UNIVERSITY OF COPENHAGEN

Language Processing 2

Sentiment Analysis

Andreas Rugård Klæsøe,	nmh341
Qian Dong,	krf303
Chuang Wu	whj433

Department of Nordic Studies and Linguistics
University of Copenhagen

December 4, 2018

Contents

1 Introduction

Sentiment analysis is a method for predicting emotion or opinion - i.e. predicting sentiment - expressed by a person in some piece of communication. In this case we are working only with sentiment analysis of text.

Contrasted to many topic-based multi-class text categorization problems, sentiment-related multi-class classification can naturally be formulated as regression problems because ratings are normally ordinal[5]. In general, sentiment analysis works by assigning a rating of some manner to a text. The ratings can be as simple as "positive", "neutral", "negative" or they can be some numerical range. A specific example might be sentiment analysis of reviews, where the text is the review itself, and the score is the review score. The sentiment analysis tool would have to predict the actual score based on the review text.

The reasons for performing sentiment analysis are legion, but include among others: business analytics, collating word-of-mouth style consumer information, and targeted advertisements. Being able to, with good accuracy, automatically assign ratings to huge collections of texts is an absolute requirement for these tasks, as the sizes of the collections quickly grow to exceed what could reasonably be rated using humans in similar timeframes.

At heart, the process for sentiment analysis is quite simple. Given texts with labels, determine features in the text to use to predict the label. This might be as simple as counting the frequency of each word, and relating words to one of the labels. More sophisticated approaches involve feature extraction based on semantic qualities. For instance classifying words such as "good" because they semantically denote "positive" and extracting features based on these can lead to superior results. Even more complex feature extraction models may include accounting for negation, or mapping words into a Word Embedding space and using the resulting vectors to take advantage of the structures implied by the Word Embeddings.

In our specific case, we compare uni-gram and bi-gram models using features based on frequency, as well as Word Embeddings and Word2Vec.

Given some feature representation, different machine learning algorithms may then be used to predict sentiments. As a rule of thumb, the most appropriate ones would be non-binary, discrete classifiers, but depending on the label type, binary or continuous classifiers may be appropriate.

Results vary based on the dataset in question, but research into the state of the art suggests that Long Short-Term Memory methods have the best generalized performance over varying datasets [2]. We do not make use of this algorithm, but instead compare several other classic machine learning algorithms.

2 Previous Studies

Early work by Yang and Liu (1999) examined a controlled study with statistical significance test on linear Support Vector Machines (SVM), k-Nearest Neighbors (kNN), LLSF, Naive Bayes (NB) and Neural Networks (NN). They set k to 45 in kNN, and the number of hidden units in middle layer of the NN was set to be 64. Their results showed that SVM, kNN and LLSF significantly outperformed NN and NB in text categorization[25]. Considering the expensive training time and relatively low performance, we can see why NN was not widely used in the early time for text categorization.

Algorithm	**Nnumber of features**
Linear SVM	10,000
kNN	2,415
LLSF	2,415
NN	1,000
Naive Bayes	2,000

Table 1: Number of features in Yang and Liu's study(1999)

Bo Pang et al. conducted a study on sentiment classification using machine learning techniques in 2002[3]. They exploited three different machine learning techniques (Naive Bayes, Maximum Entropy and Support Vector Machines) to classify movie reviews and examine whether it suffices to treat sentiment classification simply as a special case of topic-based categorization (with the two "topics" being positive and negative). The accuracies they achieved were around 78%-82%, which were impressive but not comparable to those reported for standard topic-based categorization [3]. They discussed the reason for this and suggested it could be a "thwarted expectations" narrative, where the author sets up a deliberate contrast to earlier discussion. For example:
"This film should be brilliant. It sounds like a great plot, the actors are first grade, and the supporting cast is good as well, and Stallone is attempting to deliver a good performance. However, it can't hold up" or "I hate the Spice Girls. ...[3 things the author hates about them]... Why I saw this movie is a really, really, really long story, but I did, and one would think I'd despise every minute of it. But... Okay, I'm really ashamed of it, but I enjoyed it. I mean, I admit it's a really awful movie ...the ninth floor of hell...The plot is such a mess that it's terrible. But I loved it."[4]

They had also mentioned another common theme, "a good actor trapped in a bad movie":
"AN AMERICAN WEREWOLF IN PARIS is a failed attempt...Julie Delpy is far too good for this movie. She imbues Serafine with spirit, spunk, and humanity. This isn't necessarily a good thing, since it prevents us from relaxing and enjoying AN AMERICAN WEREWOLF IN PARIS as a completely mindless, campy entertainment experience. Delpy's injection of class into an otherwise classless production raises the specter of what this film could have been with a better script and a better cast ... She was radiant, charismatic, and effective ...." [4].
This thwarted-expectations could appear in many types of texts. Hence they believed that the important next step to increase accuracy is the identification of features indicating whether sentences are on-topic[4].

In 2013, Mikolov and his Google team proposed two novel model architectures: Continuous Bag of Words(CBoW) and Continuous skip-gram, which are the important elements in word2vec, or the word embedding framework. The new models they proposed maximized accuracy of vector operations, while minimizing the training time. Compared to Feedforward Neural Net Language Model(NNLM), the expensive non-linear hidden layer was removed from CBoW, and the projection layer was shared by all words (not just the projection matrix)[12]. The skip-gram model is like a reversed version of CBoW. Instead of predicting the current word, it maxmizes classification of a word based on another word in the same sentence[12]. They compared different architectures using models trained on the same data, with 640 dimensional word vectors, and skip-gram outperformed Recurrent Neural Net Language Model(RNNLM, NNLM and CBoW, with accuracy of 55% and 59% in semantic and syntactic tasks respectively.

3 Method and Results

3.1 Material and Normalization

The data used in this experiment was $Tweets - airline - sentiment.csv$,in a format of 15 columns and 14606 rows. Most of the information from the dataset was discarded for the purposes of this study, retaining only the labels and the text. A total number of 14605 tweets and their corresponding sentiment-tags were extracted from the csv file. Python library *RE* was adopted in normalizing tweet-corpus before the building feature representations. Arabic numbers(0-9) and airplane company names at the start of each tweet were removed during normalization. Case differences, special characters and punctuation were also removed during tokenization or parsing steps inasmuch as was possible.

3.2 Feature Representations

Converting a document (tweet) into a feature vector or other representation that makes its most salient and important features available is an important part of data-driven approaches to text processing[6]. In this chapter, we show our experiment on n-gram, word-embedding, word2vec (Continuous Bag of Words, continuous skip-gram) for feature representations.

3.2.1 n-gram

A simple definition of n-grams is that they are sequences of n words extracted from a text. A more rigorous definition is that an n-gram is a contiguous sequence of n items from a given sample of text or speech[13]. For instance, given a sentence, 'This sentence is going to be used to generate uni-gram or bi-gram', the uni-grams are 'This', 'sentence' and 'is', etc. while the bi-grams include 'This sentence', 'sentence is' and 'is going' and so on.

The reasons to use n-gram models (and algorithms that use them) include the following two benefits: simplicity and scalability. With larger n, a model can store more context with a well-understood space-time trade-off, enabling small experiments to scale up efficiently.[13]

However, machine learning algorithms are not capable of analyzing the raw input text as most of them expect numerical feature vectors with fixed size rather than the raw text documents with variable length.[1]

To address this issue, we use the well-known machine learning package $scikit-learn$ which includes an API for text processing.

The scikit-learn package, $CountVectorizer$, implements both tokenization and occurrence counting in a single class[1], returning feature vectors. To be more specific, it assumes that the frequency of each individual token (i.e. uni-gram) is a feature. Furthermore, each document in the corpus could be interpreted as a feature vector composed of these n-gram features. In this paper, when generating features such as uni-gram or bi-gram, the parameter settings were as follows: *encoding* was set to 'utf-8', *ngram_range* was set depended on whether we want to generate uni-gram-feature or others, *lowercase* was activated in order to convert all characters to lowercase before tokenizing[16] and $max\_features$ was set to different number(for instance 500 or 1,000 so that we could focus on the top 500 or 1,000 features). Finally we obtain a sparse matrix of token counts from a collection of text documents which facilitate our research of sentimental analysis with $CountVectorizer$.

3.2.2 Word Embeddings(Context Free)

Word Embeddings are mappings from words to some high-dimensional vector space, here $word \rightarrow \mathbb{R}^{300}$, where the values are initially randomized, and subsequently altered to generate some underlying structures which can then be reused in other applications.

In practice this structure generally manifests as fixed difference vectors encoding shifts in meaning, for instance a difference vector between "male" and "female" words, which when identified can be used to shift from the word "king" to "queen" simply by sliding along the difference vector, starting at "king".

We used a shortened set of pretrained word embeddings based on English wikipedia by Bojanowski et al[7].

Given the pretrained word embeddings, we built a feature representation of the data where each word in each tweet was mapped to a word embedding vector, which was then been turned into a large, flat feature vector. Since the vectors had to all be the same length, but the tweets contained varying amounts of words, the vectors were padded or sliced where necessary. A fixed parameter, the number of words per tweet, was established, and all tweets were either padded with zero vectors or cut down until they fit this length. Padding was done at the start of the feature vector, and cutting was done from the end of the feature vector. The resulting feature vector was $N \doteq 300$ large, where N was the number of words per tweet. The resulting representation worked with all the algorithms except Naive Bayes, which assumed no negative values, a trait word embeddings wasn't guaranteed to satisfy. While it would be possible to account for this by generating a mapping from the word embeddings into a non-negative feature space (eg. by discretizing them by running the K-means algorithm), we elected not to do this.

3.2.3 CBoW and skip-gram

CBoW is a context-based model, which generates vector representations for the words in the range of windowsize apart from the current word, and can be used to predict current word based on the given context. The structure of CBoW is shown as Figure 1 below. Whereas skip-gram has a similar but reversed structure of CBoW, and predict possible classification of a word based on another given word nearby.

We built CBoW model from *keras* and *Sequential*. The unacceptable long training time for vector-generation forced us to drop the CBoW in keras and switched to Gensim. Gensim had its own implemented word2vec function, which took only a few seconds to generate vector-model for entire corpus in our experiment. Radim Rehurek, the creator of Gensim had optimized word2vec function by applying Cython, BLAS and C. The uasge of *saxpy* or *dgemm* in BLAS are routines for vector calculation, such as $y \mathrel{+}= \alpha * x$ where x and y are vectors and α is scaler. Those basic operations can be heavily optimized in saxpy, which is much faster than a naive C loop. The original C code also contains an extra optimization in RAM and CPU, which seems to be unnecessary in modern computers and was removed. All these optimization results in 70 times speedup compared to plain NumPy implementation[14].

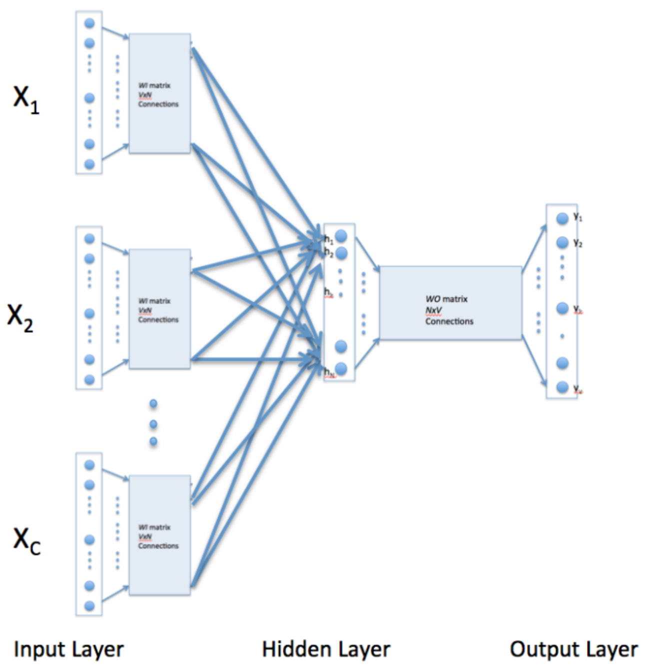


Figure 1: Architecture of CBoW [9]

Feature representation created by word2vec was trained from the tweet-corpus based on *Tweets-airline-sentiment.csv* file, with vector dimension varied in 100, 200 and 300. The window size was a fixed value of 5. Default sampling, negative sampling was used, and the minimum count of frequency was set to be 5.

3.3 Algorithms

We had exploited NB, Perceptron, SVM, LR, kNN, Multilayer Perceptron and RF in our experiment.

3.3.1 Naive Bayes

Naive Bayes classfier is one of the successful text processing technique and was introduce to text mining in the early 1960.[15] It is based on Bayes' theorem, and construct the classifier by assuming each feature is conditionally independent of each other and then choose the hypothesis with maximum probability.

Although the model is simple and one of the assumptions that features are independent of each other might be flawed considering the real condition, Naive Bayes-based text catergorization still performs well.[11]

The implementation was done by scikit-learn[20] and we used the default settings: $alpha = 1.0$, fit_prior = True.

3.3.2 Perceptron

The Perceptron will generate a hyperplane that separates the classes. The specifics depend on the exact type of perceptron used. The specific implementation here is based on Stochastic Gradient Descent[21]. It is a simple model, but it has the advantage that the worst-case runtime scales almost linearly with the number of training samples. It performs best when the data is linearly separable, since it will attain 100% accuracy in this case. The implementation was by scikit-learn[21].

We used the default settings: $alpha = 0.0001$, $fit\_intercept$ set to True, $shuffle$ set to True, $n\_jobs = 1$, and $random\_state$ set to None.

3.3.3 Support Vector Machine

Support Vector Machines (SVMs) generate a hyperplane that divides the classes in the feature space such that the margin from the classes to the hyperplane is as large as possible. The training time is high, with the implementation we use taking between $\mathcal{O}(n_{features} * n_{samples}^2)$ and $\mathcal{O}(n_{features} * n_{samples}^3)$ [23], though as our features are generally sparse, $n_{features}$ may be replaced by the average number of non-zero features. This high cost is part of what lets SVMs require few assumptions about the data to attain good results. We used an implementation by scikit-learn[23].

We used linear SVM with default settings:$penalty$ set to 'l2', $loss$ set to 'squared_hinge', max_iter set to 1000 and $fit\_intercept$ set to True.

3.3.4 Logistic Regression

Logistic Regression attempts to determine the coefficients of a logistic model, usually a log-likelihood model. Using this model allows for assigning probabilities that data points will belong to a class. It has highly variable training time, but under good conditions will converge to a solution quickly. Prediction is done in constant time. To use logistic regression, the data must be receptive to continuous numerical methods and ideally approximably by a non-degenerate convex function. It often benefits from low amounts of overlap or noise in the data to make the log model smoother. We use an implementation by scikit-learn[18].

We made no special parameter selection for logistic regression: $fit\_intercept$ set to True, *normalize* False and n_jobs set to 1.

3.3.5 K-Nearest Neighbours

K-Nearest Neighbours is a simple classifier that assigns label to new data based on the label of the majority of its K nearest neighbours. In its base formulation it takes O(N) training time, and O(N) time per prediction. More advanced data structures can reduce prediction time at the cost of more time taken to encode the training set. To work well, the data must naturally cluster with low overlap. It suffers from high dimensionality as it is based on distance measurements. We use an implementation by scikit-learn[17].

We used K = 3 for all runs. We did attempt variant values of K, but it made no significant difference in accuracy so we have elected not to report the results.The rest parameters setting are as followed: $leaf\_size$ set to 30, *metric* set to 'minkowski and n_jobs is 1.

3.3.6 Multilayer Perceptron

The Multi-layer Perceptron is a variant of the standard perceptron which includes hidden layers, allowing it to approximate non-linear functions. The time complexity scales badly with the number and size of hidden layers, especially if there are many of them. Further, the loss-function is not convex, and as such there is no guarantee that any given run will find the global optimum. However for a problem that appears to clearly be non-linear it is a very flexible method. We use an implementation by scikit-learn[19].

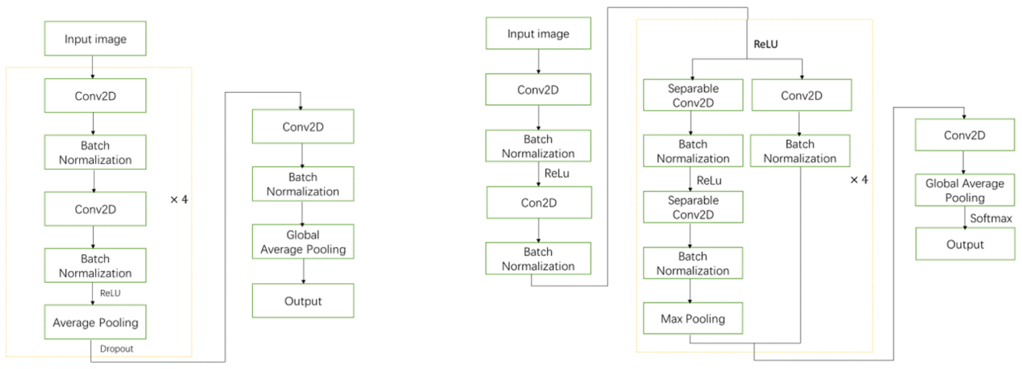
We used the default settings: $hidden\_layer\_sizes$ set to 100, *activation* set to 'Relu', *solver* set to 'adam', $batch\_size$ set to 'auto', max_iter 200.

3.3.7 Random Forest

Random Forest generates a large number of Decision Trees during training, which then vote on the correct label during testing. This combats the tendency of Decision Trees to overfit. Random Forest training time is proportional to the number of trees by the size of the trees, while testing is worst-case proportional to the depth of the trees by the number of trees. We use an implementation by scikit-learn[22].

We used forests of size 10 for all runs of Random Forest, and the criterion is 'gini'.

3.3.8 Convolutional Neural Network: Xception



(a) Simple Convolutional Neural Network (b) Xception

Figure 2: Visualization of Two Types of Neural Network

Convolutional neural network (CNN) is a class of deep, feed-forward artificial neural networks in machine learning. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing[24]. A typical CNN could consist an input layer, an output layer, multiple convolutional layers(feature extraction), pooling layers(reduce dimension), fully connected layers and normalization layers.

Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

In this paper Xception, one of CNN models, is implemented because the number of parameters of such model is less than normal CNN's which greatly reduces the computation cost.The first model structure shown in figure 2a, implements a classical CNN structure, where the convolutional layer becomes thicker as it goes deeper. The Batch Normalization layer added after each convolution operation, accelerates the training step [10]. The Fully Connected layer is deleted to reduce the parameters. The second model structure 2b is adapted from the Xception model developed by [8]. Moreover, one of the notable properties of the Xception is that it is based on depth-wise separable convolutions instead of normal convolution, which helped the model learn the feature representation in a better way. [8].

3.4 Results

Due to the expensive training time, we did not employ a grid-search to feature representations. The tested parameters were considered not be able to reflect any trend of changes.

3.4.1 n-gram

Algorithm	**uni-gram**	**bi-gram**	**uni-&bi-gram**
Naive Bayes	73%	67%	67%
Perceptron	71%	60%	60%
SVM	76%	69%	69%
Logistic Regression	77%	69%	69%
K-Nearest Neighbor	49%	53%	53%
Multilayer Perceptron	74%	66%	66%
Random Forest	73%	65%	65%

Table 2: Performance of Each Models(maximum feature 500)

Algorithm	**uni-gram**	**bi-gram**	**uni-&bi-gram**
Naive Bayes	75%	68%	68%
Perceptron	74%	65%	65%
SVM	77%	69%	69%
Logistic Regression	78%	70%	70%
K-Nearest Neighbor	49%	52%	52%
Multilayer Perceptron	75%	67%	67%
Random Forest	74%	65%	65%

Table 3: Performance of Each Models(maximum feature 1000)

Algorithm	500 features	1000 features	5000 features	9000 features
Naive Bayes	73%	75%	76%	76%
SVM	76%	77%	76%	76%
Logistic Regression	77%	78%	79%	79%

Table 4: Performance of Models in Difference Feature Space

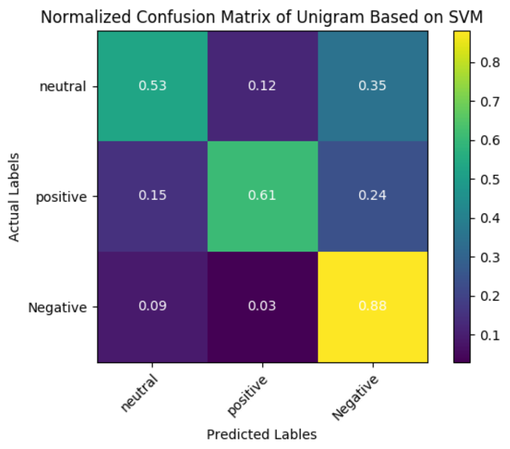


Figure 3: Confusion Matrix of Uni-gram(500 features) Based on SVM

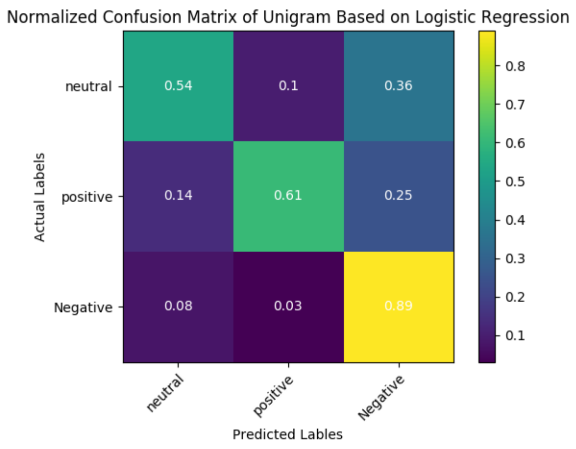


Figure 4: Confusion Matrix of Uni-gram(500 features) Based on Logistic Regression

After we applied the vectorization of corpus, uni-gram, bi-gram or mix of these two are fed into six machine learning algorithms and convolutional neural network. Outputs of each model are shown in Table 2 and Table 3(feature dimension numbers are 500 and 1000 respectively), among all seven models both logistic regression and support vector machine outperform the rest with test accuracy of around 77% when the models' feature is uni-gram. For most machine learning algorithm except $K - NearestNeighbors$, classification performance becomes worse when the feature is either bi-gram or the mix of uni-gram and bi-gram. Besides, it is interesting to see that it is beneficial for $K - NearestNeighbor$ to adopt bi-gram or uni-&bi-gram as its feature because its performance is better though its accuracy is the lowest one. And if we change the number of nearest neighbor $K - NearestNeighbor$, the performance of such classifier does not vary too much.

Furthermore if we compare Table 2 with Table 3 and focus on models *Logistic Regression* and *Support Vector Machine*, there is a small increase of test accuracy which is one percent, giving us a hint that we might be able to boost their performance by enlarging the feature space. Thus in Table 4 comparison of results(achieved by Naive Bayes, Support Vector Machine and Logistic Regression models) is presented. However presumable better performance did not happen as feature number increase, from which we could draw a conclusion that feature dimension is not influential to models' ability of sentiment classification.

Finally we decide to move further to analyze performance of two models(*Support Vector Machine* and *Logistic Regression*) as feature is uni-gram. In Figure 3 and Figure 4, where horizontal coordinate and vertical coordinate stand for predicted labels and actual ones. If we look at figure *Support Vector Machine*'s performance in 3, we can see more mis-classification of label 'Neutral' compared with other two labels('Positive' and 'Negative'). However, SVM is capable of predicting document's sentiment of 'Negative'(88%) while it is less capable for positive-type document. *Logistic Regression* share the approximate same performance as Support Vector Machine, and that is reasonable as both classifiers made almost the same test accuracy.

3.4.2 Word Embedding

The results, for three different choices of fixed word length are shown below in Table5.

From the results, it is clear that Logistic Regression and Multilayer Perceptron perform the best, even for the more limited feature vectors. KNN performs quite poorly, but interestingly improves notably as the feature vector shrinks. It seems plausible that the very large feature vectors caused the curse of dimensionality to set in and make distances mean very little.

Random Forest and SVM were largely stable with differences that cannot be conclusively attributed to the variations in feature vectors. Finally perceptron showed the largest gain from larger feature vectors. though the gain is much smaller from 3000 long vectors to 6000 long vectors compared to from 1500 long to 3000 long.

Algorithm	**20 words**	**10 words**	**5 words**
Naive Bayes	N/A	N/A	N/A
Perceptron	68%	66%	61%
SVM	68%	67%	68%
Logistic Regression	72%	70%	69%
K-Nearest Neighbor	44%	53%	63%
Multilayer Perceptron	73%	72%	68%
Random Forest	66%	67%	67%

Table 5: Word Embeddings, based on word.en.vec.short

3.4.3 Convolutional Neural Network: Xception

Algorithm	**uni-gram**	**bi-gram**	**uni-& bi-gram**
Xception(CNN)	66%	64%	68%

Table 6: Accuracy of Xception(500 features)

The model performs best if the feature is uni-&bi-gram. However compared with Support Vector Machine or Logistic Regression, our Xception model is not really good at classification of text sentiment. Besides, compared with sole uni-gram or bi-gram, there is a slight increase of accuracy if we combine both uni-gram and bi-gram together as feature. However, this result is coherent with Yang and Liu's early work, in which Neural Network's performance was not comparable with SVM[25].

3.4.4 CBoW and skip-gram

Algorithms	**Feature number**		
	100	**200**	**300**
Naive Bayes	N/A	N/A	N/A
Perceptron	59%	60%	59%
SVM	72%	73%	72%
Logistic Regression	72%	72%	72%
K-Nearest Neighbor	65%	67%	65%
Multilayer Perceptron	67%	66%	65%
Random Forest	68%	67%	68%

Table 7: Accuracy achieved by employing CBoW

Algorithms	**Feature number**		
	100	**200**	**300**
Naive Bayes	N/A	N/A	N/A
Perceptron	67%	68%	69%
SVM	73%	73%	73%
Logistic Regression	74%	74%	74%
K-Nearest Neighbor	N/A	N/A	N/A
Multilayer Perceptron	72%	71%	71%
Random Forest	68%	67%	68%

Table 8: Accuracy achieved by employing Skip-gram

In comparasion of CBoW and skip-gram, as shown in Table 7 and Table 8, we can see that skip-gram outperformed CBoW in Perceptron where dimension is 100 and 300 respectively; skip-gram also performed better in Multilayer Percptron where dimension is 100; SVM and Random Forest's performance maintained the same in both models; In this experiment, Table 7 and Table 8 demonstrate that SVM and Logistic Regression have better performance(around 73%) than their counterparties. We also could noticed that the difference between feature numbers in both CBoW and skip-gram did not affect the accuracy that much, as shown in Table 7 and Table 8.

4 Quantitative Evaluation

Our core evaluation method is the Stratified K-fold cross-validation method. For each algorithm, we split the data into K folds, roughly equally sized chunks of the data, where we iterate through the folds, using all folds except one as our training set. Once trained, we test our result on the held-out fold and save the accuracy of the test-fold. Repeating this until it has been done for all the K-folds gives K different accuracy values. Taking the mean of these K accuracy values gives a reliable approximate accuracy value for the algorithm in question.

The usage of a held-out set is critical in attaining good accuracy, as it means the algorithm will be tested on data that it was not trained on, which would mean that in the case of overfitting, the algorithm would attain much higher accuracy than it would on new data.

However simply holding out one part of the data doesn't ensure that the accuracy is good, as it might simply be a fluke of the test-train split. Using K-fold means the test-train split is repeated K times, with no overlap in the test set. The resulting average accuracy is generally a reliable indicator of the true accuracy of the method.

Other automatic measures can be various types of loss-functions or measures of accuracy improvement as the dataset grows, but we elected to restrict our evaluation to only final accuracy after doing Stratified K-fold cross-validation.

5 Qualitative Evaluation

From the results, it appears that a great deal of the accuracy comes from the feature representation. While some of the algorithms are clearly better than the others in general, the feature representation makes a major difference. Most telling is that SVM and Logistic Regression attain accuracies around 77% for the uni-gram feature representation, which is around 5% better than even the best non-uni-gram feature representation, regardless of algorithm.

The question is, why is this? Intuitively it would seem like bi-grams would encode more complex meanings such as negations of "good" or "bad" which have reversed polarity, but bi-grams perform worse for all algorithms except KNN. Word Embeddings do better, particularly with Logistic Regression and Multilayer Perceptron, but nonetheless in general still performs worse.

The answer is almost certainly feature selection. The n-gram based features are simply the most frequent 500 or 1000 words. A great deal of these will be largely irrelevant but highly frequent words, and even once the majority of these almost ubiquitous words are past, a great deal will likely not carry much, if any, sentiment meaning. The majority of nouns and verbs will simply not indicate anything about sentiment polarity. At 500 or 1000 words, it is nonetheless likely that a large number of words that do carry sentiment meaning have been included, and it is likely these that are central to the performance when using uni-grams.

Why then do bi-grams perform less well? Likely because there are many more bi-grams involving irrelevant words at the top of the frequency list. The benefit of bi-grams would be the ability to capture constructions like "not good", but if it takes more than 1000 bi-grams to reach those constructions in the frequency ranking, the benefit simply doesn't materialize with the feature size we chose.

Our Word Embedding representations aren't based on frequency, but instead attempt to capture fixed amounts of each tweet. The likelihood of capturing the relevant words is thus much higher, but equally the amount of irrelevant words is also much higher, and the end result is extremely large feature vectors where only a small subset of the feature space dimensions are relevant, and no ability to distinguish. Even worse, since the feature vectors are based on the actual ordering of words, but the interesting word embeddings are likely the ones from words carrying sentiment meaning, the information will not even be encoded reliably in the same dimensions of the feature vector.

These are obviously significant problems, but to a large extent they might be solvable simply by employing a whitelist to filter features. Such a whitelist would not be trivial to construct by any means, but equally it would not be so onerous as to require automation. At some level, accounting for positional information seems necessary, but that seems perhaps ill-suited to the current models using fixed-size vectors.

Moving beyond the simplest issues of the feature representation, there are also specific concerns that relate to domain-specific features. Tweets are a specific type of document, and tweets relating to airlines are more specific still. While it is obviously desirable to have generalized sentiment analysers be capable of producing good results on any data, expanding the feature selection with domain-specific vocabulary seems like it would naturally improve the outcome. Pang and Lee argued that in multi-class sentiment classification each class may have its own distinct volcabulary[5]. For instance, "mediocre" or "so-so" can be used as signature word to identify a tweet's sentiment.

As an example, the dataset we worked with in this experiment contained emojis. Some of the emojis indicate strong sentiment meaning, such as 😊 and ❤️ represent significant positive feelings, while 😡 is associated with strong negative experience. 🍷 is very often combined with ✈️, which could also indicate a pleasant journey. However, n-gram dropped those emojis where CountVectorizer was applied because they were infrequent enough, or perhaps because they could not be handled by the internals of CountVectorizer. WordEmbeddings and Word2Vec also excluded emojis in the vector representations.

Similarly, proper nouns in some cases encode sentiment meaning, particularly in the case of slurs and nicknames. These, again, were not included. Of course detecting proper nouns encoding sentiment meaning is a complex problem on its own that goes beyond the scope of this paper.

6 Conclusion

In our experiment, among all the algorithms we used, SVM and Logistic Regression had the best performance, and the Multi-layer Perceptron had stable and ideal performance across all feature representations. Furthermore as models' input features were word-embedding, word2vec or bi-gram, they acquired worse performance compared to those with uni-gram features.Besides, although skip-gram performed slightly better than CBoW but was still not comparable with n-gram. Compared to Pang's early work with accuracy from 78% to 82% [3], our best accuracy in n-gram(79% with dimension 5000) is lower. This may due to smaller number of feature dimensions in our experiment. Finally, SVM or Logistic Regression with feature being uni-gram, which output the best accuracies, are both good at predicting negative-type text while less capable dealing with neutral-type text.

Due to limited working hours, we did not test a series of feature numbers. The feature numbers we had chosen in this experiment might not be representative. Considering the small corpus we had, employing a smaller number of vector dimension in word embedding and word2vec might improve the accuracy in the future.

On evaluation and given the cost in time of training a large dataset with a large number of features, using a whitelist to control the size of the feature space seems necessary for future research. Including emojis in the whitelist or features in the future would also be necessary in the future.

7 List of Contribution

Chapters	**Andres**	**Qian**	**Chuang**
Introduction	X		
Previous Studies		X	
Feature Representations	X	X	X
Algorithms	X		X
Results	X	X	X
Quantitative Evaluation	X		
Qualitative Evaluation	X		
Conclusion		X	X
Proofreading	X	X	X
Coding	X	X	X

8 Appendix - Code

`https://github.com/TrunWu/language_processing`

9 Bibliography

References

[1] *Bag of Words representation.* `http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`. [Online; accessed June 10,2018].

[2] J. Barnes, R. Klinger, and S. S. i. Walde. "Assessing State-of-the-Art Sentiment Models on State-of-the-Art Sentiment Datasets". In: *ArXiv e-prints* (Sept. 2017). arXiv: `1709.04219 [cs.CL]`.

[3] Pang Bo, Lee Lilian, and Vaithyanathan S. "Thumbs up? Sentiment Classification using Machine Learning Techniques". In: *EMNLP 2002, pp. 79–86* (2002).

[4] Pang Bo, Lee Lilian, and Vaithyanathan S. "Thumbs up? Sentiment Classification using Machine Learning Techniques". In: *EMNLP 2002, pp.85* (2002).

[5] Pang Bo and Lee Lillian. "Opinion mining and sentiment analysis". In: 2, No 1-2 (2008) 1–135 (2008), p. 17.

[6] Pang Bo and Lee Lillian. "Opinion mining and sentiment analysis". In: 2, No 1-2 (2008) 1–135 (2008), p. 17.

[7] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606* (2016).

[8] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *arXiv preprint* (2016).

[9] *Figure: architecture of CBoW.* `http://nooverfit.com/wp/wp-content/uploads/2016/09/screen-shot-2015-04-12-at-10-58-21-pm.png`. [Online; accessed June 12,2018].

[10] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[11] David D. Lewis. "Naive(Bayes) at forty: The independence assumption in information retrieval". In: (1998), pp. 4–15.

[12] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: 2013 (Jan. 2013).

[13] *n-gram.* `https://en.wikipedia.org/wiki/N-gram`. [Online; accessed June 10,2018].

[14] *Optimizing word2vec in gensim.* `https://rare-technologies.com/word2vec-in-python-part-two-optimizing/`. [Online; accessed June 10,2018].

[15] Peter Russel Stuart; Norvig. In: 1995 (2003).

[16] *sklearn feature_ extraction CountVectorizer Documentation.* http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer. [Online; accessed June 10,2018].

[17] *sklearn K-Nearest Neighbors Documentation.* http://scikit-learn.org/stable/modules/neighbors.html#classification. [Online; accessed June 11,2018].

[18] *sklearn Logistic Regression Documentation.* http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. [Online; accessed June 11,2018].

[19] *sklearn Multi-layer Perceptron Documentation.* http://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Online; accessed June 12,2018].

[20] *sklearn Naive Bayes Documentation.* http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. [Online; accessed June 11,2018].

[21] *sklearn Perceptron Documentation.* http://scikit-learn.org/stable/modules/linear_model.html#perceptron. [Online; accessed June 11,2018].

[22] *sklearn Random Forest Documentation.* http://scikit-learn.org/stable/modules/ensemble.html#forest. [Online; accessed June 11,2018].

[23] *sklearn Support Vector Machine Documentation.* http://scikit-learn.org/stable/modules/svm.html. [Online; accessed June 11,2018].

[24] *Wikipedia: Convolutional neural network.* https://en.wikipedia.org/wiki/Convolutional_neural_network. [Online; accessed June 12,2018].

[25] Yang Yiming and Liu Xin. "A Re-Examination of Text Categorization Methods". In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval pp. 42-49* (1999).