Kernel Sanders: T1-10: Text Understanding from Scratch

1st Kyle Levy
260604024
Montreal, Canada
kyle.levy@mail.mcgill.ca

2nd Nick Nasirpour
260561658
Montreal, Canada
nick.nasirpour@mail.mcgill.ca

3rd Phil Ryjanovsky
260612028
Montreal, Canada
philip.ryjanovsky@mail.mcgill.ca

***Abstract*—The aim of this project was to analyze performances of different simple classifiers on various data sets presented in the paper "Text Understanding from Scratch" [1]. In the research paper, the authors (Zhang and LeCun) presented both a small and a large temporal convolutional neural network that attempted to perform text understanding from character level inputs all the way up to abstract text concepts. While the authors of the paper trained and tested their convolutional neural networks on data sets to perform ontology classification, sentiment analysis, and text categorization, we attempted to outperform their highly complex models with simple classifiers using optimal hyperparameters found through tuning. We hypothesized that with thorough hyperparameter tuning, we could use these much simpler classifiers to at least match the complex models that they presented with much less expensive computation. For most cases, we saw that our tuned simple classifiers were slightly less accurate than their convolutional neural networks, but spent substantially less time in the training phase.**

I. Introduction

A. Background and Paper's Objective

Historically, text understanding has been resorted to matching various word statistics through tokenization techniques including methods frequency bag-of-words and binary bag-of-words. Although these methods have been relatively successful, they are restricted in that they often require prior knowledge of the language, embedded information about the words, phrases, syntactic or semantic structures. This knowledge is not cheap and, more importantly, narrows the domain of expertise any trained model would not only be tailored to a specific language, but even misspellings and slang may not be accurately recognized.

Inspired by ConvNet's success in computer vision, the authors of the paper aimed to circumvent this problem by applying deep learning from character-level inputs. Their hypothesis was that, when trained from raw characters, temporal ConvNet is able to learn the hierarchical representations of words and phrases in order to understand text. This model would then be possible to generalize the model to all languages by avoiding the necessity of training through the use of a specific dictionary. The aim was to outperform their complex convolutional neural networks as well as their baseline classifier, multinomial logistic regression, for 3 tasks:

- Ontology Classification
- Sentiment Analysis
- Text Categorization

B. Data Sets

The data sets used were obtained through various sources and included:

1) DBPedia: data set with 560,000 training examples and 70,000 test examples. Each example includes a title and a short abstract corresponding to a Wikipedia page, along with a class number representing one of 14 non-overlapping topic categories (ex: Athlete, Building, Animal). This corpus is well-maintained and probably proofread.

2) Amazon Review: data set with 3,000,000 training examples and 650,000 test examples. Each example includes text from a review on a product written by a user and a corresponding discrete numbered rating out of 5. The authors constructed a sentiment polarity data set by converting ratings 1 and 2 to negative (1), and ratings 4 and 5 to positive (2). Examples with a rating of 3 were ignored. This synthesized data set has 3,600,000 training examples and 400,000 testing examples. This corpus contains many misspellings and non-words.

3) Yahoo! Answers: data set with 1,400,000 training examples and 60,000 testing examples. Each example includes a question and the corresponding best answer, along with a class number representing one of 10 topic categories (ex: Science Mathematics, Sports, Business and Finance). This corpus contains many misspellings and non-words.

4) AG English News Corpus Categorization: data set including 120,000 training examples and 7,600 test examples. Each example includes a news article title and its corresponding description, along with a class number representing one of 4 topic categories (ex: World, Sports, Business, Sci/Tech). This corpus is well-maintained and probably proofread.

5) Sogou Chinese News Corpus Categorization: data set including 450,000 training examples and 60,000 test examples. Each example includes a news article title and its corresponding description, along with a class number representing one of 5 topic categories (ex: Finance, Sports, Entertainment). This corpus is well-maintained and probably proofread.

C. Our Objective

Our goal is to reproduce the baseline performance reported in the paper and then aim to improve the accuracy through hyper-parameter tuning. Additionally, we will explore simple machine learning algorithms, such as naive Bayes, linear

SVM, logistic regression, and random forests, to perhaps further increase performance. Ultimately, our objective is to use simple algorithms that have less computational complexity and are less intensive to challenge the performance and/or speed of the paper's ConvNet models.

II. Key Assumptions

A. Language

In order to compare with Zhang's and LeCun's generalization to all languages for his CNN, we will assume a few commonalities between all languages, the most important for our simple classifiers being:

1) A Language L, at its base contains a set of Words (a Vocabulary) $W_L \subset W$, where W the set of all Words across all Languages.
2) A Word may be broken down into a sequence of Phonemes, which can be encoded by computer-recognizable characters.
3) Words within a Language can be combined to form word clauses, $C = \sum_{w \in W_L} w$.
4) A model M can be learned to classify Clauses to Labels (e.g. meanings, ontologies, or sentiments), within a Language.

An important observation to note from our generalized definition of Language is that we make no assumptions about syntactical structure or grammar of the language, which would require prior knowledge and we believe would go against the spirit of what Zhang and LeCun were attempting to demonstrate.

III. Comparison Methods

An obvious issue that we noticed in *Text Understanding From Scratch*'s comparison method, is that one cannot truly take the CNN described by the authors [1], which trains over the sequences of frames of characters in the data set (and we believe is even expressive enough to be able to detect patterns similar to syntax/grammar), and then compare it with a simple classifier that can only see a small subset of the data set, which is limited by the size of its vocabulary (in the case of bag of words) or with a classifier that is allowed to train on the entirety of the samples, but decreases the variance of the features (in the case of bag of centroids).

IV. Pre-Processing

Each sample S contains a label and a word clause. Both of the following methodologies lead to sparse vector representations of the samples, which are very memory efficient.

A. Methodologies

- Bag of Words:
  We take all Word Clauses from the train set and count number of occurrence of each Word. We take the Words mapping to the top v (arbitrary positive) counts and this becomes our Vocabulary. For each sample S in both the train set and the test set, we create a new sample S' with the same Label, but whose Word Clause is replaced by the frequency of the occurrence of a Word in the Word Clause mapped to the Word, only if that word exists in the Vocabulary. This only requires the original data set.
- Bag of Centroids:
  We take vector representation of Words, which are learned from a large Language-specific corpus by the Word2Vec algorithm [5]. We run K-Means with k = v (arbitrary positive) and map each Word to its Cluster. For each Sample S in both the train set and test set, we create a new Sample S' with the same Label, but whose Word Clause is replaced by the frequency of a Cluster, where each Word is mapped to its Cluster, but only if that word exists in the original corpus. This requires a large Language-specific corpus.

B. Observations

- Bag of Words:
  This method is computationally inexpensive and provides a feature reduction without accounting for class prior probabilities (not the best method).
  We hypothesize that a larger vocabulary will lead to an increase in test performance. The sparse features that are added to the data set after increasing the size of the vocabulary could prove important in discriminating the posterior class probability as we know nothing about the feature's relationship with classes, or for calculating the the class prior probabilities to generate the posterior class probability (Naive Bayes).
- Bag of Centroids:
  Through the use of Stochastic Gradient Descent and Mini-Batching, we were able to obtain a k-clustering of the Word2Vec representations of the Words in the Google News Corpus for k = 5000. This was very computationally intensive and took a very long time to process, however this clustering can be used for any other data set for within the Language, and is thus is a powerful tool.
  This method serves as feature reduction, but it does attempt to account for relationships between the features, at the cost of generalizing words whose vector representations are close together in the Word2Vec Space. It directly reduces variance, and this is why we theorize Logistic Regression performs so poorly. The samples start to get closer and linearly separability decreases substantially, so it has a tough time deciding where to place its decision boundaries. On the bright side, this method should be a well-suited for a model such as Random Forests, where relationships between features can be interpreted non-linearly.
  Another note to make is that Bag of Centroids will actually account for many misspellings, which makes its use in the Yahoo Answers and Amazon Review data sets interesting.

A key observation that was made early on was that when deleting some of the most common words in the bag of words, our accuracy performance improved with all classifiers.

For the remainder of the paper when a bag-of-words of 5000 words is referred to, it is in fact the top words which exclude the some of the most common words that would not improve classification accuracy. There were different numbers of excluded words with each respective data set.

V. Models Considered

All models considered were obtained via the scikit-learn package for Python (v0.17.1 or v0.19.1).

A. *Multinomial Logistic Regression*

The multinomial logistic regression classifier was used as the research paper's baseline model. It is a linear, discriminative classifier and is trained with conditional maximum likelihood estimation. It is relatively efficient in terms of time and memory requirement, while also being robust to small noise in the data and mild cases of multi-collinearity, though we implemented regularization to handle any sort of severe cases. Our concerns regarding this model lay with the fact that its decision boundary is linear in terms of the feature space, and its decision boundary may not necessarily be appropriate for this data. We implemented batch gradient descent/stochastic gradient decent, but this still leaves room for some error.

The most important hyperparameters considered for tuning were:

- Tolerance for stopping criteria
- Regularization strength
- Alpha (SGD)
- Learning Rate (SGD)

When varying between 10-fold and 5-fold cross-validation, we noticed no real performance increase.

Given the efficiency of the classifier, we realized that increasing the vocabulary dictionary size could have a positive effect on performance. Given the number of examples used for training, doubling the dictionary from 5,000 words to 10,000 words would allow the classifier to learn more of the unique words corresponding to a category class. As expected, we saw increases in performance once training the multinomial regression model with larger dictionaries.

We were able to not only successfully reproduce the papers baseline performances, but also beat them through tuning across all of the data sets. More so, we were able to near the performances of the ConvNet models implemented by the authors with the simple baseline model that, notably, is much less computationally intensive and slow.

B. *Multinomial Naive Bayes*

The multinomial naive Bayes classifier is used for multinomially distrubuted data and many times used in text classification due to its efficiency. Naive Bayes classifiers follow a strong assumption that the features being looked at are conditionally independent of one another. This assumption can be shown with the equation:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{1}$$

where y is a given class variable and x_1 through x_n is a feature vector.

In regards to text classification, often times in real-world cases, the features are not independent of one another, immediately contradicting the inductive bias of this classifier. Initially, we believed that the Naive Bayes algorithm would perform poorly due to this. However, the classifier still performs fairly well given the main assumption is flawed. In Harry Zhang's paper, *The Optimality of Naive Bayes*, he states that no matter how strong the dependencies among features are, naive Bayes can still be optimal if the dependencies distribute evenly in classes, or if the dependencies cancel each other out [3]. Additionally, using sparse (Compressed Sparse Row) matrices, multinomial naive Bayes is extremely time efficient. Computation took approximately one second for the smaller data sets AG's news corpus and DBPedia Ontology and no more than 20 seconds for larger models like the Amazon reviews. Generally, naive Bayes performed the worst among the simple classifiers, but served as a good baseline due to its efficiency.

Naive Bayes only really has one hyperparameter to tune with which is the Laplace/Lidstone smoothing parameter, α. This parameter is used as a way to deal with features that are seen in the validation or test set, but not in the training set. Without smoothing, features seen only in the training set would classify the corresponding sentence of the testing example as impossible and thus affect its classification. In the larger data sets, tuning α made very little difference, as there were less times that a feature in the test set was not seen in the training set. In the smaller data sets, however, tuning α had large impact on test set accuracy. The difference in data set size and effects of α on the training and validation errors can be seen in figure 1:

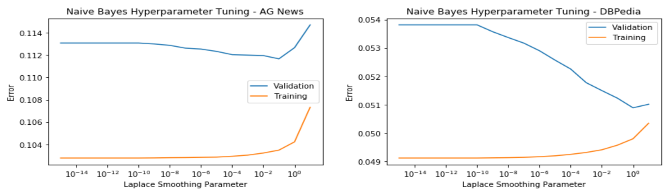

Fig. 1: These two images show how Laplace smoothing in naive Bayes greatly affects smaller data sets.

In figure 1, we used the smoothing parameter with the lowest validation error to use for the corresponding testing sets.

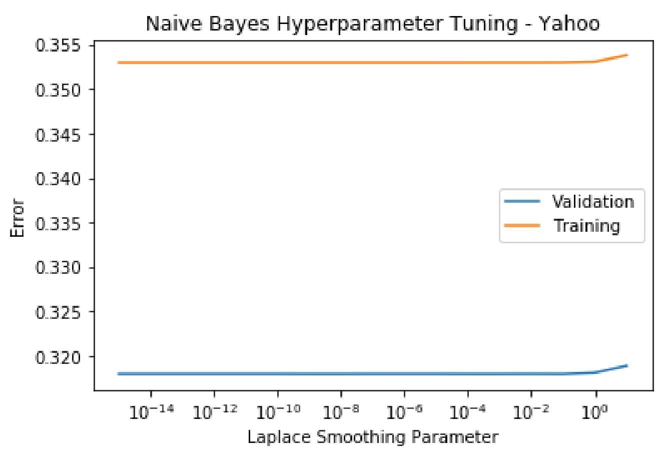

Fig. 2: Laplace smoothing has almost no effect on larger data sets.

It is clear in the graphs that we used a semi-log x plot to graph these error plots because α values from 10^{-15} through 10^{1} were tested. Using these hyperparameter values, we included models with very little smoothing along with models with a lot of smoothing. In both large and small data sets, too much smoothing led to bad classifiers, as classification accuracy approached that of a random classifier. Naive Bayes was quite simple for tuning hyperparameters and getting mediocre results within a short period of time which made it a good baseline for each of the data sets that we looked at. We did not expect high accuracy performance, and our experiments supported that expectation.

C. Stochastic Gradient Descent (SGD)

As another baseline model, we attempted to use stochastic gradient descent classifiers as variations of the normal logistic regression and linear SVM classifiers. We varied the loss function with both 'hinge' and 'log' to act as these classifiers. The most important hyperparameter tested was the learning rate which we cycled through a constant learning rate, an optimal learning rate, and an inverse scaling learning rate. The optimal learning rate takes an α parameter that is used to change the learning rate as the gradient is changing throughout the running of the algorithm. The L1 ratio was also tested which is the elastic net mixing parameter which essentially blends L1 and L2 into one penalty type. We expected a L1 ratio of 1.0 (which is L1 penalty) to perform well because it prefers sparse models, but to our surprise a L1 ratio of 0.0 (essentially L2 penalty) outperformed the L1 ratio of 1.0. These SGD classifiers gave a good insight as to what the performances of both logistic regression and linear SVM would look like.

D. Support Vector Machines (SVM)

Before actually starting to train models, we wanted to try nonlinear SVM, as we thought that the data may not have been linearly separable. We intended on using three different kernels for the nonlinear models: polynomial (degrees 2 through 4), radial basis function, and sigmoid. Knowing that nonlinear SVM scales very poorly with large data sets, we attempted to model on only a fraction of the examples in the sets that we looked at. We quickly realized that the algorithms not only took a long time to execute, but did not achieve better performance than the linear classifiers used. This is why we focused on linear SVM.

We expected linear SVM to be one of the best classifiers to use with these data sets. After seeing results from some of the other linear classifiers we concluded that the data was linearly separable. One data set that we thought linear SVM would do extremely well in was the Amazon Polarity review set. This is because there are only two classes being looked at here: positive and negative reviews. We believed that by excluding 'average' reviews (3/5 stars), the two classes would be easily separated with a linear decision boundary. As it turned out, our hypothesis was incorrect. For this data set the performance was only marginally better with linear SVM than with their baseline of logistic regression (pun intended).

Using linear SVM, we only considered the primal version due to the number of examples in comparison to the number of words that we used in our bag of words implementation. All data sets used were multi-class classifications, so a 1-vs-rest scheme was used to make decisions. The first hyperparameter that we explored was the penalty term used. Initially we thought that L1 regularization may be a better option due to the nature of the bag of words. L1 regularization prefers sparse models which is what we expected. Upon testing with both penalties, we saw that both L1 and L2 had very similar performances. In most cases actually, L2 slightly outperformed L1. For all tests, the squared-hinge lost function was used due to its compatibility with L1 regularization.

Tolerance for stopping along with the penalty parameter, C, were also thoroughly tested to see their effects. We tried ranges of tolerance from 10^{-5} through 10^{-1}. It was surprising for us to find that tolerance had very little effect on the accuracy of each model. In most cases actually, the accuracy received between the tolerances with fixed penalty terms was almost identical. As for the penalty parameter, we found that it had a much greater impact. We tested values from 10^{-4} through 10^{1}. Initially, we had tried a larger range, but saw that penalty terms smaller than 10^{-4} were too flexible, while penalties greater than 10^{1} greatly increased run-time, and gave poor results. This does make sense, as smaller penalties would not penalize misclassifications enough, while high penalty terms would overpenalize for misclassifications eventually leading to more misclassifications on the test set. For the AG News, DBPedia, and Yahoo! Answers data sets, a penalty term of $C = 0.01$ was the optimum value for that hyperparameter. For all of these data sets, linear SVM performed the best among the simple classifiers used.

E. Random Forests

We considered Random Forests due to the Decision Tree's innate ability to model multi-class problems with relative ease compared to other simple classifiers. The expressiveness of a Decision Tree allows it to fit to shape of data very well and model non-linear relationships between features, which seems ideal for learning on text-based samples. The expressiveness leads to a model that has low bias and high variance and hence overfits to the train set easily due to noise sensitivity. Random Forests allow us to maintain the low bias, while at the same time decrease the variance (by averaging over many Decision Trees and randomizing the tree creation process). [2] This moves us closer to a minimum for the test error.

The most important hyperparameters considered were:

- Number of estimators (trees) in the forest (n-estimators)
- Maximum number of features to consider at each split (max features)
- Min impurity decrease

We had originally considered rules for pruning leaf nodes as well, however it quickly became evident that setting these hyperparameters beyond their default minimums does not improve test performance enough to warrant the computation time. This is most likely because one of the primary goals of the Random Forests algorithm is to increase the randomness of the estimators by forcing splits on different randomly sampled subsets of the feature set earlier/higher up the trees. Pruning does not achieve this goal. However, setting max features to lower values does.

By lowering the value of max features, we can achieve more randomness by decreasing the probability of obtaining similar decision split nodes, and thus similarly structured trees. For AG News, we found that lowering the value to the range of 10-30 for max features leads to approximately the same train error (maintaining bias to the shape of the data) with large enough n-estimator values (between 100-500 depending on the data set), while actually increasing the test performance (from defaults of $\sqrt{70}$) and strategically selected values of 50 90 ($\pm$20 of the default), suggesting that lowering the value of max features decreases the variance of the estimators (though it would increase the bias).

It is advisable, as well, to set the min impurity split hyperparameter to a value close to 0 for small values of max features. This is because with a very small randomly sampled subset of features, the probability of finding a feature split which will actually create a valuable partition of the samples decreases.

This increase in estimators is balanced out in terms of compute power required by the fact that with less features to compute information gain or impurity over at each node, trees are created much more efficiently, however they do still occupy approximately the same amount of memory. Memory became an issue as we noticed memory increased linearly with the number of estimators in our experiments. We figured out the issue was due to the underlying design of the sklearn.ensemble.RandomForestClassifier implemented for versions prior to v0.19.1. Probabilities were computed on the individual tree level, then only aggregated at the very end. This issue was resolved in v0.19.1, however we were using v0.17.1 for a very long time so many of our computations did not work for large n-estimators and could not load the larger data sets. No noteworthy difference was observed between the Gini and Entropy criterion for determining best split, so Gini was chosen for computational efficiency.

Overall, we strongly believe that, although random forests did not surpass our other models in terms of test performance for our given vocabularies, (Bag of Words) and clusters (Bag of Centroids), they are an intrinsically well-suited data structure for classification. We further hypothesize that if trained on the Bag of Centroids model with larger feature sets (requires running KMeans multiple more timeswhich we were unable to accomplish due to memory restrictions), we may observe significant improvement in test performance. Evidence to this suggestion is in the fact that our Random Forest models using the bag of centroids with 5000 clusters outperforms the paper's logistic regression baseline by a great margin for the Yahoo Answers and DBPedia data sets.

VI. DATA SET RESULTS

The following subsections summarize the results obtained from the aforementioned simple classifiers on the data sets being modelled. Results come from the bag of words methodology with vocabulary size of 5000, unless otherwise specified.

A. AG News Corpus

The AG News Corpus was the smallest data set looked at in this paper. Due to its size, more computationally expensive methods could be used to analyze the entire set. Table I shows the results using optimal hyperparameters used:

Accuracy Received for Optimum Simple Classifiers AG News Corpus		
Classifier	Train Accuracy	Test Accuracy
Naive Bayes	89.4%	88.6%
Logistic Regression	92.7%	90.3%
Linear SVM	92.1%	**90.4%**
Random Forest	99.9%	90.0%
SGD	89.5%	88.6%
Large CNN	99.4%	**87.2%**
Small CNN	99.2%	84.4%
Baseline	88.02%	86.69%

TABLE I: This table shows the results from the simple classifiers using optimum hyperparameters found using an 80/20 validation set for AG News. The CNN results are results that the authors of the paper used with their complex models.

We can see from rigorous hyperparameter tuning that with regards to the test set all simple classifiers beat both the small and the large CNNs. The training accuracy for both of their CNNs are quite high. The difference between their training and testing accuracies displays overfitting in in the CNNs. This definitely suggests that their CNNs need a larger corpus in order to really understand the text 'from scratch'. It was stated in their paper that each epoch took about three hours with the large CNN took and one hour for the small CNN. None of the simple classifiers listed in the table took more than about 30 seconds. It is clear that with small data sets and hyperparameter tuning our simple classifiers greatly outperformed the CNNs.

B. DBPedia

DBPedia was the second smallest data set and dealt with ontology classification. While DBPedia contained approximately 50 times more examples than the AG News Corpus, it was still possible to analyze the entire data set with the computing resources that we had. We expected the issue of overfitting to be less of an issue for the authors' CNNs for this data set due to the large increase in examples. In table IV we see that our hypothesis was correct:

Accuracy Received for Optimum Simple Classifiers DBPedia		
Classifier	Train Accuracy	Test Accuracy
Naive Bayes	95.5%	95.4%
Logistic Regression	96.5%	96.4%
Linear SVM	98.2%	**97.8%**
Random Forest	99.9%	97.4%
Logistic Regression (SGD)	95.5%	95.4%
Large CNN	99.9%	**98.3%**
Small CNN	99.4%	98.0%
Baseline	96.3%	96.2%
Paper Baseline(w2v)	89.3%	89.1%
Random Forest(w2v)	99.9%	94.5%

TABLE II: This table shows the results from the simple classifiers using optimum hyperparameters found using an 80/20 validation set for the DBPedia data set. The CNN results are results that the authors of the paper used with their complex models. Note: Random Forest trained on randomly sampled 10% of train data, memory restrictions did not allow for grid search for hyperparameter tuning.

The table indicates how the larger data set greatly helped the CNNs learn the model. In this case the CNNs learned the model better than all of the simple classifiers. This did not come as a surprise to us. With that being said, linear SVM performed extremely well with a 97.8% accuracy in the test set. This is just below the performances of both the large and small CNNs with a fraction of the execution time. Depending on the task at hand, it may be more practical to use a linear SVM than those large complex models. With proper tuning linear SVM showed accuracies very comparable to the CNNs.

The w2v results for Random Forests, which are without fully-exhaustive hyperparameter tuning, provide evidence to prove our theory that the bag of centroids approach is well suited for the Random Forest model, but not for logistic regression.

C. Yahoo! Answers

The Yahoo! Answers data set consisted of 1,400,000 examples consisting of a question, question description, and the best answer to the question. Again this data set had significantly more examples than both of the previous data sets. It is important to note that for each of the simple classifiers only a random 30% of the data set was used to tune the hyperparameters. We realize that this may very well lead to non-optimal hyperparameters, but due to resources and computation limitations (computer crashes, memory issues), we went under the assumption that optimal hyperparameters for the random 30% should at least be near the true optimum hyperparameters. Therefore, it was again hypothesized that the CNNs would outperform the simple classifiers at the expense of computation. However, for linear SVM the hyperparameters were even more finely tuned in the neighborhood of the optimal hyperparameters obtained from the 30% of the data used. The full data set was used for the fine-tuning mentioned to hopefully receive slightly different hyperparameters that are more indicative of the true model. Table V supports our hypotheses.

Accuracy Received for Optimum Simple Classifiers Yahoo! Answers		
Classifier	Train Accuracy	Test Accuracy
Naive Bayes	64.8%	64.9%
Logistic Regression	59.7%	59.7%
Linear SVM	68.6%	**68.0%**
Random Forest	99.9%	62.6%
Logistic Regression (SGD)	64.8%	64.9%
Large CNN	73.4%	**70.4%**
Small CNN	72.8%	70.2%
Baseline	66.8%	66.6%
Paper Baseline(w2v)	56.37%	56.47%
Random Forest(w2v)	99.9%	61.7%

TABLE III: This table shows the results from the simple classifiers using optimum hyperparameters found using an 80/20 validation set. The CNN results are results that the authors of the paper used with their complex models. Note: Random Forest trained on randomly sampled 10% of train data, memory restrictions did not allow for grid search for hyperparameter tuning.

As expected, the CNNs presented in the authors' paper outperformed the simple classifiers that we implemented. Even with a very substantial tuning with the seemingly best simple classifier, linear SVM, we could not match their results. At a certain point, it became clear that we would not be able to use simple classifiers to beat the CNNs.

The w2v result for Random Forest, trained on a randomly sampled 10% the train set, and without exhausting hyperparameter tuning, outperforms the paper's baseline w2v again, giving further evidence to our theory that the bag of centroid approach is better suited for Random Forests.

D. Amazon Reviews

The Amazon Reviews data set was by far the largest, with 3,000,000 training examples across 5 classes. The constructed polarity dateset included 3,600,000 training examples across 2 classes. We were limited to the simplest classifiers due to the size of the data set, and were not able to run Random Forests with realistic and effective parameters that gave competitive results. However, Naive Bayes, Logisitc Regression and Linear SVM gave very strong results on both the Full data set and the Polarity data set, beating the baseline and coming within a couple percent of the ConvNet, all the while training much faster - unlike the Large and Small ConvNet that took 5 days and 2 days, respectively. Were we not limited by our processing power, we believe our excellent performance on the smaller data sets could be replicated on the Amazon data set, and would result in matching the ConvNet performance in much less time spent training. Despite the large number of examples in the polarity data set, we were able to run the models quite quickly and effectively. This is due to the fact that the polarity construction removes the examples rated 3/5 and groups 1/5 with 2/5, and then 4/5 with 5/5. This construction decreases the confusion of the classifiers by increasing the separability between the classes. Notably, SVM can run quickly and with the best performance by establishing the support vectors along the examples rated 2/5 and 4/5, within the negative and positive classes, respectively.

Test Accuracy Received for Optimum Simple Classifiers Amazon		
Classifier	Full data set	Polarity data set
Naive Bayes	49.5%	85.2%
Logistic Regression	54.0%	90.0%
Linear SVM	53.6%	**90.2%**
Linear SVM (SGD)	53.7%	90.0%
Large CNN	58.7%	94.5%
Small CNN	59.5%	**94.5%**
Baseline	54.17%	89.9%

TABLE IV: This table shows the results from the simple classifiers using optimum hyperparameters found using an 80/20 validation set. The CNN results are results that the authors of the paper used with their complex models.

E. Sogou News

We did not at all expect good results from this data set given that the training and testing file converted from Chinese had numbers within words, due to improper translation of certain Chinese characters. We removed these numbers in pre-processing, leaving just the converted letters. This leaves open the possibility that differing Chinese words would end up looking identical in English after the translation and removal of improperly converted characters. Nonetheless, we were able to train Naive Bayes, as well as SVM and Logistic Regression using SGD, on the corrupted translation and achieve results near the paper's baseline and ConvNet performance.

Accuracy Received for Optimum Simple Classifiers Sogou News		
Classifier	Train Accuracy	Test Accuracy
Naive Bayes	85.3%	85.3%
Logistic Regression (SGD)	89.8%	**89.5%**
Linear SVM (SGD)	90.5%	88.6%
Large CNN	99.1%	**95.1%**
Small CNN	93.1%	91.4%
Baseline	93.0%	92.8%

TABLE V: This table shows the results from the simple classifiers using optimum hyperparameters found using an 80/20 validation set. The CNN results are results that the authors of the paper used with their complex models.

VII. Conclusion

Given limited knowledge of a language other than frequencies of words, simple classifiers trained using the bag of words model are able to perform competitively against the CNN described in the paper that was put forth by Zhang and LeCum for nearly all of the data sets. Naive Bayes was a particularly poor choice of classifiers for these approaches, as it is outperformed during virtually every experiment by both logistic regression and Linear SVM - our most preferable model from these experiments. Naive Bayes was not expected to do well, but served as a very good baseline for us as computation was quick and gave mediocre accuracy performances.

Random Forests trained using the bag of centroids approach had promising early results in comparison with logistic regression, which was a poor classifier to train on bag of centroids, as many more of the samples cluster close the decision boundary, reducing linear separability. Given unlimited computing power, we would have been able to run K-Means efficiently and test what a larger feature set for bag of centroids method would have resulted in. We would have also been able to run more exhaustive grid searches to tune hyperparameters and hence perform better in prediction for the test set. Depending on what the experimenter is attempting to accomplish, it is quite possible that simple classifiers with an exhaustive hyperparameter tuning could be the experimenter's preferred option.

VIII. Summary of Contributions

For this project Kyle Levy focused on the simple classifiers and hyperparameter tuning for the AG News, DBPedia, and Yahoo! Answers. He also wrote the corresponding sections for those data sets along with the linear SVM, naive Bayes, SGD, and abstract sections in the report. Nick Nasirpour worked to pre-process the data and turn the original raw data into readable CSV files that could be turned into sparse matrices. He also worked extensively with random forests for each of the data sets, as that is a classifier that we thought would perform very well. He wrote the random forest and pre-processing sections of the report. He, as well worked the most with the bags of centroids. Phil Ryjanovsky wrote the introduction, logistic regression, and Amazon data set sections of the report and completed some of the NB, Logistic Regression and SVM analyses on the Amazon data sets, as well as preliminary RF on the Yahoo! data set. We all contributed to the code as we each had slightly different ways of analyzing the data and hyperparameters. We hereby state that all the work presented in this report is that of the authors.

References

[1] Zhang, Xiang, and Yann LeCun. "Text understanding from scratch." arXiv preprint arXiv:1502.01710 (2015).
[2] Leo Breiman. 2001. Random Forests. Mach. Learn. 45, 1 (October 2001), 5-32. DOI: https://doi.org/10.1023/A:1010933404324
[3] Zhang, Harry. "The optimality of naive Bayes." AA 1.2 (2004): 3.
[4] Lehmann, Jens, Isele, Robert, Jakob, Max, Jentzsch, Anja, Kontokostas, Dimitris, Mendes, Pablo N., Hellmann, Sebastian, Morsey, Mohamed, van Kleef, Patrick, Auer, Soren, and Bizer, Christian. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web Journal, 2014.
[5] Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S., and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In Burges, C.j.c., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K.q. (eds.), Advances in Neural Information Processing Systems 26, pp. 31113119. 2013b.

IX. Appendix

The purpose of this project was to test the accuracy performances of our hyperparameter tuned simple classifiers versus the authors' complex convolutional neural networks. We attempted to make these classifications by using methods such as bag-of-words and bag-of-centroids to construct sparse matrices that contained information about word usage in certain datasets. We made no assumptions about the grammar or syntax of the underlying language, such as word placement. One of the main strategies that further increased performance by a range of 1%-2% was the exclusion of certain top 'common' words that have no descriptive meaning. While it served as a slight dimensionality reduction, this elimination was primarily used to disallow these uninformative words from affecting the classification decision.

In our pursuit to beat the authors' convolutional neural networks' performances, we implemented three simple classifiers: naive Bayes, logisitic regression, and linear SVM. Linear SVM and logistic regression used two variations of gradient descent: batch gradient descent and stochastic gradient descent. Stochastic gradient descent proved to be extremely important in executing these classification algorithms in a reasonable amount of time. Random forests which is an ensemble method (essentially bagging with decision trees), however, was used extensively in order to greatly reduce the variance of each learned model. We found that random forests were effective in learning the models of the data sets presented in *Text Understanding from Scratch* by Zhang and LeCun, but still beat by linear SVM. While in most cases we could not beat the authors' small and large CNNs, our simple classifiers made classifications in substantially less time than their complex models, with accuracy performances near, but slightly less than what they had achieved. The following tables show our best simple classifer as well as the results obtained from Zhang's and LeCun's CNNs as well as their baseline model which was logistic regression.

Accuracy Received for Optimum Simple Classifiers AG News Corpus		
Classifier	Train Accuracy	Test Accuracy
Linear SVM	92.1%	**90.4%**
Large CNN	99.4%	**87.2%**
Small CNN	99.2%	84.4%
Baseline	88.02%	86.69%

Accuracy Received for Optimum Simple Classifiers DBPedia		
Classifier	Train Accuracy	Test Accuracy
Linear SVM	98.2%	**97.8%**
Large CNN	99.9%	**98.3%**
Small CNN	99.4%	98.0%
Baseline	96.3%	96.2%
Paper Baseline(w2v)	89.3%	89.1%
Random Forest(w2v)	99.9%	94.5%

Accuracy Received for Optimum Simple Classifiers Yahoo! Answers		
Classifier	Train Accuracy	Test Accuracy
Linear SVM	68.6%	**68.0%**
Large CNN	73.4%	**70.4%**
Small CNN	72.8%	70.2%
Baseline	66.8%	66.6%
Paper Baseline(w2v)	56.37%	56.47%
Random Forest(w2v)	99.9%	61.7%

Test Accuracy Received for Optimum Simple Classifiers Amazon		
Classifier	Full data set	Polarity data set
Logistic Regression	**54.0%**	90.0%
Linear SVM	53.6%	**90.2%**
Large CNN	58.7%	94.5%
Small CNN	**59.5%**	**94.5%**
Baseline	54.17%	89.9%

Accuracy Received for Optimum Simple Classifiers Sogou News		
Classifier	Train Accuracy	Test Accuracy
Logistic Regression(SGD)	89.8%	**89.5%**
Large CNN	**99.1%**	**95.1%**
Small CNN	93.1%	91.4%
Baseline	93.0%	92.8%

It is essential to consider the task at hand when deciding between complex models and simple classifiers. Considerations include time constraints, computation resources, and size of the set. As we saw with our experiments, smaller data sets worked better with tuned simple classifiers. As the size of the data set increased, more complex models like the authors' CNNs outperformed the simpler classifiers. With regards to time constraints simple classifiers proved to obtain accuracy within a few percent of the complex model's accuracy, but ran extremely fast in comparison.