unstructured data that generally require a human level of understanding in order to decipher. For example, a form typically filled in by a physician or nurse during a doctor's visit would include structured data (e.g. name, weight, blood-type, etc.) as well as typed comments about the patient's symptoms or reasons for visiting. Although these typed notes were originally meant for human consumption, there is an obvious potential to algorithmically analyze this information to provide extra value (e.g. feedback, quality measures, clinical trials, medication interaction, etc.).

Hence, in many industries, there exists a need to automatically extract key information from unstructured text. However, most commercially available systems lack a flexible, user-friendly, and configurable solution for this type of automation. Three of the reasons for this are: (a) Most of the commercial packages for this task are rule-based and require subject knowledge expertise in order to design the appropriate rules for the task. Even with considerable effort, the rules generated by users can have poor coverage and performance. (b) open source Natural Language Processing (NLP) toolkits generally require experts to configure and apply and (c) getting large amounts of ground truth data is prohibitively expensive and time consuming. It also generally requires specialized graphical user interfaces.

There are existing systems that allow users to perform simple searches to retrieve documents that contain key phrases. Sometimes these systems leverage predefined dictionaries (synonyms) or basic topic-modeling to get a slightly richer set of documents. However these search-based approaches are far from optimal and fundamentally limited in their ability to answer complex questions or extract nuanced concepts.

In this paper, we describe **RoCKET** (Robust Classification and Knowledge Extraction from Text), a system that is flexible, powerful and easy-to-use.

RoCKET allows users to automatically process a

given corpus of unstructured text records to answer non-trivial questions or extract concepts without requiring extensive knowledge about the subject. It combines state-of-the-art text representations and leverages the crowd-sourcing active learning platform NEXT [13] to efficiently gather labels from any number of users simultaneously.

In the insurance industry, there are many interactions with customers that are logged by company representatives. Some of the information found within these notes is not captured in any structure form in the corresponding systems. For example, during a call, the representative should ask the customer if renting a car is necessary or acknowledge if a car rental was requested. While existing systems would store whether a car rental has been provided, they would not indicate if it were offered or requested. Hence, there is an opportunity to extract this information from the unstructured notes. This extra information is useful for all kinds of analyses about standard of work and customer experience.

Formally speaking, for any given concept c we wish to produce a predictor function f_c that maps an unstructured body of text t to a binary label indicating if that concept is present. That is, $f_c : \mathbb{T} \to \{0, 1\}$, where $\mathbb{T}$ is our corpus. $\hat{y} = f_c(t)$ denotes our prediction of whether concept c is present in document t. To produce f_c, we take a collection of labeled pairs $\{t_i, y_i\}_{i=1}^n$ and learn a function such that $f_c(t_i) = y_i$ for as many i as possible but generalizes well to any other $t \in \mathbb{T}$. Thus, we have formulated our concept extraction problem as a machine learning classification problem.

In Section 2 we discuss related work and similar systems. In Section 3, we discuss the details the **RoCKET** framework. In Sections 4 and 5 we show empirical evidence of the performance and applicability of our system. Finally, in Sections 6 and 7 we draw conclusions and discuss some ways we are improving **RoCKET**.

2 Related Work

Concept extraction is usually domain specific, so researchers prefer to build their own tools. The recently developed DeepDive is a knowledge base construction (KBC) system to populate a structured database with information extracted from unstructured documents [32]. DeepDive is characterized by statistical inference using factor graphs constructed by entities and relationships extracted from unstructured data, and its incremental KBC process is iteratively improved by human expert supervision. In DeepDive, human feedback is provided through interaction from both a domain knowledge expert and a DeepDive-trained data scientist.

More recently, in order to shift away from data labeling dependency, Ratner et al. proposed "data programming" [27]. Data programming aims to automatically generate training data by fitting a generative mixture model comprising a handful of weak supervision strategies or domain heuristics, denoted as labeling functions. Experimental results show that discriminative models (Logistic Regression and Long Short-Term Memory (LSTM) networks[28]) optimized on training data automatically generated by these labeling functions yield comparable performance to models trained on manually labeled data. The success of data programming brings hope to replace human supervision with automated labeling functions [5]. However, this process relies heavily on the successful setup of training data, composition of labeling functions (which needs deep expert and domain knowledge) and statistical characteristics of models so it is still ongoing research.

MITIE is an open-source NLP tool focused on Named Entity Extraction (NER) and binary relation extraction [22]. MITIE offers advanced machine learning models and it achieves comparable performance with Stanford NLP on the CoNLL 2003 NER benchmark. MALLET (Machine Learning for LanguagE Toolkit) is collection of open-source Java tools which provides statistical NLP models for document classification, clustering, topic modeling and information extraction [19].

MITIE and MALLET do not provide friendly graphical user interface, and thus require extensive effort for implementation, integration, model training and performance evaluation. Neither has a web-application implementation or involves active learning for efficient and interactive model training.

Other Information Extraction systems developed earlier, such as GATE (General Architecture for Text Engineering)[23], AI::Categorizer [30], KnowItAll [30], ReVerb [1], TextRunner [2] and NELL [3] are all based on pattern matching of semantic classes and rules. It is challenging to apply these approaches to information extraction at large scales.

3 The RoCKET Framework

RoCKET aims to solve these problems with a framework that involves little intervention from NLP experts or data scientists. Utilizing the **RoCKET** framework involves the following steps:

- **Concept definition and keyword selection**: The user settles on a concept they wish to extract from the documents of interest, and then they brainstorm a short list of *keywords* that are related to the concept. **RoCKET** offers the ability to enrich this set of keywords by supplementing semantically similar words.

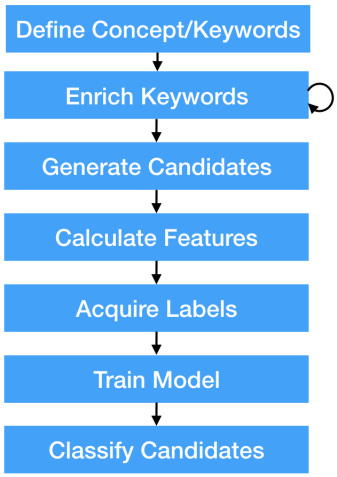

Figure 1: The **RoCKET** work-flow

- **Candidate generation**: **RoCKET** queries a database of documents and selects all documents that contain any of the keywords obtained from the previous step. These documents are *candidates* for the concept.

- **Feature calculation**: **RoCKET** embeds the documents into a vector space.

- **Label acquisition**: **RoCKET** generates a URL which can be sent to as many users as desired. The URL leads to an interface where users can label documents as positive or negative for the concept. Labels are acquired until satisfactory *stopping conditions* are met.

- **Final model training**: **RoCKET** uses the labels from the previous step to train and optimize a model which is then used to make predictions on the remaining documents.

A summary of these steps is shown in Figure 1. In the following section we describe more details about the tools, techniques and algorithms used in each of these steps.

3.1 Concept definition, enrichment, and candidate generation The first step in the **RoCKET** work-flow is to define the concept and find all documents which may contain the concept. Typically the entire corpus is very large (in our case, hundreds of millions of documents), so it is necessary to narrow down the possible candidate documents to a manageable size so features can fit in memory and algorithms can run within a practical time-frame.

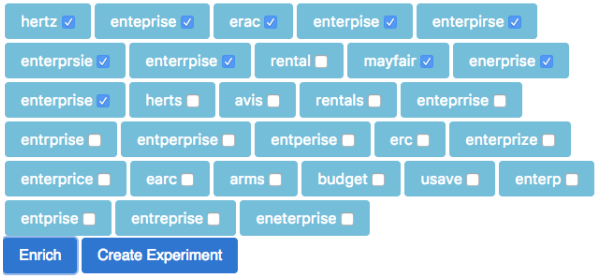

Figure 2: A screenshot from the enrichment process. The model suggests misspellings as well as semantically similar words (e.g. other rental companies).

In the car insurance domain, one might be interested in selecting all documents that contain the concept *car rental*. We define the specific concept in plain English terms, such as *customer asking about a car rental*, or *vehicular accident involving a rental car*.

Next we initialize set of keywords $C = \{c_1, c_2, \ldots, c_k\}$ of tokens or n-grams related to concept. For our example, this might be:

$$C = \{car\ rental, rental\ car, enterprise, hertz\}$$

Eventually, we will query the corpus for all documents that contain any of these keywords. At this stage, we are interested in maximum recall to obtain all possible candidates, so low precision is expected.

The keywords are generally provided by a user with some domain knowledge about the concept. However, it is not required that this is an exhaustive list of keywords, because **RoCKET** leverages a word2vec word-embedding model [21] to suggest semantically similar words by finding nearest neighbors of each keyword in the embedding space. While **RoCKET** provides the ability to use a pre-trained, general-purpose word2vec model (e.g. one trained on Wikipedia articles), we see more relevant suggestions when the model is trained on documents within the domain of interest. These models are more likely to suggest related domain-specific jargon or misspellings commonly found in the corpus being analyzed. For example, in claims notes in the insurance domain, "insured" is often abbreviated as "insd", and this relationship is easily discerned from examining nearest neighbors of each word.

We call this stage *concept enrichment*. This stage can be repeated as many times as the user desires until a satisfactory set of keywords is obtained. Figure 2 shows a screenshot captured from this process. Finally, we query our corpus for any documents containing any of the keywords in C. Beforehand, the corpus has been indexed using Elasticsearch [11], which pro-

vides the flexibility and scalability necessary to perform our token-based search over hundreds of millions of documents in a practical time-frame.

3.2 Feature calculation In this step the goal is to find a text representation that is well suited for learning models with limited labeled data. In earlier stages of this work we experimented with different text representations, including the widely used n-grams (bag-of-words) and tf-idf representations [6]. However, n-grams produces very high-dimensional representations and "the curse of dimensionality poses even greater challenges in the case of limited data, which is precisely the setup for active learning" [7]. We consistently achieved better and more robust performance by taking a word-embedding model and averaging the word vectors for a any given document. For this paper, we call this method "Text2Vec" and it is inspired by the experiments in [4]. The resulting dimension is thus the same size as the word embeddings (usually $200 - 300$) instead of in the thousands.

It is important to note that the n-gram representation can be very sparse (especially if the documents are short), and it is limited by the number of words in the bag-of-words dictionary. In contrast, the Text2Vec model produces dense representations that use all words that are present in the document provided that they have a corresponding word embedding.

3.3 Label acquisition The most difficult part of machine learning with unstructured data (and most machine learning problems in general) is acquiring a sufficiently large set of representative and good quality labels. Our goal for this step is to make this process as easy and efficient as possible. We leverage the crowd-sourcing/active-learning platform NEXT [13] to implement active learning algorithms and deploy them at the scale of many users. NEXT's primary goal is to test and evaluate active learning algorithms to facilitate research at crowd-sourcing scales, and it makes it easy to compare the performance of active learning algorithms to baseline passive (e.g. uniform sampling) methods.

3.3.1 Active learning Active learning is a subset of machine learning that addresses the issue of how efficiently algorithms can learn by choosing which samples in the dataset to train with. The typical setup for pool-based active learning for classification is: an unlabeled pool of examples $\mathcal{U}$, a labeled pool $\mathcal{L}$ of example-label pairs (x, y_x), an oracle that can supply the label of any $x \in \mathcal{U}$, and a querying strategy that selects which example in $\mathcal{U}$ the oracle should label based on the current state of $\mathcal{L}$. *Passive learning* would just sample uniformly from $\mathcal{U}$ as its querying strategy. In our case, human annotators provide the role of the oracle. Active learning is particularly useful when human annotation is involved, because one can substantially reduce the cost and time needed.

The goal of the querying strategy is to select $x^* \in \mathcal{U}$ such that $\mathcal{L}^* = \mathcal{L} \cup \{(x^*, y_{x^*})\}$ yields the maximum information gain versus $\mathcal{L} \cup \{(x, y_x)\}$ for any other $x \in \mathcal{U}$[17]. Maximal information gain is usually defined as maximally changing the predicted distribution towards the true distribution, which could be quantified by greatest decrease in risk:

$$x^* = \text{argmin}_{x \in \mathcal{U}} \mathbb{E}\left[\ell(f_{\hat{\mathcal{L}}})\right]$$

Where $\mathbb{E}\left[\ell(f_{\hat{\mathcal{L}}})\right]$ is the expected loss of the classifier trained on the labeled set $\hat{\mathcal{L}} = \mathcal{L} \cup \{(x, y_x)\}$ over the true distribution of your samples. This has some obvious practical limitations: we do not have the true distribution of our samples, and we cannot look at the label of each unlabeled example.

Instead, we might define a function $\mu(x)$ that serves as a proxy or approximation of the information gain yielded from obtaining y_x and optimize that. For **RoCKET** we implement *uncertainty sampling*, which is a querying strategy where $\mu(x)$ captures the classifier's uncertainty about the class of x[15]. The intuition is that not much information is gained if a classifier gets a new label with which it already agreed with high certainty. If your classifier outputs probabilities for each class of an unlabeled sample, one way to define $\mu(x)$ is the Shannon entropy of the predicted classes:

$$\mu(x) = -\sum_{c \in \mathcal{C}} p_c(x) \log p_c(x)$$

Where $\mathcal{C}$ are our possible classes, and $p_c(x)$ is the probability that our classifier assigns to x having class c. Note for binary classification ($|\mathcal{C}| = 2$), $\mu(x)$ is maximized when $p_c(x) = .5$ for both classes. For a linear classifier, this is equivalent to finding unlabeled samples that lie closest to the decision hyper-plane and can be implemented in $\mathcal{O}\left(|\mathcal{U}| \log |\mathcal{U}|\right)$ time.

3.3.2 Active learning architecture NEXT provides the architecture to implement active learning algorithms and scale the labeling process to as many users as necessary. The application itself is stateless, jobs are run asynchronously, and all variables and computations are stored in a database or cache accessible by all instances, meaning it is easy to distribute the processes over a cluster to scale with the number of labelers.

In order to streamline the labeling experience, we do not retrain the algorithm in real-time for every label.

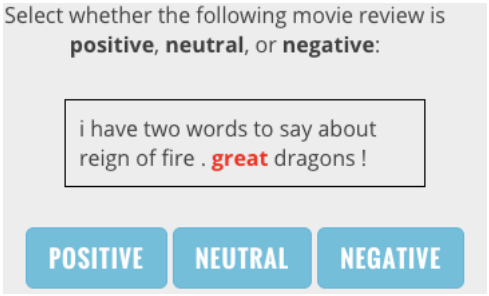

Figure 3: A example labeling UI in **RoCKET**. For this example, we are applying a three class classifier extract the concepts *positive sentiment*, *neutral sentiment*, and *positive sentiment* from movie reviews. Keywords from the candidate selection step are highlighted.

Instead, we maintain a queue of unlabeled examples chosen by the active learning algorithm. When a user is ready to label, they are simply served the first query in the queue and an asynchronous job is started to refill the queue. If the queue is empty, an unlabeled example is sampled *passively* (uniformly at random) so there is no delay in the labeler experience. The queue size is dynamically chosen based on the number of users present, the median time to run the algorithm, and the median response time for users. Since it is often the case that users take longer to label an example than it takes to run the algorithm, the queue size is usually 1, but the queue size can be larger if there are many users.

3.3.3 Evaluation and Stopping Conditions An important part of efficiently gathering labels is knowing when to stop, so you may conserve labor costs or move on to new concepts. Performance diagnostics and stopping conditions are necessary to evaluate the progress of the label acquisition and resulting model. NEXT provides the ability to build a customized dashboard for this purpose.

To evaluate progress, we initially split the candidates into train and hold-out sets. Since active learning can heavily bias your sample pool, we uniformly sample examples from the hold-out set to be labeled. These uniformly sampled examples are mixed in with the actively chosen ones so the labeler doesn't know which set they are labeling. This allows us to measure prediction performance against "ground truth" labels obtained from humans.

With a labeled holdout set, the easiest and most effective performance metric to consider is predictive accuracy on the holdout set. Since **RoCKET** is usually dealing with unbalanced sets, we use the area under the receiver-operator-characteristic (ROC) curve, or AUC.

As another stopping condition, we also consider the label stability between successive iterations. Borrowing from inter-coder reliability techniques, we compute Cohen's Kappa for the predictions made by each pair of successive classifiers as new labels are obtained. Cohen's Kappa is a measure of agreement between two labelers that is effective when there is a class imbalance as is usually the case in concept extraction. If p_o is the observed proportion of the labels that agree and p_e is the observed portion of labels that disagree, then Cohen's Kappa is:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Note that a score of 1 means there was perfect agreement. The motivation for using an agreement metric as a stopping condition is clear: if our newest classifier's predictions on a hold-out set match the predictions of the previous classifier, then the new labels did not provide any new information. [8] show that a high Cohen's Kappa between two classifiers implies a low difference in F-scores, meaning the performance of your classifier is not improving.

This stopping condition is much more valuable than the measuring holdout performance because it is unsupervised; no labels are required, so all annotation can be focused on the training set.

3.4 Final model training Once satisfactory stopping conditions are met, we use standard tuning and validation practices to produce a final model. Since we don't end up with a large quantity of labeled data, we use classifiers with limited expressive power in order to avoid over-fitting. Our final models are linear classifiers: SVMs [9], logistic regression [12] or proximal (least-squares) SVMs [10]. We use Python's scikit-learn library [25] and Pyspark's MLlib library [20] for model training and implementation.

4 Experiment I: Simulations

To evaluate the performance of **RoCKET** on realistic data, first we simulate active learning experiments using open source datasets. Throughout this section, we use the Stanford Large Movie Review Dataset [16], 20 Newsgroups [14], and Rotten Tomatoes sentence polarity dataset [24]. For the 20 Newsgroups scenarios, we set up the problem as a "one-vs-all" classification problem, where the goal was to classify each document as positive or negative for certain categories (e.g. *baseball* or *autos*). Note that this creates very unbalanced datasets (the positive class is 1/20th the size of the negative class). However, this accurately reflects real-world scenarios, where certain concepts in claims notes can be very rare. To simulate this rareness for the balanced

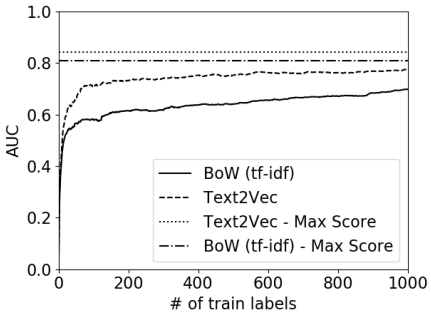

Figure 4: Performance of tf-idf bag-of-words and Text2Vec features on the Rotten Tomatoes dataset.

datasets, we only used a small sample from the positive classes.

In general, we are interested in measuring performance against the number of training labels. Our claims are about getting more performance with fewer labels. So to demonstrate our claims, we plot performance metrics against the number of training labels: a steeper curve indicates better performance. Since there is a lot of randomness involved in these experiments, we've run each of the simulations 10 times and averaged out the results to smooth the curves.

4.1 Bag-of-Words vs. Text2Vec In Section 3.2 we proposed a word2vec-based model ("Text2Vec"), since the representations are dense and lower dimensional. We claim that this means the features perform better, especially when labeled examples are limited. In Figure 4 we show the performance (AUC) of Text2Vec and tf-idf bag-of-words features on a holdout set versus the number of training labels. Since we are interested in showing the effectiveness of the features themselves, both curves are generated using passively learned training examples. The Text2Vec curve dominates the tf-idf curve even though their final max scores (i.e. with all the training labels) are similar, indicating the dense features achieve better performance with fewer labels. For all of the remaining analyses, we use the Text2Vec features.

4.2 Active learning Most importantly, we show that our implementation of active learning is more efficient than baseline passive learning given the **RoCKET** architecture. We simulated active learning and passive learning on all of our datasets. The results are shown in Figure 5.

4.3 Stopping condition: Cohen's Kappa Here we demonstrate the significance of using a label agree-

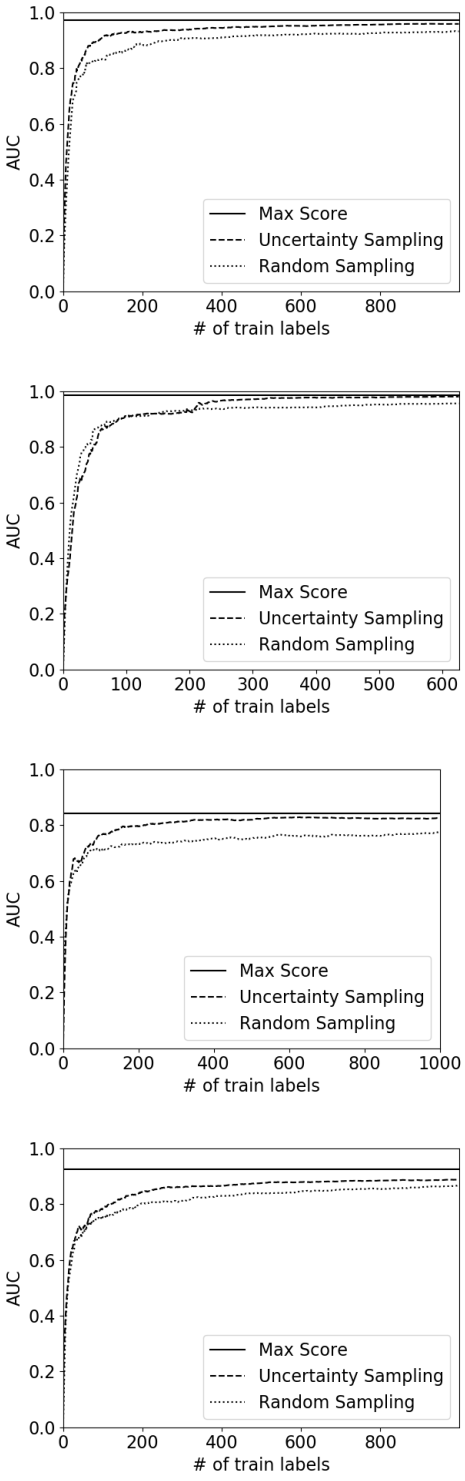

Figure 5: Active and passive learning performance on various datasets: 20 Newsgroups (autos), 20 Newsgroups (baseball), Rotten Tomatoes, and Stanford IMDB.

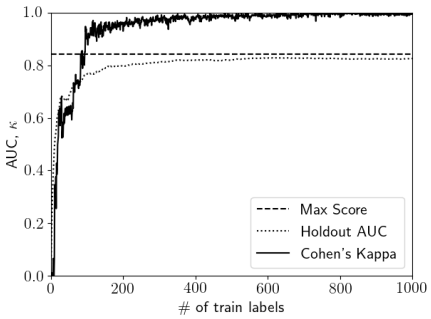

Figure 6: Active learning performance vs. label stability/prediction agreement (Cohen's Kappa) on the Rotten Tomatoes dataset.

ment technique to determine convergence. In Figure 6, we plot the holdout set AUC and Cohen's Kappa versus the number of training examples. We see that Cohen's Kappa approaches its max (1.0) at the same time that the performance of the classifier stabilizes.

5 Experiment II: Using RoCKET for the insurance industry

In insurance, like in many industries, data collection is process-driven. Legacy data collection systems are built around a specific process and only persist structured data centered around that process. However, the industry and processes are constantly changing and structured data quickly becomes outdated and irrelevant. Thus, there is an increasing need for approaches that extract information from sources that contain unstructured data and can adapt to these changes. In this section we present several cases where we applied **RoCKET** to extract information from raw, unstructured text for analytical purposes.

5.1 Concept extraction from adjuster notes In order for an insurance policy holder to get reimbursed for a loss or incident, a claim is filed with the corresponding insurance company. The claim process usually consists of a series of interactions between the insurance company and the insured and other involved parties. With every interaction, information, mostly in the form of unstructured notes, is gathered by adjusters working on the claim case. We applied **RoCKET** to extract the following concepts from a corpus of 160 million adjusters' notes:

1. **Communication:** If there was a documented communication with the insured. This includes phone calls, emails, or any contact via chat.
2. **Failed contact:** If there was a documented failed communication attempt from the company to the insured or party involved in the claim.
3. **Physical damage:** If when the claim was reported the first time, physical damage to any vehicles involved was reported.
4. **Detrimental weather:** If bad weather played an important role in the loss (e.g. hail storm in a roof claim, heavy rain or snow during a car accident, etc.).

For this deployment of **RoCKET**, we trained a word2vec model using the claims notes to be used for concept enrichment and feature generation. As mentioned in Section 3, the domain-specific model is extremely useful for the concept enrichment step because it suggests misspellings and semantically similar words found in the same corpus.

We worked with adjusters to determine and define a useful set of concepts to extract and set of keywords to use. The candidate generation step yielded between 10,000 and 50,000 notes, depending on the concept. Next, a joint team comprising both adjusters and data scientists used the NEXT interface described in Section 3.3 to label notes for each concept until satisfactory stopping conditions were met. Finally, we used the models to assign a score to all the notes in the corpus for each concept. The score indicates the probability that each concept occurs in the particular note. These scores are stored in a structured table to be used later for predictive models and other advanced analytics projects.

Figure 7 show the AUCs for the four concepts described above for both the training and the holdout test set.

5.2 Sentiment analysis for article comments A company internal website is an important vehicle to facilitate internal communication and collaboration. Often, these internal websites have options for employees to leave comments on published articles. It would be valuable to Human Resources (HR) to analyze these comments to track and measure employee engagement. We tested an open source pre-trained sentiment analysis model from the CoreNLP library [18] but the results were not satisfactory.

We deployed **RoCKET** to obtain labels for comments from 10 members of the communications team that is in charge of publishing articles on the internal company website. We collected labels for approximately 2000 comments to be used for training and testing our models. Additionally, for a disjoint set of 250 comments we obtained labels from each of 4 labelers to be used as

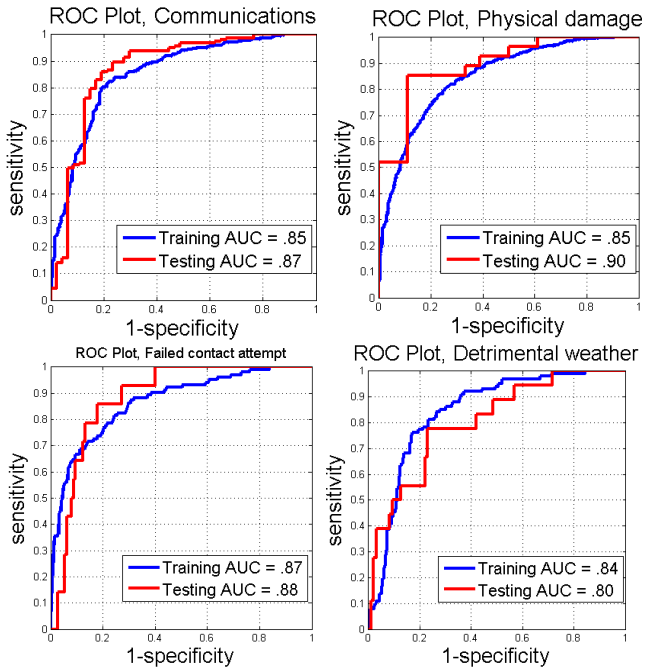

Figure 7: Training and holdout AUCs for the four concepts extracted from adjuster notes

	L1	L2	L3	L4	ST	**RK**
L1	1	0.86	0.81	0.86	0.35	0.69
L2	0.86	1	0.82	0.82	0.36	0.654
L3	0.81	0.82	1	0.84	0.38	0.70
L4	0.86	0.82	0.84	1	0.38	0.71
ST	0.35	0.35	0.38	0.38	1	0.44
RK	0.69	0.65	0.70	0.71	0.44	1

Table 1: Spearman's rank correlation between each labeler, the Stanford model and **RoCKET (RK)**. $L_i, (i = 1, \ldots, 4)$ denote the labelers and ST is the Stanford model.

a special testing set for this project. By obtaining labels from 4 different labelers, we were able to perform a more in-depth analysis of the model performance by examining inter-labeler agreement. The comments were labeled in as negative, positive and neutral, and we trained a one-vs-all classifier. The average test set AUC was 0.90.

In order to further assess performance of the model we calculated Spearman's rank correlations between each labeler, the Stanford sentiment model and the model produced by **RoCKET** over the 250 testing comments. Results are summarized in Table 1. The **RoCKET** model performs close to the human labelers and significantly outperforms the generic Stanford sentiment model.

6 Conclusions

We have designed and deployed a platform that is providing significant value to our company. In addition to the use-cases we show in this paper, we have used **RoCKET** on other company projects, including an ongoing one where we are extracting information from customers calls to automatically fill fields of a call summary form. We are pleasantly surprised by the level of engagement we have had from our business partners on NLP-related projects where they get involved by labeling text through the NEXT web-service. We are constantly improving the framework and learning as we apply it to more projects. We have many ideas to improve our framework which we will discuss in more detail in the next section.

7 Future Work

We are currently researching more sophisticated approaches to almost every aspect of the **RoCKET** framework. In this section we present a summary of some ongoing research projects.

Right now, the framework uses only one set of features. In the future, we will implement the ability to choose any number of sets of features and have **RoCKET** optimally select the best performing features. Motivated by the success of recurrent neural networks (RNNs) in sequence labeling and sequence prediction tasks, we are exploring using deep neural networks, RNNs and LSTMs to build richer text representations that have the potential to extract more nuanced information from large bodies of text. We are also interested in applying these models for the final model training.

However, applying these approaches to our framework is difficult given they generally require large amounts of labeled data to perform well. We are exploring applying the "data programming" approach and using weak supervision in the form of user-provided labeling functions combines with the active learning framework to create the large labeled datasets necessary for training more complicated models. We also are experimenting with leveraging the vast amount of unlabeled data in our datasets using semi-supervised approaches described in [26]. We are also researching improvements to the active learning algorithms and architecture to improve that process. One consequence of the queue-based approach described in Section 3.3.2 when many labelers are involved is that the queue can get filled with many similar examples (e.g. if there was a cluster of similar examples near the decision boundary), leading to a suboptimal active learning sequence. We are exploring different algorithms that incorporate a "diversity criterion" to ensure the queue gets filled with uncertain *and diverse* examples. Another consequence of having mul-

tiple labelers is that there is sometimes disagreement between labelers, especially when labelers have different levels of expertise. This scenario was considered in [31], where more complicated active learning algorithms learn different models for each labeler in order to select the best example for each individual users based on their responses so far.

References

[1] Reverb: Open information extraction software, 2003.

[2] Open information extraction: Text runner, 2007.

[3] Nell (never-ending language learner), 2010.

[4] S. Arora and Y. Liang. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations (ICLR) 2017*, 2017, to appear.

[5] J. Attenberg and F. Provost. Inactive learning?: Difficulties employing active learning in practice. *SIGKDD Explor. Newsl.*, 12(2):36–41, Mar. 2011.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, NY, 1999.

[7] M. Bilgic. Combining active learning and dynamic dimensionality reduction. In *SIAM International Conference on Data Mining (SDM)*, 2012.

[8] M. Bloodgood and J. Grothendieck. Analysis of stopping active learning based on stabilizing predictions. In *CoNLL*, 2013.

[9] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[10] G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 77–86, New York, NY, USA, 2001. ACM.

[11] C. Gormley and Z. Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2015.

[12] D. W. Hosmer and S. Lemeshow. *Applied logistic regression (Wiley Series in probability and statistics)*. Wiley-Interscience Publication, 2 edition, 2000.

[13] K. G. Jamieson, L. Jain, C. Fernandez, N. J. Glattard, and R. Nowak. Next: A system for real-world development, evaluation, and application of active learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2656–2664. Curran Associates, Inc., 2015.

[14] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[15] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR '94*, pages 3–12. Springer-Verlag, 1994.

[16] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[17] D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Comput.*, 4(4):590–604, July 1992.

[18] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60, 2014.

[19] A. K. McCallum. MALLET: A Machine Learning for Language Toolkit. http://mallet.cs.umass.edu, 2002.

[20] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *CoRR*, abs/1505.06807, 2015.

[21] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[22] MIT. Mitie: library and tools for information extraction, 2016.

[23] T. U. of Sheffield. Gate: General architecture for text mining, 1995.

[24] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*, 2005.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko. Semi-supervised learning with ladder network. *CoRR*, abs/1507.02672, 2015.

[27] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3567–3575, 2016.

[28] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pages 338–342, 2014.

[29] M. Wall. Big data: Are you ready for blast-off. *BBC Business News*, 2014.

[30] K. Williams. Ai::categorizer - automatic text categorization, 2003.

[31] Y. Yan, R. Rosales, G. Fung, and J. G. Dy. Active learning from crowds. In L. Getoor and T. Scheffer, editors, *ICML*, pages 1161–1168. Omnipress, 2011.

[32] C. Zhang. *DeepDive: A Data Management System for Automatic Knowledge Base Construction*. PhD thesis, Stanford University, 2015.