Search Final Project

Ryder McMinn
Indiana University - Bloomington
rmcminn@iu.edu

Abstract

Code located here:
https://github.com/mcminnra/search_final_project

Introduction

In this final project, we will attempt two tasks working off the Yelp dataset (https://www.yelp.com/dataset). In the first task, our goal is be able to predict the categories of a business using textual features. In the second task, I picked applying Youtube's Deep Recommendation approach [1] to the Yelp Dataset to be able to recommend businesses to users.

Task 1 - Multi-Label Classification for Yelp Business Categories

In this task, our goal is to build a model that is able to predict the categories of a Yelp business. Our approach was to use the text in user reviews plus the business's title to be able to generate predictions of that business's Yelp categories (See Figure 1.)

Figure 1: Photo of a local business with it's associated categories.

Data Preprocessing

For task 1, we took a a couple of steps to preprocess the review text before we attempted learning. After some testing, we realized that including the business's title significantly improved performance without affecting preprocessing or training time in any meaningful way. Upon this combination of each review and it's associated business's title, we then removed all punctuation marks and stop words [2]; removing the punctuation marks allowed us to tokenize the text easier, and removing the stop words deleted words that provided little to no predictive value. This also had the result of speeding up training time, and reducing the training corpus.

After these steps, we sought to normalize the text. Normalization allows us to reduce the overall corpus by either stemming [3] each word to a root spelling or to convert a word to it's associated lemma via lemmatization [4]. This has the effect of reducing training time and increasing accuracy because it more concisely captures a words semantic meaning in all it's various forms, declensions, and conjugations. Looking at Table 1, we can see that stemming is the best approach for our review data. I suspect this has to do with the fact that you lose some semantic meaning with lemmatization. For example, 'great' becomes 'good' and 'awesome' becomes 'good', since 'good' is the lemma of both of these words. However, you lose the information that 'great' and 'awesome' provide. These offer different interpretations of 'good' that have some meaning in the context of a review or business. Losing this information means we lose some predictive value when training. With no word normalization, we then have to train on all the various forms of a word. In this case, we don't lose predictive value, instead, we don't have enough training data to be able to capture all the variations. For example, 'greater', 'greatly', and 'greatness' are all forms of 'great', and if we don't reduce them to just 'great', we need many more examples of each of these to be able to capture the information that 'great' provides. In essence, we may gain some accuracy when we have an immense amount of data, but when we have relatively few examples, it fails to learn each of them, which instead it can learn if we reduce all of them to just 'great'.

Table 1: Normalization Comparison over 200,000 Reviews (**Best Results in Bold**)

	Stemming	**Lemmatization**	**None**
Preprocessing (min.)	**21.06**	25.16	27.58
Training (min.)	**13.55**	14.58	14.86
Train Loss	.0048	.0048	**.0047**
Validation Loss	**.0070**	.0071	.0071
Test Loss	**.0069**	.0070	.0071

After normalization, we converted our reviews to a sequence of word ids and padded them with zeros, so they are all the same length. This will allow us to do convolutions over

our text data (e.g. [14, 23, 2, 67, 0, 0, 0, 0])

Approach and Architecture

Our approach to generating these categories was to use trained word embeddings and then apply a convolution layer over them (See Architecture below)

```
Embedding Layer (length=50)
↓
1-D Convolution Layer
(filters=128,
 filter_size=5,
 activation='relu')
↓
Global Max Pooling Layer
↓
Dense Layer (neurons=1000)
↓
Softmax Output(1300 categories)
```

We trained out output layer using Binary Crossentropy loss in order to be able to learn multiple categories (Multi-Label) on output. From personal experience if each text example is short (like a review), and contains natural spoken speech patterns, the best approach is to simply use the above architecture of WordEmbeddings+CNN. I tried added an LSTM layer, but since more reviews were short and contained natural speech, the LSTM had trouble finding key dependencies to remember that would improve performance. The result was significantly increasing training time, while not improving performance. I, also, tried using a simple Multi-Layer Perception with TFIDF, but the feature size grew exponentially large, and made meaningful training impossible. Memory became a huge bottleneck and prohibited using large data sizes for training

Results

Final Results

Task 2 - Deep Recommendation

Data Preprocessing

Approach and Architecture

Results

Final Results

References

[1] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[2] Wikipedia contributors. Stop words — Wikipedia, the free encyclopedia, 2019. [Online; accessed 25-April-2019].

[3] Wikipedia contributors. Stemming — Wikipedia, the free encyclopedia, 2019. [Online; accessed 25-April-2019].

[4] Wikipedia contributors. Lemmatisation — Wikipedia, the free encyclopedia, 2019. [Online; accessed 25-April-2019].