Project 1 Report

Amazon Review Evaluation

Team name in Kaggle: Zezhou Sun

Final submission is using LSA, SVD and cosine similarity algorithm. Only need to check that part. Machine learning part is just a try to compare with LSA, so this part can be ignored.

# Thoughts

This Project ask us to using exist dataset to predict unknown review’s score. There is totally about 560,000 reviews, 100,000 for test and 460,000 reviews can be used for training.

So here is two ways to solve this problem in my mind:

1. Count words in summary and text, and calculate word’s occurrence frequency in each review. Use word’s frequency as a feature. How many unique words in whole dataset, the same number of features there will be. Once I get a new review, calculate the distance between it and other reviews in dataset. Review with closest distance must have similar attitude or score with this new review. Then we can get some reviews with same attitude.
2. Use regression and machine learning. Reviews are just sequences of words, so we can input it in time sequence. Then use Long Short Term Memory, build cells to accept data, then training network to let it learning what to memory what to forget. And in the last layer, use an ANN with one hidden layer to map features in LSTM to 5 score level, use cross entropy as cost to training it.

# Analysis

But both of them have problems, here is the analysis of them.

1. There are so many unique words in all these reviews. I did a basic statistic to count it, there are about 120, 000 words in total. Many of them are just typo, or useless self-made word, or transformation of other words. And also, not every word matters so much, some of them means nothing. Calculate 100,000 reviews in a 120,000 rank matrix is not possible. So we need some method to reduce running time and at the same time keep completeness of data.

Solution: Here we introduce **LSA** (Latent semantic analysis). In LSA, it assumes words that are close in meaning will occur in similar pieces of text. And by using **SVD**, we can remove those useless concepts in a review and only keep useful concepts as features. This will reduce running time of our algorithm. Also, lines with same shape of words frequency distribution (that is, in some words, the distribution of frequency of these words in one review proportion to another review) may describe the same thing, so we use **cosine similarity** to calculate distance between reviews.

Discussion solution details: Calculate word frequency will not be hard to do, so we discuss LSA part. LSA is mapping terms to concepts, and then we can use concept to distinguish each reviews. LSA use SVD to get those concept, and we discuss in our class, we can remove the concepts with low importance score and this will help us to reduce rank of whole SVD. But another problem: how many concepts should we keep? Applying Johnson-Lindenstrause Lemma will be a good idea. Notate number of concepts we want to keep as, and according to JL lemma, So to keep distance in we need at least 5 (or 6) concepts left. But we can improve it by using more concepts. To keep distance in we need approximately concepts (Also known as topics in gensim library lsi\_model).

And use cosine similarity to calculate distance between features will be easy to do.

1. Machine Learning is a good and powerful tool for this semantic problems, though we didn’t cover it in our class, but worth a try. To build a LSTM we need fixed number of cells in LSTM. But length of each review is different, the maximum length is about 3,800.

Solution: Keep only first N words, and use N to build LSTM. I run a count of this. For summary the result is: 8(90%), 9(95%), 12(98%), 19(99.9%). Number before parenthesis is N, and the number in parenthesis is how much reviews’ length is less than this N. For text the result is: 160(90%), 216(95%), 381(99%), 746(99.9%).

# Algorithm and tricks

*Note: At here I may use some terms to explain, all of them are class name I defined. I will use italic of those terms. And the definition of these terms can be found in “Top Design” part. Also because there is no need to discuss machine learning part, only discuss algorithm using LSA at here.*

There are *classifiers* in program, and for each *classifier*, once a new review input to it, it will evaluate it and return a prediction score for it. Before using *classifier*, initialize it with parameters it requires, and push data to run function. The return of run function is the predict score.

Algorithm has already described above, the main idea is to use LSA to analysis reviews and use SVD to reduce dimension of features and use cosine similarity to calculate distance between reviews. But there are more tricks.

1. Because there can be biases (or flaws) in every *classifier*, if we can use lots of *classifiers* at the same time, by aggregating them, we can get more precise result (Just as we discussed in class, like the cluster aggregation).

There are many ways to aggregate *classifiers*, but as we use RMSE as evaluation function, it is better for us take average of each classifiers. But each *classifier* may be different in accuracy, so we use weighted average of aggregate *classifiers*.

1. But it is hard for us to evaluate which *classifier* is better manually, so we use the thought of regression to do this. Use *normal\_classifiers* to do this. Every time we get a result, if it have a high deviation, reduce its weight. If it is accurate, keep the same weight. After adjustment of all weight, normalize all weight so the sum of them is still 1. Every time we adjust them, there is a learning rate, at first learning rate is high, but later it will descend and finally converge to 0 (it’s actually a like function). So finally we will get a stable weights for exist *classifiers*. We can also use this to evaluate which *classifier* is better.
2. Once we calculate the cosine similarity distance between new review to exist reviews in dataset, we will get a long list of cosine similarity distance. There are two ways we can get prediction: Use the score with highest cosine similarity (close to 1) or use the average score of first n reviews.

The advantage of using average score of first n reviews is will reduce the bias of only one review. But sometimes only the score with highest cosine similarity is actually correct. So we don’t take much reviews. In offline experiments, I found the first 3 reviews have the greatest value.

1. As there are more words in text than summary, use LSA to analysis text will be more inaccurate than the analysis of summary. This is tested in offline test. So we must aggregate the result of text with summary, so the result will be more accurate.
2. Not all words are useful. Those words only appear once cannot tell us anything, only bring higher noise to our analysis. So remove all words only appear once in whole dataset. Also, some meaningless but high frequency words like and, or, a, of, on, in…. Also cannot bring us useful information. So also remove them. This is bring us less running time and low noise.

# Offline Evaluation

To evaluate offline, in pre-processing we take 10% data (this percentage can change) out of training data.

There are totally 560,000 reviews, so after pre-processing there are 416,000 for training, 46,000 for validate and 100,000 for test.

Just as described above in “Algorithm and tricks” part No.2, we use *normal\_classifers* to evaluate *classifiers*. The classifier with higher weight is definitely better than the low weight one. So we use code BLOCK 5 in main function to do this. (Usage of this please check “How to Run” part).

After a lot of test, despite all those heuristic classifier, the order of other classifiers sort by accuracy is following:

Heuristic classifier have about 40-50% accuracy, and a straight 5 classifier (predict all to 5 score) have about 60-70% accuracy.

# Top Design

A good designed program will be friendly for code reading and pre-processing of data will need only once. Once finished, next time we can get directly from local files.

Table 1: Classes in program

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Inherit From** | **Class usage** |
| utils | - | Some common tools to use. Like parsing sentences, write and read json file. |
| dictionary | utils | Class to count words and build dictionary for reviews. |
| manipulate\_comment\_file | dictionary | Main class to store data and store models, and provide some query methods |
| normal\_classifiers | utils | A class to use lots of different classifiers to get final result |
| machine\_learning\_classifier | - | Use tensorflow to run LSTM. Different from other classifier |
| classifier | - | Interface class for a classifier provided to normal\_classifiers |
| classifier\_statistic | classifier | Heuristic, Use the distribution of score to randomly generate scores |
| classifier\_product\_average | classifier | Heuristic, Use the average of same product review history to predict new review |
| classifier\_semantic\_analysis\_highest | classifier | Use generated LSA model to run similarity query. Use score in review which have highest similarity with current one as return |
| Classifier\_semantic\_analysis\_average | classifier | Use generated LSA model to run similarity query. Use the average of scores in reviews which are first n reviews with highest similarity |

# How to Run

After the deadline of project, I will post code to my GitHub, can also check it at there.

<https://github.com/micousuen/CS565-AmazonReviews>

In main function there are 8 blocks, cancel comment if want to run this block. Red block is required for running, and blue block is optional, black block is not required or recommended.

BLOCK 1 should always be used. Will read data from local train.csv file.

BLOCK 2 should be used if this is the first time running this program. Will generate some dictionary and models and save to local, all these can be used in BLOCK 3. Loading from local will save a lot of time in next round.

BLOCK 3 is used to load dictionary and models saved in BLOCK 2. Only available after BLOCK 2 is run for at least once.

BLOCK 4 is only available for large memory computer, no need to run.

BLOCK 5 is offline testing block, use validate data to prove a classifier is useful, and also used for compare classifiers.

BLOCK 6 is actual running part for test data. There are many different set of classifiers, can only run a sub-block or several at the same time.

BLOCK 7 is the part I tried to use machine learning algorithm. But the result is not good. Model need improvement.

BLOCK 8 is only for offline testing to estimate running time. Use with BLOCK 5, but not required.