Search Final Project

Ryder McMinn
Indiana University - Bloomington
rmcminn@iu.edu

Abstract

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

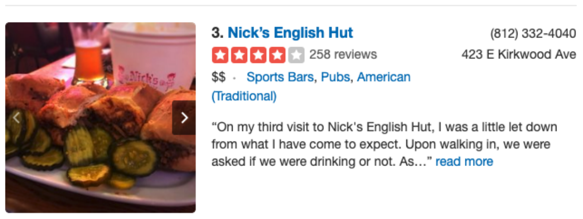
I did this project solo, so all work is my own.

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.)



3. Nick's English Hut (812) 332-4040
258 reviews 423 E Kirkwood Ave
$$ · Sports Bars, Pubs, American (Traditional)

"On my third visit to Nick's English Hut, I was a little let down from what I have come to expect. Upon walking in, we were asked if we were drinking or not. As..." read more

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 predicting 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 our 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 to add an LSTM layer, but since most 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 the loss. I, also, tried using a simple Multi-Layer Perception with TFIDF features, 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

Here you'll see my best model results. I ran out of time to do comparisons to previous iterations and techniques unfortunately. This is taken as an average of each category (class) individually since we used a multi-label approach.

Table 2: Best Results over Test Set of 199934 reviews

Metrics	**Test Set**
Loss	0.0048
Accuracy	0.9986
Average Macro Precision	0.6095
Average Macro Recall	0.3601
Average MCC	0.4507

In this problem, we want to pay attention to precision more than recall. We care more about trying to predict relevant categories than we care about predicting all the relevant categories. In our case, it's better for a business to be unlabeled than to be mislabeled. If we look at MCC, we can see our model performs well, but there is quite a bit of room for improvement. Below there is a random sample of businesses and our predicted categories for them.

```
Predicted Categories: ['Restaurants']
Real Categories:
['Restaurants', 'Seafood', 'Steakhouses']

Name: Ah-So Sushi & Steak
Predicted Categories:
['Sushi Bars', 'Steakhouses',
 'Japanese', 'Restaurants']
Real Categories: ['Japanese', 'Restaurants']

Name: Chang's Hong Kong Cuisine
Predicted Categories:
['Cantonese', 'Diners', 'Seafood',
'Dim Sum', 'Chinese', 'Restaurants']
Real Categories:
['Cantonese', 'Chinese', 'Dim Sum',
'Diners', 'Restaurants', 'Seafood']

Name: Sushi Wa
Predicted Categories:
['Japanese', 'Sushi Bars', 'Restaurants']
Real Categories:
['Japanese', 'Restaurants', 'Sushi Bars']

Name: Rancho Pinot
Predicted Categories:
['Mexican', 'Restaurants']
Real Categories:
['American (New)', 'Bars',
'Italian', 'Nightlife', 'Restaurants',
'Wine Bars']
```

Future Work

There is a lot that can be done to our approach to improve performance. Firstly, we use very little in the way of features. We only use review text and the business name. We can include location information, rating stars, various review attributes, etc. I believe if we had more time and less scrubbed data, we could build a much better performing model. Secondly, we can leverage more complex models and increased number of training data. I limited myself to a small subset of the data in interest of time, but in a real-world environment, we could train longer and possibly see more performance. As an aside, most the concepts and techniques I personally learned was from the Deep Learning [5] textbook.

Task 2 - Deep Recommendation

In this task, our goal was to apply Youtube's Deep Recommendation approach [1] to yelp businesses. We have each User, businesses they've rated, and the stars they gave it in order to hopefully learn a relation between users and businesses, so we can leverage them to make recommendations.

Data Preprocessing

Data Preprocessing for this task was reletively simple. As we needed was to merge user ids and business ids together with the stars they gave that business. This means we may have multiple samples of a single user that each contain a different business it has reviewed. Finally since each id is an alphanumeric string, I converted them to a single integer index for training.

Approach and Architecture

Our approach to generating these recommendations is to learn embeddings for both the user and the business, then concatenate them together so we can feed them into dense layers. As an output, we produce a single integer presenting the stars we believe the user would rate the business. We can then use this stars output to rank each business. (See Architecture Below)

```
User Embedding Layer (length=50)
+
Business Embedding Layer (length=50)
↓
Dense Layer (neurons=512)
↓
Dropout Layer (0.2 dropout rate)
↓
Dense Layer (neurons=256)
↓
Dropout Layer (0.2 dropout rate)
↓
Dense Layer (neurons=128)
↓
Output Layer (single value)
```

Results

Below is two randomly sampled users, see the issue?

```
User 1:

smNH7VoP8GI4RMpkM3JFLw:: Pablo
1:: Precision Window Tint - 4.56
2:: Battlefield Vegas - 4.51
3:: Gelatology - 4.51
4:: Insectek Pest Solutions - 4.51
5:: DC Auto Luxury Window Tinting - 4.51
6:: Las Vegas Spaw - 4.51
7:: HydroCare Services - 4.5
8:: Noble Carpet Cleaners - 4.5
9:: Cheba Hut Toasted Subs - 4.5
10:: La Maison de Maggie - 4.49
```

```
User 2:

A56wp0Eeaz_x5ZDDGiW_EQ:: Kjelsey
1:: Precision Window Tint - 4.51
2:: Battlefield Vegas - 4.48
3:: Gelatology - 4.47
4:: Insectek Pest Solutions - 4.47
5:: Las Vegas Spaw - 4.47
6:: HydroCare Services - 4.47
7:: DC Auto Luxury Window Tinting - 4.47
8:: Noble Carpet Cleaners - 4.46
9:: Cheba Hut Toasted Subs - 4.46
10:: La Maison de Maggie - 4.46
```

They are recommended the same thing! The issue I'm finding is that we don't have enough reviews for each user. Each user only reviews a few businesses, so the model has trouble learning their preferences only on a few examples. This has the effect of recommending them the best businesses because we don't have enough data per user to figure out what they like. The algorithm is working correctly, it's just that the data is unfortunately not suited to this type of problem. This approach works well when each user has many interactions with the system, so it is better able to learn the preferences of that particular user. It doesn't work so well when we have only a few examples.

Future Work

There are a couple of fixes to this problem. Firsly, we can reframe the problem as a probability. Instead of outputting a starts number, we can output a probability that a user will like a business. This requires to invent a heuristic that is able to tell whether a user likes a business or not. For example, we may say that a review of greater than 4 starts is a 'like' and less than 4 starts is a 'dislike'. This will allow us ot rank on the probability and hopefully give us a simplier metric to learn upon which may be able to capture the differences between users easier. Secondly, we can add more data and features. In Youtube's paper they use many more data sources to do recommendation. For our problem, we can add information such as location data, review attributes, etc. This can allow us to have more information to learn upon which may offer better recommendations. Thirdly, we can narrow our scope. Instead of trying to recommendation on a set of all businesses, we may approach the problem by recommending a subset like Steakhouses or Barbershops. This can allow us to learn more specific results and hope provide more accurate recommendations within that context. However, I ran out of time to try any of this. I wish I had more time with this project because I think this approach is very interesting and can be very successful. I just needed more time to refine it, so that it can provide better recommendations

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].

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.