

Opinion Mining of Customer Reviews Using Deep Learning

Ghodratollah Aalipour
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
ga5481@rit.edu

Abstract—Natural language processing (NLP), as a branch of artificial intelligence (AI), has found many important applications in human daily life. In particular, it has improved the ways that computers and humans interact. In this report, we investigate some connections between NLP and business by analyzing a dataset of 1525 negative reviews given by customers of low-rated McDonald's restaurants in the United States. The reviews are labeled by one or several labels from eight categories. We apply word embedding and two deep learning architectures to build a model to assign the best set of labels to each review. At the end, we evaluate our model performance by considering its accuracy and Hamming loss.

Index Terms—Word Embedding; Multi-label Classification; Deep Learning

I. INTRODUCTION

Human language, an old and rich form of communication carries a lot of information that can be investigated in several aspects. Extracting, modeling and exploiting this significant amount of information has made NLP a vital bridge between human linguistic information and digital data. But there are subtleties in this path. For instance, the ambiguity and the changing shape of the data are two main challenges for NLP. Ambiguity is a consequence of the several meaning a word can have and so it is difficult to identify the targeted meaning without having enough knowledge. The changing shape is due to several factors including new technologies and tools. The new technologies bring new types of structures and shortened phrases that become popular among users. For instance, with the rise of Twitter, new words and phrases started appearing in our formal languages. Thus, we have to update our NLP models over time.

Under the light of recent advances in the power of computing, having a good data for feeding a machine learning model plays a critical role in this trajectory. The first step towards analyzing text data would be feature generation from the text. Having a good set of features is one of the most important machine learning challenges.

A. Contribution of NLP to Business

NLP is used to let computers understand human language, extract information, answer questions or complete other tasks. This interaction between computers and human beings is categorized based on the task that needs to be accomplished. NLP has been found very useful in business for interaction with customers in a large scale. In order to better understand customer preferences, many companies now analyze customer call recordings and reviews. Some real-world contributions of NLP to business are as follows:

- Getting feedback from customers through their opinions and reviews
- Detect customers who like/hate company’s products and tune the recommendation systems accordingly
- Improve products receiving most complaints from customers
- Building strong relationships with the customers, vendors and suppliers
- Increase sales by tuning the products to the customer preferences

B. Problem Statement

In this project, we will process customer’s negative reviews of low rated McDonald’s restaurants in the United States. This dataset consists of 1525 negative reviews. Based on each review, a set of labels is assigned, determining which business policies of the company are violated. The reviews are labeled by one or a combination of the following categories:

- BadFood
- Cost
- Filthy
- MissingFood
- OrderProblem
- RudeService
- ScaryMcDs
- SlowService

Two examples of the reviews are presented below:
• “Terrible customer service. I came in at 9:30 pm and stood in front of the register and no one bothered to say anything or help me for 5 minutes. There was no one else waiting for their food inside either; just outside at the window. I left and went to Chick-fil-a next door and was greeted before I was all the way inside. This McDonald’s is also dirty, the floor was covered with dropped food. Obviously filled with surly and unhappy workers.”
• “I’m not crazy about this McDonald’s. This is primarily because they are so slow. My gosh what exactly is the hold up? It’s FAST food people. Also, this morning, I guess the worker thought his mic was off; but it wasn’t. I now know that he is trying to get as many hours as possible because he needs money BAD. Spread the word. Anyway, this location is on a little access road and you have to go back the way you came because there is no exit from it at the other end. It would have helped if there was one. So, in the end I think I’ll avoid this location and find another. This should be easy as there is no shortage of Mickey D’s in this piece.”

Objective: Design a model to take a (negative) review and determine the most appropriate set of labels. This is a linguistic multi-label classification problem.

II. RELATED WORK

There has been a great deal of attention to the applications of natural language processing (NLP) to business companies for analyzing contents generated by their users. One of popular forms of opinion mining is aspect extraction. It is a type of sentimental analysis that identifies the specific aspects of a product or service that the review is addressing. Poria et al. [9] employ a deep learning approach towards aspect extraction. They apply a shallow convolutional neural network that labels each token in the reviews as either aspect or non-aspect. They also handcraft several features and feed them to the network for better accuracy. The result of their deep learning approach yields remarkable improvement over other previously studied methods.

Zhou et al. [10] study online hotel reviews to provide comparative insights about customer satisfaction. The selected hotels are in Hangzhou, one of the growing tourism cities in China. The reviews are extracted from Agoda.com and the investigated hotels are four or five-star hotels. Their approach exploits techniques for feature selection and provides measures to select most influencing aspects on customer satisfaction.

Change et al. [1] analyze and extract information from user-generated data regarding the consumer opinions, ratings, and recommendations. They propose an integrated framework which combines several ideas including data visualization, aspect extraction, and category detection to gain insights into hotel ratings and reviews. They show that their methods does a better job in terms of precision (0.95) and recall (0.96) than the well-known sentiment classification methods. The visual analytics results reveal that business travelers tend to give lower ratings, while couples tend to give higher ratings. They discover that people usually tend to rate lowest in July and highest in December.

One of the deep learning approaches towards multi-label text classification for customer reviews is made by Chen et al. [2]. They propose an ensemble-based approach applying convolutional and recurrent neural networks to capture both the global and the local textual semantics and to model high-order label correlations. They benchmark their approach with some baselines using two publicly available datasets, i.e., Reuters-21578 (21,578 documents) and RCV1-v2 (800,000 manually categorized newswire stories) which are relatively nice and large datasets in compare to our dataset.

By the rise of large data and GPU computing techniques, the attention to the deep learning approaches has increased. Extreme multi-label text classification (XMTC) is the problem of assigning the most relevant subset of an extremely large label collection. The label size in an XMTC is typically ranging between hundreds to thousands to millions. The huge label space induces several sparsity and scalability challenges. To address these issues, significant development of new machine learning methods has been made. Liu et al. [7], propose the first deep learning approach for XMTC using a family of new convolutional neural network (CNN) models which are tailored for multi-label classification.

III. BACKGROUND

A. What is Multi-label Classification?

In machine learning, the multi-label classification is a variant of the classification problem where each data instance may be assigned with several labels. Multi-label classification is a generalization of multi-class classification, which is the single-label problem of associating exactly one label to each instance. In the multi-label problem, there is no limit on the number of labels a data instance may receive.

Formally, multi-label classification is the problem of finding a function that maps every input to a binary vector, which is known as one-hot vector. In the one-hot vector, we have exactly one bit of “1” and the remaining bits are “0”. This unique bit of one represents the label of our interest. In the multi-label classification, each set of labels can be represented by a binary vector which is not restricted to have exactly one non-zero bit. It could have as many one bits as the length of the original label. So, a multi-label classification corresponds to finding a function which determines the closest binary vector to the original label.

Multi-label classification has diverse range of applications from bioinformatics where each protein is labeled by multiple functionality such as metabolism and energy to other areas such as video annotation, or movie genre detection. The problem that we study in this project covers another application of multi-label classification which is assigning to each review text, a set of predicted labels.

B. Methods for Multi-Label Classification

There are several approaches to solve a multi-label classification problem but in general, these approaches can be divided
into two categories: I) problem transformation methods, and II) algorithm adaptation methods. The problem transformation methods intend to transform multi-label classification tasks into one or more single-label classification problems. The algorithm adaptation methods extend traditional classifiers to handle multi-label problems directly.

C. Problem Transformation

This method transforms the original problem to single-label classification problem(s) and trains a classical classifier for each problem. This method can be performed in several approaches as follow:

Binary Relevance is the simplest technique to solve a multi-label classification problem. This technique simply treats each label as a separate single binary classification. Thus, if we have \( k \) labels, we build \( k \) classifiers \( f_1, \ldots, f_k \). Then to assign a set of labels to a new data instance, we feed it to each classifier to see what it generates. If the \( i \)-th classifier \( f_i \), for \( 1 \leq i \leq k \), generates 1, then we include the \( i \)-th label among our prediction otherwise we skip the \( i \)-th label. We use this technique as our benchmark model. We use Support Vector Machines (SVM), Logistic Regression, and Decision Trees (DT) as two other candidates for the binary classifiers. Each time we apply the same family of models to all labels.

Another approach would be applying appropriate classifiers to each label according to the distribution of 0’s and 1’s in the \( k \)-th label. Some classifiers tend to adjust to imbalanced datasets. In our dataset, two labels “RudeService” and “SlowService” have high frequencies. As a result, in the corresponding binary vector, the bit 1 has high frequency. On the other hand, the label “Cost has low frequency, resulting a very sparse binary vector for this label.

There are two other methods for attacking a multi-label classification problem. One of them is building Classifier Chains and the second method is the Label Powerset method. In the former technique, we build the first classifier to train only on the input data and then the next classifier is trained on the input space extended by adding the previous binary label to the feature sets. Thus if the number of original features in the data is \( m \), then in the first step, we build our classifier on these \( m \) features, in the second step we add the binary vector of the label \( l_1 \) to the feature set and make a new classifier on these augmented \( m + 1 \) features. We iterate this procedure \( k \) times so that the final model is trained on \( n + k - 1 \) features. This approach include the concurrences among the labels in a backward form. But it does not count for group dependencies.

In the Label Powerset approach, we treat each \( 2^k \) subsets of labels as a unique single label and then transform our multi-label problem into a multi-class problem with a single multi-class classifier on all of these unique labels. The number of labels for a single classifier is exponential in the number of the original labels. In particular, if we don’t have uniform distribution on the combinations of the original labels, then this approach won’t attain good accuracy as it will ultimately misses the combinations with low frequencies which is the case for our dataset in this project.

D. Algorithm Adaption

In this technique, we adjust our algorithm to directly conduct multi-label classification. There are several extensions of the traditional classifiers. For instance, the multi-label extensions of Support Vector Machine (MSVM), Decision Tree (MDT), kNN (MLkNN) or other learning algorithms. Our proposed method is based on the adaption of neural networks to multi-label problems.

E. Ensemble Methods

The ensemble method, as another approach for multi-label classification, is a supervised learning algorithm in which an agent forms a committee of classifiers and then combines their outputs to make a prediction. The member of the committee are called the base-level ones. In recent years, ensemble techniques have been receiving much attention in machine learning, because they are well-known for handling the over-fitting problems which decrease the accuracy of the models particularly in highly unbalanced datasets [7]. These approaches are either homogeneous or heterogeneous, where in the homogeneous approaches, the base-level classifiers are constructed using the same algorithms, whereas in the heterogeneous approaches, the base-level classifiers are constructed using various algorithms to improve performance.

IV. EVALUATION METRICS

Evaluation of single-label classification is different with multi-label classification. In single-label classification, the predictions made by the models are either true or false but in multi-label classification the prediction can be partially true. This is the case when the trained classifier correctly generates a set of labels where at least one of the labels but not all, belongs to the ground truth label set. In addition, a classifier may also assign to a data instance one or more labels that should not have been generated. In this sense, the evaluation of multi-label classifiers needs different tools than those used in single-label methods and they should generalize the previous tools for the single-label classification. Some of the proposed approaches in the literature can be categorized in three classes: evaluating partition, evaluating ranking, and using label hierarchy. The partition based measures, namely the example-based-measures, evaluate the quality of the classification into the available labels over all examples of the evaluation data set. The ranking-based approach checks if the labels are ranked in order relevance. The hierarchy -based approach measures how well the system is preserving the existing hierarchical structure among the labels. In this project, five measures are selected from the evaluation partition approach.

As we noted earlier, we are addressing the notion of partially correct. One strategy is to find the average differences between the predicted labels and the actual labels for each test example. This approach is called example-based approach and addresses the existing correlations among the labels. Suppose that we have \( n \) samples, \( k \) labels, \( z_i \) is the predicted multi-label and \( y_i \) is the ground truth, then we can consider the following metrics:
A. Exact Match Ratio

One trivial way to take the notion of “partially correct” into the account is simply ignoring partially-correct answers and only focus on the fully correct predictions. This can be captured by the Exact Match Ratio defined by

\[
\text{ExactMatchRatio} = \frac{1}{n} \sum_{i=1}^{n} I(y_i, z_i)
\]

where \( I \) is the Boolean indicator function, checks whether \( y_i \) and \( z_i \) match or not.

The above metric does not credit partial matches. The following metrics count partial matches.

B. Hamming Loss

The Hamming Loss counts on average for the portion of the wrong labels to the total number of labels. It takes into the account both prediction error (the labels that should not have been presented) and the missing error (a correct label that should have been predicted). It is defined by

\[
\text{HammingLoss (HL)} = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{Hamming}(y_i, z_i)}{k},
\]

where \( \text{Hamming}(U, V) \) is the Hamming distance between two vectored \( U \) and \( V \) and is the number of bits they disagree. Ideally, we expect to have \( HL = 0 \), thus a smaller Hamming Loss gives a better model. We also consider the following measures, which concerns with positive-answer rates, negative-answer rates.

C. Accuracy, Precision, Recall and \( F_1 \)

\text{Accuracy} is the average proportion of correctly predicted labels to the total number of labels (both predicted and actual).

\[
\text{Accuracy} = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i \cap z_i|}{|y_i \cup z_i|}.
\]

The \text{Precision} is the average proportion of correctly predicted labels to the total number of predicted labels.

\[
\text{Precision} = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i \cap z_i|}{|z_i|}.
\]

The \text{Recall} is the average proportion of correctly predicted labels to the total number of actual labels.

\[
\text{Recall} = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i \cap z_i|}{|y_i|}.
\]

The definitions for Precision and Recall lead to the following definition, known as \( F_1 \)-Score as the harmonic mean of Precision and Recall.

\[
F_1 = \frac{1}{n} \sum_{i=1}^{n} \frac{2|y_i \cap z_i|}{|y_i| + |z_i|}
\]

Just similar to the binary classification, the higher value for Accuracy, Precision, Recall, and \( F_1 \)-score, the better performance the learning model has.

V. Data Transformation – Texts to Numbers

In building machine learning models, understanding the data, cleaning, processing and removing inconsistency are the main and primary steps. The data engineer should understand the data from different aspects such as dependencies among the variables, the correlations between each and the target variable, how the model works and what label(s) would have the most misclassification.

There are some basic and intuitive approaches for feature generation. The presence of some words such as nice, happy, fun, etc in a positive/negative classification problem would be the baseline for the feature generation. But a good set of features does not limit to these features only. We still can find more complicated features such as part-of-speech (POS) tagging, relation identification, etc. Also statistical features such as frequency and likelihood can be applied to add to the early set of features. The author of this report found feature extraction as one of the most important parts of NLP.

A. Feature Extraction

The next step toward building a model is feature extraction. Features facilitate transformation of the text into continuous values. One of the main problems in machine learning is the number of features that are involved in the analysis. The analysis with a large number of variables generally requires powerful infrastructures for computation. It may also make the prediction model suffer from overfitting to training samples and generalizing poorly to new samples. Feature extraction is a general term for methods of designing the variables to address these problems while still preserving the information with sufficient accuracy. Many machine learning practitioners believe that properly optimized features form a significant part of effective model construction.

There are some linguistically motivated methods for feature extraction. Presence of some words like tasty, nice, friendly, garbage, disappointing, and dirty can give us some sense about the general sense of the review. The dependency relation between the concepts in the text can also be useful in this regard. For example the presence of some words like “noisy”, “crowdy”, “downtown” may direct us to conclude that the customer is complaining about the location. Comprehending the text is also another approach. Detection of the set of features and making connection between the phrases would have a great impact on designing a classifier.

N-gram Models: The primary methods for feature selection is making a bag of words. For each category of interesting words, we find the top frequent words and put them in our bag. After lemmatizing words, we filter over the noun, verbs, adjective, and adverbs to find the top frequent ones. By selecting a specific number of each, we build a binary dataset representing the existence of each of these words. We also include bigrams and trigrams. It remains as a problem for cross-validation dataset to determine which portion of each category should be included in our bag of words. To illustrate the features for our model, we include the top 20 bigrams and also the top 10 trigrams in the tables below:
POS Tagging: As we can see from the tables above, the top bigrams and trigrams are somewhat unrelated to the reviews. Some less frequent bigrams and trigrams are more informative than the top 20 unigrams. This motivates us to extract more features. A careful look at the labels such as “Filthy”, “RudeService” encourages us to consider POS tags such as nouns, adjectives and adverbs. This is a part of our earlier discussion that some features are linguistically informative. We use the pos_tag built-in function of nltk to extract these POSs.

### Bigram Freq

<table>
<thead>
<tr>
<th>Bigram</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>('in', 'the')</td>
<td>431</td>
</tr>
<tr>
<td>('to', 'the')</td>
<td>344</td>
</tr>
<tr>
<td>('of', 'the')</td>
<td>308</td>
</tr>
<tr>
<td>('and', 'the')</td>
<td>248</td>
</tr>
<tr>
<td>('it', 'was')</td>
<td>236</td>
</tr>
<tr>
<td>('i', 'was')</td>
<td>236</td>
</tr>
<tr>
<td>('drive', 'thru')</td>
<td>228</td>
</tr>
<tr>
<td>('the', 'drive')</td>
<td>227</td>
</tr>
<tr>
<td>('i', 'have')</td>
<td>225</td>
</tr>
<tr>
<td>('at', 'the')</td>
<td>220</td>
</tr>
</tbody>
</table>

### Trigram Freq

<table>
<thead>
<tr>
<th>Trigram</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>('the', 'drive', 'thru')</td>
<td>150</td>
</tr>
<tr>
<td>('in', 'front', 'of')</td>
<td>59</td>
</tr>
<tr>
<td>('one', 'of', 'the')</td>
<td>59</td>
</tr>
<tr>
<td>('the', 'drive', 'through')</td>
<td>57</td>
</tr>
<tr>
<td>('i', 'had', 'to')</td>
<td>48</td>
</tr>
<tr>
<td>('in', 'the', 'drive')</td>
<td>48</td>
</tr>
<tr>
<td>('to', 'this', 'mcdonalds')</td>
<td>43</td>
</tr>
<tr>
<td>('up', 'to', 'the')</td>
<td>42</td>
</tr>
<tr>
<td>('this', 'mcdonalds', 'is')</td>
<td>42</td>
</tr>
<tr>
<td>('to', 'the', 'window')</td>
<td>41</td>
</tr>
</tbody>
</table>

**POS Tagging:**

<table>
<thead>
<tr>
<th>POS = “NN”</th>
<th>Freq</th>
<th>POS = “VB”</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>'order'</td>
<td>731</td>
<td>'be'</td>
<td>4998</td>
</tr>
<tr>
<td>'food'</td>
<td>657</td>
<td>'have'</td>
<td>1465</td>
</tr>
<tr>
<td>'time'</td>
<td>448</td>
<td>'get'</td>
<td>973</td>
</tr>
<tr>
<td>'drive'</td>
<td>413</td>
<td>'go'</td>
<td>820</td>
</tr>
<tr>
<td>'service'</td>
<td>404</td>
<td>'do'</td>
<td>517</td>
</tr>
<tr>
<td>'place'</td>
<td>351</td>
<td>'take'</td>
<td>388</td>
</tr>
<tr>
<td>'don't'</td>
<td>334</td>
<td>'say'</td>
<td>358</td>
</tr>
<tr>
<td>'people'</td>
<td>289</td>
<td>'give'</td>
<td>344</td>
</tr>
<tr>
<td>'location'</td>
<td>279</td>
<td>'come'</td>
<td>305</td>
</tr>
</tbody>
</table>

**B. Word Embedding**

In the previous methods, we assigned a vector representation to each document using a one-hot encoded approach where 1 stands for the position where the word exists and 0 everywhere else. The vector representation of words in this format is binary and is based on frequencies. There is another approach to assign vectors to words which is prediction based embedding. It preserves the lexical relations between words.

Vector space models (VSMs) assign continuous vectors to the words where semantically similar words are mapped to nearby points. VSMs have a rich history in NLP, but all methods rely on the fact that the words that appear in the same contexts share semantic meaning. The different approaches employing this fact can be categorized into two groups: count-based methods such as Latent Semantic Analysis (LDA) and predictive methods such as neural probabilistic language models. Count-based methods are based on the statistical analysis of how often some word co-occurs with its neighbor words in a large text, and then map these count-statistics down to a small, dense vector for each word. But the predictive models predict a word from its neighbors in terms of small and dense embedding vectors.

Mikolov et al. [8] in 2013 introduced word2vec to the NLP community as the first prediction based model in the sense that they provided probabilities to the words. They could achieve results like King-Man + Woman = Queen, which is remarkable in preserving lexical relations between words. This relation says that if we find the vector representation for “King” and subtract the vector representation of “Man” from it and add the result to the vector representation for “Woman”, the resulting vector is very close to the vector representation for Queen. Moreover, word vectors capture many other linguistic regularities, for example if \( V_p, V_f, V_i \), denote vectors for Paris, France and Italy, respectively, then \( V_p - V_f + V_i \) results in a vector that is very close to the vector for Rome. In the next steps, we will use word2vec as main method for feature extraction.

There are two models for word2vec, the continuous bag of words (CBOW) and the skip-gram model, which both are shallow neural networks. Both of these techniques learn weights which act as word vector representations. Both are neural networks that have just one single hidden layer and they are trained to perform a specific task. This specific task and this hidden layer is the difference between the skip-gram model and the CBOW model. In the skip-gram model, we train the neural network to predict the probability for each words in our vocabulary to be the nearby word to a given input word. So, in the skip-gram architecture, the model predicts the surrounding window of the current word. But in CBOW, the objective is to maximize the conditional probability of the output word given the input context. Thus, it predicts the current word from a window of surrounding context words. After training these the network, we won’t use their outputs, instead we use their connection weight matrix as our word embedding. These two models are depicted in the Figure 1, shown by Mikolov et al. [8]. Though the skip-gram is slower, it performs better for infrequent words and large datasets. We use a pre-trained word2vec model, trained by Google that generates 300 dimensional continuous vector for each word. It is available in the package Gensim of Python and we can easily call it from there. It carries the word vectors for a vocabulary of 3 million words trained on around 100 billion words from the Google news dataset.

It is hard to visualize a 300-dimensional vector, but by utilizing dimension reduction techniques, we can project each vector down to the plane (2 dimensions). There are several methods for dimension reduction. One of them is the t-
C. From Word Embedding to Document Vectors

A word embedding generates a vector \( U \) in the \( n \)-dimensional vector space over real numbers, one vector per each word in the document. We can exploit these vectors for the words to represent a review through a new vector representation. In particular, since the semantic similarity is preserved under word embedding, utilizing the word embedding can keep such relations in the feature vector representing our document. For generating a vector for a review, we apply the following methods. The first approach is based on the average vector for all words in the reviews. So, we simply take the average of all \( 300 \)-dimensional vectors of all words in the document. This results in a stable length (\( 300 \)-dimensional) vector representation for the reviews, no matter how large the review is. Moreover, since the semantic relations are preserved in the word embedding, we expect that the average vector conveys the average concept that the keywords within this document carry. Furthermore, this fixed-length input vector can be fed to a feed-forwarding neural network for the modeling task. Actually, we have applied this method as one of our proposed method for multi-label classification by modifying a neural network architecture that can capture correlations between the labels and can predict several labels simultaneously.

The second approach for vector representation of a document is concatenating all words in the document. For a document with \( n \) words where each is represented by a \( 300 \)-dimensional vector, this approach would generate a \( 300n \)-dimensional vector. This variable length input data cannot be fed to any traditional classifier unless all inputs are mapped to the same length vectors. One new architecture of neural networks that has recently received much attention because of its performance for sequential data is the LSTM networks. They are able to handle variable length input and they can capture long distant dependencies between words and phrases. We will explain this approach and the feed-forwarding network in the next section.

There are other recent approaches that are becoming popular. The most famous approach for representing a document with a vector is doc2vec (sometimes known as paragraph2vec) proposed by Le and Mikolov in [6]. This technique requires training on a large data corpus. The vectors generated by doc2vec can be used for tasks like finding similarity between the documents. Just like word2vec, it is also available in Gensim. For other relevant approaches, we refer the interested reader to [3]. Since we want to take the advantage of related distant words in a review, we ignore the doc2vec approach and instead stick to the aforementioned average vector and concatenation vector to use a pre-trained word embedding. After generating such continuous values for each review, we can apply a classifier to the generated vectors for the review. One choice of classifiers that can learn the correlations between the labels for a multi-label classification purpose is deep learning models.

VI. THE PROPOSED METHOD

As we discussed earlier, we are looking for a model that can predict the best set of labels for an unseen data instance. The notion of “best” is determined by the evaluation metrics that we discussed in the previous section. To design this model, instead of predicting a binary vector, we predict a continuous-valued vector whose entries are in the interval [0, 1]. Then we apply a threshold on these values to discretize them to a binary vector. Thus, we have two challenges involved in this approach. One is how we can generate such a continuous vector and second, which threshold to choose after generating the continuous values. The solution to the former problem is

Distributed Stochastic Neighbor Embedding (t-SNE) as a powerful technique for dimension reduction that is particularly appropriate for the visualization of high-dimensional data. This technique models each high-dimensional data point by a two- or three-dimensional point such that the close points are mapped to nearby points and dissimilar points are mapped to distant points. By using the t-SNE dimension reduction technique, we can investigate these embedded vectors.

Through the Figure 2, available in tensorflow.com, it becomes clear that these vectors capture useful semantic information about words and their relations. It is interesting to see the directions in the induced vector space follow certain semantic relationships, e.g. male-female, verb tense and even country-capital relationships between words, as illustrated, see [8].

Now, we have a word representation for each context word. But our input data is the user review. We should find a method to change each review as a sequence of words into a vector so that we can feed it to a classifier. In the next section, we discuss some approaches to go from word vectors to documents vectors.
through adapting a neural network approach and the solution to the latter problem is through a Brute Force search (grid search) for all possible thresholds.

Here we present the formal solution methods. Let $X = \{X_1, \ldots, X_n\}$ denote the set of input data instances and $L = \{l_1, \ldots, l_k\}$ denote the set of labels. Thus, we can represent the pair of instances and their corresponding labels as $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, where $Y_i \subseteq L$ is the set of labels assigned to the instance $X_i$, for $1 \leq i \leq n$. We denote the set of these multi-labels by $Y$. We are aiming to build a classifier model $H : X \rightarrow \{0, 1\}^L$ such that $H$ minimizes the hamming Loss between the actual labels and the generated labels. Hence, $H$ is a solution to the following optimization problem:

$$\min_{H : X \rightarrow \{0, 1\}^L} \text{HammingLoss}(H(X), Y)$$

Since the Hamming Loss is the measurement for the average distance between two binary vectors, we relax this problem by dropping the notion of “discrete” and letting the problem to search for a wider range of vectors such as continuous vectors. After that, we will apply a threshold on the continuous values to get a binary vector:

$$\min_{h : X \rightarrow [0, 1]} \sum_{i=1}^{n} ||h(X_i) - Y_i||_2^2$$

This relaxation has several significant advantages. First, this approach converts the search space of first optimization problem which has the size $2^k$, to a $k$-dimensional search space over the real numbers in $[0, 1]$. Consequently, it permits applying the powerful optimization methods leveraging derivatives of the functions to reach the global optimum. Second, there are advanced approaches such as neural networks to form this model as the solutions to an optimization problem by applying gradients to get the global optima. Third, if $h_0$ is the answer to the relaxed problem, then we expect the entries of each $h_0(X_i)$ to be close to zero or close to one. Thus, the choice of the threshold is irrelevant here.

A. Neural Network Adaption

Human’s breakthrough achievements such as planes are inspiration induced from the nature. Recently, another big achievements on artificial intelligence was made by artificial neural networks (ANN). Though birds inspired humans to fly, planes are not the identical copy of the bird’s wings. Similarly, ANNs have become different from their biological prototype. Some researchers even argue that we should stick to our own-human terminology e.g., by saying “units” rather than “neurons”. ANNs are at the very core of Deep Learning. They are versatile, and scalable, which make them great candidates for large and highly complex Machine Learning tasks, such as classifying billions of images (e.g., Google Images), powering speech recognition services (e.g., Apple’s Siri), recommending the best products to buy (Amazon), or learning to beat the world champion at the game of Go (DeepMind’s AlphaGo).

ANNs have been around since they were first introduced in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts. In their breakthrough paper, “A Logical Calculus of Ideas Immanent in Nervous Activity,” McCulloch and Pitts proposed that a computationally simulated model for how biological neurons might be working for performing complicated tasks. This was the first artificial neural network architecture and after that many other architectures have been studied.

We leverage the great power of ANN in our project. More specifically, we design an ANN to find the model for generation of continuous-valued vectors. There are two types of neural networks that we will apply towards our solution. The first network is a feed-forwarding neural network, the most widely used networks for regression/classification task.

In the Figure 3, we depict a simple neuron for a feed-forwarding neural network. This node can be any hidden node in any layer. This figure represents how a single perceptron, known as a logistic regression, is working. This node receives inputs with some weights associated to them. Then take the weighted summations of its inputs and feed them to the sigmoid function $\sigma(x) = \frac{1}{1 + e^{-x}}$. This value along with the corresponding weight would be one of the inputs for the nodes in the next layers. Instead of this the sigmoid function, there are other well-known functions that have better performance than the sigmoid function. One of these function is the Rectified Linear Unit (ReLU), defined by $ReLU(x) = \max\{0, x\}$. For our architecture, we use the sigmoid function because it always generates values in the interval $[0, 1]$. These values will be binarized into either 0 or 1 through a threshold. This function that lies in each neuron (node) is called the activation function.

These neurons can be stacked together to form a layer. By further stacking the layers, we form a network. A Deep Neural Network (DNN) is any network with more than three layers. In a feed-forwarding style, each neuron in one layer has directed connections to the neurons of the next layer but has no connection with the other neurons withing the same layer. This architecture of neural networks is called a Multi Layer Perceptron (MLP). In Figure 4, we have depicted an MLP with three layers, one output layer and two hidden layers. The hidden layers have green nodes whereas the output layer is the most-right layer in the network. The connection weights $w_{ij}^l$ is the weight for the connection between $i$-th node of the $l$-th layer with the $j$-th node in the previous layer (layer $(l - 1)$-
th). For the sake of less-crowded picture, we have dropped the weights on the connections and the sigmoid function on each node from the Figure 4. The MLP’s are very powerful in estimating any continuous function.

The Universal Approximation Theorem states that every continuous function can be approximated by a multi-layer perceptron with just one hidden layer. This result holds for a wide range of activation functions. A multi-layer network learns to choose the connection weights \( w_{ij} \)'s such that the loss function (usually the sum of squared errors) decreases as much as possible. Thus training a network means solving an optimization problem whose variables are the connection weights \( w_{ij} \). Now, it is clear that why training a large network takes long.

There are a variety of learning techniques to train a network. The most popular one is the back-propagation where the output values are compared to the actual answers to get the error. By various techniques, the error flows into the network. Using this information, the algorithm adjusts the connection weights to reduce the value of the error function. This procedure is iterated over and over again until the network converges to some states where the error of the calculations does not change or is insignificant.

The update rule is induced by a powerful method for non-linear optimization that is called Gradient Descent. In this method, the network calculates the derivative of the error function with respect to the network weights, and changes the weights through the direction of the gradient vector. This direction makes going downhill on the surface of the error function. That's why we choose our networks with differentiable activation functions so that we can apply the back-propagation algorithm. There are other hyper-parameter fine tuning that we discuss some of them and skip the rest.

One of the important features of a deep learning model is the optimization technique for updating the connection weights of the network. Though the Gradient Descent is a good technique, it suffers from several issues. The first and the most important issue is that the gradient can be trapped in numerous suboptimal local minima. Moreover, selecting a proper learning rate can be difficult because a small rate would imply slow convergence while a large rate hinders the convergence and causes the loss function to fluctuate around the minimum or even to diverge. To address these issues, Kingma and Ba [5] proposed a powerful method known as Adaptive Moment Estimation (Adam). This technique applies adaptive learning rates. Just like momentum optimization, Adam keeps track of an exponentially decaying average of past squared gradients, as well as exponentially decaying average of past squared gradients. Moreover, Adam avoids getting stuck in the local minima. In addition, it has much higher convergence speed than the traditional Gradient Descent. We use this method as our optimizer algorithm. The network that we applied is depicted in the Figure 4.

B. Feed-Forwarding Network Parameters

Our network has three layers consists of two hidden layers and one output layer that predicts eight continuous values as a vector of length eight whose elements are in the interval \([0, 1]\). The input data is the vector representation of the reviews using the average vector approach discussed in the previous section. Since our embedding dimension is 300, the average vector of all words in each review is still a 300-dimensional vector. The first hidden layer has 180 units (neurons) and the second layer has 70 units. The number of units in the hidden layer is always a hyperparameter in designing neural networks and there is no valid precise measure for determining the sizes of layers. The best way is to try a few different architectures on the cross-validation set and see which design has better performance. In our case, even one single hidden layer works as the same as this two hidden layer (The Universal Approximation Theorem). As we prefer simpler models, the network with one single layer is preferred.

To update the weights, we do not use the entire training data. We use batches of the training data of size 10 and let the Adam optimizer updates the weights on each batch. We set the number of epochs to 100 so that the update cycle iterates 100 times with the learning rate 0.01. The activation function in the model is a sigmoid function because at the end we want the output layer generates values between zero and one. The other activation functions such as Rectified Linear Unit (ReLU) usually perform better as they avoid vanishing or exploding gradient but they do not satisfy our requirements for generating values in the interval \([0, 1]\). This architecture of neural networks is an example of Multi-Layer Perceptrons (MLP).

C. LSTM Network Model

The second type of networks that we apply is the recurrent neural networks (RNN). The RNN’s are usually used for sequential data such as sentences, videos, etc. In a recurrent neural network (RNN), the connections between nodes form a directed graph along a sequence. Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. Recurrent neural networks have two general types with a similar general structure, one is finite impulse and the other is infinite impulse. A finite impulse recurrent network can be unrolled and replaced with a strictly
feed-forward neural network, but an infinite impulse recurrent network cannot be unrolled. Both finite impulse and infinite impulse recurrent networks can have additional parts that handle the long dependencies between the input data.

The Figure 5 from [4] shows an RNN unrolled (or unfolded) over the time. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5 time steps.

RNN’s have gained much attention in many NLP tasks. One type of RNN’s is Long-Short Term Memory networks (LSTM). They can capture the long dependencies between words in the text. The Long Short-Term Memory (LSTM) cell was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber, and it was gradually improved over the years by several researchers. In the Figure 6, we can see the general structure of an LSTM cell [4].

The functions involved in the LSTM cell are represented in the Equations 1, where ⊗ denotes the entry-wise product between two vectors.

\[
\begin{align*}
    i_t &= \sigma(W_{ix}\cdot x_t + W_{ih}\cdot h_{t-1} + b_i) \\
    f_t &= \sigma(W_{xf}\cdot x_t + W_{hf}\cdot h_{t-1} + b_f) \\
    o_t &= \sigma(W_{ox}\cdot x_t + W_{ho}\cdot h_{t-1} + b_o) \\
    g_t &= \tanh(W_{xg}\cdot x_t + W_{hg}\cdot h_{t-1} + b_g) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
    y_t &= h_t = o_t \odot \tanh(c_t)
\end{align*}
\]

where \(W_{xi}, W_{xf}, W_{xo}, W_{xg}\) are the weight matrices of each of the four layers for their connection to the input vector \(x_t\) and \(W_{hi}, W_{hf}, W_{ho}, W_{hg}\) are the weight matrices of each of the four layers for their connection to the previous short-term state \(h_{t-1}\). Finally, \(b_i, b_f, b_o, b_g\) are the bias terms for each of the four layers.

The key idea about an LSTM network is that it can learn what to keep in the long-term state, what to throw away, and what to read from it. The LSTM cell looks exactly like a regular cell in an RNN network except that its state consists of two vectors: \(h_t\) and \(c_t\) ("c" for "cell"), where \(h_t\) and \(c_t\) are thought as the short-term state and the long-term state.

First, the current input vector \(x_t\) and the previous short-term state \(h_{t-1}\) are fed to four different fully connected layers where each layer serves a different purpose. The main layer is the one that outputs \(g_t\). It has the usual role of analyzing the current inputs \(x_t\) and the previous (short-term) state \(h_{t-1}\). The other three layers are called gate controllers. They use a sigmoid activation function, so their outputs are in the interval \([0,1]\). These outputs are multiplied element-wise with other values. So if they output 0s, they close the gate, and if they output 1s, they open it. The forget gate (controlled by \(f_t\)) determines which parts of the long-term state should be erased and the input gate (controlled by \(i_t\)) determines which parts of \(g_t\) should be added. Finally, the output gate (controlled by \(o_t\)) determines which parts of the long-term state should be read and output at this time step. The long-term state \(c_{t-1}\) traverses the network from left to right, then it enters a forget gate, dropping some memories, and then it adds some new memories via the addition operation. The result \(c_t\) is sent straight out, without any further transformation. So, at each time step, some memories are dropped and some new memories are added.

Now, that we explained how an LSTM cell works, we focus back to our model. The architecture of the network that we use for our classification task is shown in the Figure 7.

We start with sequential data derived from concatenating of word vectors in the reviews. This results in a variable length input. To fix the size of the input data, we should specify this length which is denoted by \(k\) in the Figure 7. This value is known as the number of time steps in the literature and specifies how many steps from your current position you have looked back to predict the future value. In this context, it specifies how many words of a review we will be using. If the review is shorter than this number, then that review is padded with vectors of zero’s. We set \(k\) to two different values \(k = 28\)
and $k = 60$ and call the resulting models as LSTM28 and LSTM60. The former model has an input data of $28 \times 300$ dimension and the latter model has an input of $60 \times 300$ values.

Each LSTM cell has 150 units. Thus, for all $t$, the states $h(t), c(t)$ and $y(t)$ are $300$-dimensional vectors. In this architecture, we throw away all outputs $y(0), \ldots, y(k-2)$ and only keep the last output $y(k-1)$. If a review has length less than $k$, then this output would be the output for the last word because other outputs up to the $(k-1)$-th output would be zero. Then we feed this output to a fully connected layer of eight nodes, one for each label. Just like the feed-forwarding network we use the output of this fully connected layer as the probabilities for the labels generated for the review. The learning rate here is set to a smaller value $0.001$ but other parameters such as optimization method and the number of epochs are identical to those for the feed-forwarding model.

D. Experimental Results and Discussions

We test the performance of these models on the test set which forms around 20% of the original data. The results are shown in the table below. The metric that we use is the Hamming Loss. Based on this metric, the feed-forwarding model using the average vector approach for review representation outperforms the classical models. As expected, our neural network models perform much better than the classical models for multi-label classification using the binary relevance approach. It worth mentioning that the worse performance of both LSTM28 and LSTM60 could be due to a shallow architecture of these network. We only have a single layer of LSTM cells and it is performing better than classical approaches. Another possibility is that our concatenation approach needs some improvements so that the the stop words like “the” induce less effect.

One possible future approach is that instead of concatenating all vectors, we only concatenate the vectors words that belong to a specific category. For this purpose, we can utilize POS tagging to build these categories, derive the words from four categories nouns, verbs, adjectives and adverbs and then concatenate these refined vectors. Another interesting approach would be using a pre-trained doc2vec model to represent a review by a single vector and then feed this vector to a feed-forwarding network.

<table>
<thead>
<tr>
<th>Model</th>
<th>Hamming Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>28%</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>25%</td>
</tr>
<tr>
<td>MLP</td>
<td>17%</td>
</tr>
<tr>
<td>LSTM28</td>
<td>23%</td>
</tr>
<tr>
<td>LSTM60</td>
<td>22%</td>
</tr>
</tbody>
</table>

VII. Conclusion

Feature extraction is one of the most important parts of building a machine learning model. For the text data, it is significantly important to find a transformation that takes raw data and generates a vector of numbers such that this transformation can preserves the semantic relations. The classical methods such as bag of words, or TF-IDF are frequency-based models and they cannot capture such a lexical connections between the words in the document. Word embedding as a predictive model has this ability to map the words that are semantically close to the vectors that are geometrically close as well.

Another unique feature of this project is the nature of the task the it does. We are building a multi-label classifier. Since we are aiming to predict the best combination of labels out of $2^L$, the exponential size for the search space makes such prediction really hard on small datasets. Moreover, the model should be able to capture the correlations between labels. This is one reason why we use a neural network to capture these correlations. Moreover, the existence of several labels in the prediction induces another challenge for evaluation of the model. We use a new set of metrics that are able to take the notion of “partially correct” for the prediction into the account.

ACKNOWLEDGMENT

The author would like to thank Professor Zack Butler for valuable discussions, comments and directions. The author also thanks Professor Cecilia Ovesdotter Alm for encouraging the author to work on this problem in the Fall 2017. The author of this report also appreciates Professor Minseok Kwon for interesting and valuable comments on the style of this report.

REFERENCES