Sentiment Analysis of Internet Reviews using Machine Learning Algorithms

Ömer Gözüaçık[1] and Emin Onur Karakaşlar[1]

Abstract—**Providing a decent customer service has always been a enormous trouble for companies. With the emergence of the Internet, the evaluation of the customer reviews has become a huge challenge too since the data that has been gathered was extremely big. Therefore, all of the comments cannot be labeled via human supervision. Here, we investigated solving this problem via state-of-the-art machine learning methods such as SVM, k-Neigbors etc. Using a clearly labeled dataset from yelp.com, amazon.com and imdb.com, we achieved a peak test accuracy of 80% using different methods and analyzed their differences.**

I. INTRODUCTION

Classification of customer reviews have become an important problem as tremendous of amount of data is being produced each and every second and, it is almost impossible to label each comment by hand. Therefore, an automated method for classifying the customer comment is needed. In our project, we have a dataset of 3000 labeled comments from different websites; amazon.com, yelp.com, imdb.com[5]. Our goal in this project is to successfully classify whether a comment has positive or negative sentiment. For this aim we first pre-processed the data using state-of-art techniques from literature and then applied several well-known machine learning models. In the end, we had more than 80% accuracy classifying the comments using our approach.

II. RELATED WORK

Automated sentiment analysis has always been an important topic in Information Retrival and there are still a lot of projects going on over it. Nowadays, sentiment analysis mainly focuses on how to label sentences or long texts. However, in past, there were more specific works. Some of these researches include finding sentiment of a word [1],subjective expressions [2], subjective sentences [3] and topics [4]. Then these things were used to build an hybrid system to support the document opinion classification [3]. They believed that, the sentiment of topic, word and expression in a document would can influence classification of the whole document a lot. There are multiple ways defined in the literature for sentiment analysis. Most of these works are based on Natural Language Processing [4], Machine Learning [5] and hybrid mixture of them. There are some unsupervised methods as well [6]. In this project, we will be using both NLP and ML together to build up a model. As machine learning models require a vector to work on, converting words and sentences have become an important topic. In recent years, there have been an advancement in representation of words as vectors. Word2Vec, GloVe and FastText is an example to these advancements. Usually, it is very costly to train these models but there are lots of pre-trained vectors published by the developers of these models. Even though vectors for words are found, there is no common agreement upon how to combine them. Most straightforward way is to average the word vectors in a sentence. However, there are some suggestions to combine minimum and maximums of the vectors in a sentence coordinate based [7]. They have found that this method works quite well for small sized texts but we did not apply it in our project.

III. METHODS

There are two main components while dealing with sentiment analysis, first is to process the data so that the features are extracted in a robust manner. Secondly, choosing a proper classification algorithm that will classify each sentence in a correct manner. In that respect, we will explain the pre-processing of the data using tokenization and stemming, and then, state the results of different machine learning methods such as Naive Bayes, SVM and k-Nearest Neighbors etc, and why they reveal such outcomes.

A. Dataset Description

Our dataset consists 3000 reviews from amazon, yelp and imdb each labelled with positive (1) and negative (0) sentiment. The classes are equally distributed and 1000 samples are taken from the each site. According

*This work was not supported by any organization

[1] Computer Engineering Department, Bilkent University, Ankara 06800, Turkey `http://w3.cs.bilkent.edu.tr/tr/`

to datasets description, the reviews that have clear sentiments are chosen from larger datasets. There are reviews having multiple sentences to single word only. In general, there are no typing errors or mis-spellings.

B. Pre-processing Steps

As our data is in raw text, we had to apply some preprocessing to convert it to the vector format. First of all, we tokenized every sentence into words then normalized them by lowercasing and stemming. Then, the data is separated into train(80%) and test(20%) datasets randomly. Afterwards, a bag of words model is formed. In this model, only the frequencies of the tokens are stored while their order and grammar is neglected. We also applied tf-idf weighting as an alternative approach to bag of words while creating word vectors.

1) Tokenization: Tokenization is the process of segmenting sentences into words. For the most cases, words are separated from each other by single spaces if punctuation marks are removed. That's why, we removed every one of them before separating words. Then we used Stanford NLTK's tokenizer[1] which lowercases every word in a sentence then separates them into words by splitting them according to spaces in between them. This approach is risky for words with dashes in between them as they will be considered as single word (E.g: preprocessing - preprocessing). Problems can also occur with words with spelling errors as every one of them will be considered as unique. Then, words that have no significance but common in the text are removed. These words are called stop-words. We used Stanford NLTK's stop-words list which has 153 one of them while doing this filtering process.

2) Stemming: After tokenization, we applied two different state of the art stemming algorithms to our data: Lancaster[3] and Porter[2]. Stemming is the process of reducing word into its root (imaginative, imagination - imagin). In our case, it is important as we are looking for words itself rather then their meaning in the context. This process reduces the number of dimensions a lot and decreases the noise in the data by combining words with same root together.

3) Vectorizing Data: After tokenization, data is separated into train(80%) and test(20%) datasets randomly. Then, train data is converted to Bag of Words and TF-idf. With respect to the vocabulary generated, test is also converted afterwards.

a) Bag of Words Model: BoW model is one of the most simple and effective model in natural language processing. It counts the frequencies of the tokens in a text and neglects their order. We converted train data to BoW first and then converted test data with respect to the vocabulary of the train data. Columns in the model represent words (features) and rows represent sentences. As data is gathered from multiple sources with different jargons, most of the vocabulary is unique and the number of dimensions is quite high. In order to reduce it and eliminate unique words, we ignored words that occur in one document at first but it dropped our accuracy a little bit. This problem will be investigated in further sections in more detail.

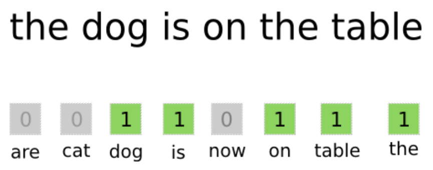

Fig. 1. This Figure illustrates an example of Bag of Words model representation.

b) TF-IDF Model: TF-IDF stands for term frequency-inverse document frequency [4] and it is a very popular method for weighting words in a text. In BoW, the weight of a word in a sample was only determined by its frequency in the sample. However, with TF-IDF, the terms frequency in whole corpus is considered. If a word is abundant in whole corpus, then its weight will be lower.

TF = (# times term i is used in the sample) / (Total # of terms in the sample)
IDF = log(Total # of samples / Total # of samples term i is used)

TF-IDF weight for term i is found by: TF x IDF.
We converted BoW model to TD-IDF using the equations above for every sample.

c) Document Vectors: Word vectors are becoming quite popular with the introduction of neural networks. Traditional models like bag of words and tf-idf are not continuous and looks only for the occurrences of the words. However, word vector representations like Word2vec, GloVe and FastText looks for the whole text continuously while creating a vector. With these approaches, the resulting vectors contain a lot of information about the syntactic and semantic property of text. Most common example given is the king&queen and man&woman. The difference between king&queen and man&woman are similar with these approaches as it can be seen from the figure below [15].

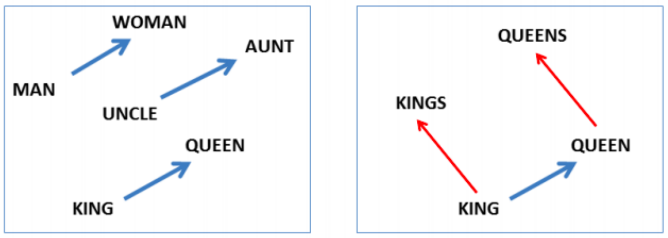

Fig. 2. This Figure illustrates an example of Word2Vec representation.

Even though word vectors are quite useful in terms of word by word similarity, it is still an ongoing research to combine them properly to create vectors for documents. One way for doing this is to average the vectors of the words in a sentence. We used a pre-trained GloVe model which is trained with 6 billion tokens from Wikipedia [14]. Then we averaged the word vectors as explained above to create a vector for every review. If a review did not have any words from the pre-trained vector, it is ignored.

C. Feature Size Reduction

The dimension of the vector that we formed was quite high (around 3500) even though we applied stemming and removed stop words. In order not to suffer from curse of dimensionality, we used PCA at first to reduce the feature size. We also realized that some words are used in only one sample. We removed these words as well and tested. We set a certain limit for words and filtered out the ones that are used less than that limit.

a) PCA: As the dimension of the vector is quite high, we used PCA to reduce its size to 300. regular PCA is costly, we used Randomized PCA which is approximation to that for large datasets [13]. Even though our dataset was small, it was quite useful due to its speed. It is approximation for singular value decomposition. In regular case, all singular vectors are calculated even though user wants to reduce the size to a certain number. In this approach only first N vectors with highest eigenvalues are calculated automatically. It speeds up the process quite a lot however number of components to be reduced need to be set beforehand. It is not possible to look for the eigenvalues and use methods like elbow method to determine the number of components. We set the number of components to be 300 and tested it with SVM, Logistic Regression and KNN.

$$A \approx QQ^T A$$

$$B = Q^T A$$

$$B = S\Sigma V^T$$

$$A \approx QQ^T A = Q(S\Sigma V^T)$$

$$A \approx QQ^T A = Q(S\Sigma V^T)U = QS$$

Randomized PCA comes with the idea that if an orthogonal matrix Q and its transpose is multiplied with a matrix A, the resulting matrix is almost equal to A. Size of the Q matrix is determined same as the number of components that user wants to reduce to (N). The dimension of A is lowered significantly when it is multiplied Q^T and B is formed. The first N vectors of the PCA of A can be found by multiplying Q with the eigenvectors of B found from PCA.

b) Excluding features that are used in N number of samples: When we created the BoW model for the data, we realized that some words only occur in only one sample. As our data is limited in size, there are too many unique words. The cause of these unique words might be due to typing errors as well. We set different limit values (N) starting from 2 to 5 filtered out words that occur less than N number of documents. The number of dimensions decreased a lot.

TABLE I
NUMBER OF FEATURES VS N (MINIMUM NUMBER OF OCCURRENCE)

N	Number of features
1	3716
2	1599
3	1061
4	793
5	634
50	36

D. Used ML Methods

After the pre-processing of the data we have a feature vector which has an approximate size of 3000 different words. This approximation is caused by the randomly splitting the dataset into train/test sets. Although, 3000 is too much for a feature vector (and generally reduced in terms of size), using a feature selection technique will not yield better results since approximately 1500 of the words are unique to the sentence. Also, since the lengths of the sentences differ from each other as shown in Table 1, not using all of the stemmed and tokenized words results in poor accuracy results. Therefore, for all

TABLE II
TWO RANDOM EXAMPLES OF A NEGATIVELY LABELED DATA

Label	Sentence
0	It's uncomfortable and the sound quality is quite poor compared with the phone(Razr) or with my previous wired headset (that plugged into an LG).
0	Was not happy.

machine learning methods that are applied, we used all the features.

Table III and Table IV in appendices summarizes the results for applied methods and their test accuracies. We trained different models for each method with different parameters, the training accuracies on the tables are obtained with the 5-K fold cross-validation method and each model was tuned by changing the models parameters which are described in the following sections. Specifically, Table III shows the results for td-idf and bow methods without using any feature reduction method and Table IV shows the same results with PCA applied to the dataset.

1) Naive Bayes: We used both Bernoulli, Multinomial and Gaussian Naive Bayes to classify our data resulting 81.0%, 81.1% and 65.5% accuracy respectively on overall for different cases. Bernoulli Naive Bayes normalizes word frequencies to binary. For all cases Laplace smoothing with $alpha = 1$ is used. As our data is very sparse, using this method was quite useful for the first two and it yielded good results as well.

2) Logistic Regression: Logistic Regression is used with different inverse regularization strengths (C) for different cases and on average yielded 80% accuracy. As we increased C training accuracy got higher while test accuracy decreased. The model gave best results when $C = 1$. When C is lower than 1 both training and test accuracies dropped.

3) SVM: SVM yielded one of the best results among all the methods that we tried for all the different conditions that are applied. We cross validated the SVM with 5-k fold with C varying from 1 to 10000 and $C = 1000$ yielded the best results. The reason C is very high is that the dimensionality of the dataset that had after pre-processing is high as well. Using 3000 dimensions $C = 1000$ is actually a reasonable number. At its peak SVM resulted in a 81.7% accuracy.

4) k-Nearest Neighbors: For k-NN, we tried with different number of k numbers from 1 to 15. It had a little effect on the test accuracy, yet we found out that $k = 12$ yields the best accuracy results which is around 75% for different cases; using different stemmers, with and without tf-idf. We tried different distance metrics such as Euclidean, Manhattan and Cosine, however, since the feature vector dimension is very high (around 3000), they do not affect the accuracy at all.

IV. DISCUSSION

In the dataset, we have reviews from three different websites, therefore, although they are all comments, their content differ from each other. Hence, it creates a dataset with a wide spectrum of words. Even though it may seem that 3000 observations are high enough, after stemming and tokenization step, dataset reveals more than 3000 features in which approximately 1500 of them are unique. Table I shows that most features are unique to a sample. We can see that only 36 features occur in more than 50 samples. This uniqueness creates additional dimensions which actually cannot be reduced or selected this is exactly why applying PCA was actually not a good idea in the first place.

Among the machine learning methods that we applied SVM and Naive Bayes yielded the best results for different cases with accuracies around 80%. Even though it may seem a little odd that Naive Bayes method, which assumes the independence between features, resulted in such high accuracy, it could be explained by the uniqueness of the dimensions. For the SVM method, the cost parameter affected the results dramatically, again because of the dimension of the dataset. We found out that $500 < C < 1500$ gives the best results for any SVM training with or without applying tf-idf and it can be seen from Table V.

Table I shows the total feature count with respect to minimum occurrences of the words which clearly shows the dramatic decline in the number when the minimum occurrence number was increased. The slight difference between accuracies while using tf-idf or not could be explained by this. Tf-idf is an algorithm that weights the features according to their appearances yet the dataset is small and there are many unique words inside, the BoW and tf-idf does not create a difference between accuracies.

While working on GloVe vectors trained on Wikipedia corpus, we had some problems finding word vectors for some of our features even though it had 400K vocabulary. We ignored them during pre-processing and this resulted some samples to have nothing in them. It could have been better if we had a larger corpus and train our own word vector model with the methods available. Despite the fact that we used a pre-trained vector of 100D, the results were quite surprising. With SVM and Logistic Regression, accuracies are around 76%. If we used a pre-trained vector with higher dimension (At most 300D is available), we might had a higher accuracy. Furthermore, finding document vectors by adding word vectors is not proven to be the best way to combine word vectors. As explained in the related works, it is the most common way however, the results are poor both accuracy wise and training wise. Salvaging words from a 400K vocabulary is quite costly. The other approach on combining word vectors can be tried in the future to see whether it effects positively on the result. There is a new approach to this problem as well. Doc2Vec is released in recent years and it automatically creates document vectors without requiring word vectors. So far, there aren't many resources and support to it but it might become an important aspect in the future. While adding up vectors, we are losing so much information. Finding document vectors directly can be quite useful but even finding word vectors is computationally expensive. Document to vector algorithms are more computationally costly than word vectors and it is not possible to train a model without a pre-trained one being published with the resources that we have right now.

As a conclusion, we can say that 3000 observations should be increased to have a better performance since the real frequency of occurrences could affect the performance dramatically which is now unknown to us. Right now, we only have samples of a small set with unique words from each sites. As each of these site's have a unique corpus, the resulting vocabulary is rich however, it is not distinctive enough to make a classification.

APPENDIX

TABLE III
TRAINING AND TEST ACCURACIES OF DIFFERENT METHODS FOR TFIDF AND BOW

	tfidf		bow	
	test acc	train acc	test acc	train acc
Bernoulli Naïve	0.8100	0.8075	0.8100	0.8075
Multinomial Naïve	0.8117	0.8121	0.8100	0.8054
Gaussian Naïve	0.6533	0.6500	0.6283	0.6325
Logistic Regression	0.8083	0.8054	0.8050	0.8071
SVM	0.8000	0.8066	**0.8117**	0.8037
KNN (euclidian)	0.7566	0.7745	0.6667	0.6679
KNN (cosine)	0.7566	0.7666	0.7516	0.7379

TABLE IV
TRAINING AND TEST ACCURACIES OF DIFFERENT METHODS FOR TFIDF AND BOW VIA PCA

	tfidf		bow	
	test acc	train acc	test acc	train acc
Logistic Regression	0.7867	0.7929	0.7750	0.7854
SVM	0.7767	0.7771	0.7700	0.7821
KNN (euclidian)	0.7033	0.7008	0.6900	0.6875
KNN (cosine)	0.7750	0.7746	0.7150	0.7292

TABLE V
SVM WITH DIFFERENT C VALUES

	BoW		Tf-idf	
C	train	test	train	test
10	0.5895	0.6000	0.4750	0.5062
100	0.7791	0.7933	0.7716	0.7754
500	0.8116	0.8037	0.7895	0.8050
1000	0.8033	0.7983	0.8037	0.8170
3000	0.7783	0.7900	0.8079	0.7900
5000	0.7767	0.7833	0.7983	0.7995
10000	0.7733	0.7800	0.7825	0.7716

TABLE VI
KNN WITH DIFFERENT NUMBER OF NEIGHBORS

	BoW		Tf-idf	
	cosine	euclidean	cosine	euclidean
1	0.7245	0.6800	0.7383	0.7387
3	0.7104	0.6633	0.7267	0.7595
5	0.7245	0.6758	0.7662	0.7650
7	0.7254	0.6750	0.7600	0.7588
9	0.7416	0.6750	0.7704	0.7675
11	0.7258	0.6680	0.7608	0.7591
13	0.7200	0.6575	0.7667	0.7630
15	0.7283	0.6595	0.7754	0.7708

TABLE VII
ML METHODS CROSS VALIDATED WITH GLOVE VECTORS WITH K=5

	Train Accuracy
Logistic Regression	0.7603
SVM	0.7687
KNN (euclidean)	0.6954
KNN (cosine)	0.6886

Contributions

This project is done in a collective way for all sections. Onur worked more on ML methods whereas Omer worked more on pre-processing. The results are analyzed and discussed in depth together.

References

[1] Hatzivassiloglou, V., McKeown, K.R.: Predicting the semantic orientation of adjectives. In: Proc. 8th Conf. on European chapter of the Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1997) 174–181

[2] Kim, S.M., Hovy, E.: Determining the sentiment of opinions. In: Proceedings of the Coling Conference. (2004)

[3] Pang, B., Lee, L.: A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In: Proceedings of the ACL. (2004) 271–278

[4] J. Yi, T. Nasukawa, R.B., Niblack, W.: Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In: 3rd IEEE Conf. on Data Mining (ICDM'03). (2003) 423–434

[5] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In Proc. of the European Conference on Machine Learning (ECML), pages 137– 142.

[6] Peter D. Turney and Michael L. Littman. 2002. Unsupervised learning of semantic orientation from a hundred-billion-word corpus. Technical Report EGB-1094, National Research Council Canada.

[7] C. De Boom, S. Van Canneyt, T. Demeester and B. Dhoedt, ”Representation learning for very short texts using weighted word embedding aggregation”, Pattern Recognition Letters, vol. 80, pp. 150-156, 2016.

[8] ”Natural Language Toolkit — NLTK 3.2.5 documentation”, Nltk.org, 2017. [Online]. Available: http://www.nltk.org/index.html. [Accessed: 23- Dec- 2017].

[9] ”Porter Stemming Algorithm”, Tartarus.org, 2017. [Online]. Available: https://tartarus.org/martin/PorterStemmer/. [Accessed: 23- Dec- 2017].

[10] ”The Stanford Natural Language Processing Group”, Nlp.stanford.edu, 2017. [Online]. Available: https://nlp.stanford.edu/software/CRF-NER.shtml. [Accessed: 23- Dec- 2017].

[11] ”Tf-idf : Information Retrieval and Text Mining”, Tfidf.com, 2017. [Online]. Available: http://www.tfidf.com/. [Accessed: 23- Dec- 2017].

[12] ”UCI Machine Learning Repository: Sentiment Labelled Sentences Data Set”, Archive.ics.uci.edu, 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences. [Accessed: 23- Dec- 2017].

[13] ”Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, Mark Tygert. ”An algorithm for the principal component”

[14] Pennington, ”GloVe: Global Vectors for Word Representation”, Nlp.stanford.edu, 2018. [Online]. Available: https://nlp.stanford.edu/projects/glove/. [Accessed: 04- Jan- 2018].

[15] ”NIPS-DeepLearningWorkshop-NNforText.pdf”, Google Docs, 2018. [Online]. Available: https://docs.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edi [Accessed: 04- Jan- 2018].