Project 1 Report

Amazon Review Evaluation

Team name in Kaggle: Zezhou Sun

User Name in Kaggle: micoubu

# 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 the main idea to solve this problem in my mind:

Count words in summary and text, and calculate word’s occurrence frequency in each review. Use word’s frequency as a feature, so how many unique words are there 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.

# Analysis

Problems: 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, called LSI in someplace, Latent Semantic Indexing). 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 a review, the frequency distribution of a subset of words in this review proportion to another review) may describe the same thing, so we use **cosine similarity** to calculate distance between reviews.

Discuss 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 review. 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 to this. 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). To get better performance, I kept 372 concepts as features in my code.

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

# Algorithm and tricks

*Note: Those italic terms are class names in code.*

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. Only need to push data to run function in that class, 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. Use regex (Regular expression) to recognize words is more accurate than use build-in split or other functions.
2. 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. So we use weighted average to aggregate *classifiers*.

1. It is hard to evaluate which *classifier* is better manually, so we 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 weights, normalize all weights so the sum of them is still 1. Every time we adjust them, times a learning rate to deviation. 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 use this to evaluate which *classifier* is better.
2. Once we calculate the cosine similarity distance between new review and existing 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 (closest to 1) or use the average score of first n reviews.

The advantage of using average score of first n reviews is it 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 3-5 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 analysis of summary. This is tested in offline test. So we must aggregate the result of text with summary, final 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…. Also cannot bring us useful information. So also remove them. This is bring us less running time and low noise.
3. Use product ID and User ID will helpful in improve accuracy. Add them to summary column.

# Offline Evaluation

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

Totally 560,000 reviews, 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 better than the low weight one. So we use code BLOCK 5 in main function to do this.

After a lot of test, here are conclusion of them:

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

In KNN, take first 3-5 elements’ average is best for prediction.

Combine 3 classifiers (highest summary, highest text and 3 highest summary) will improve accuracy and reduce cost of error.

# Appendix:

Something about the code. If you need to learn more about code including how to run it, Read this appendix. Otherwise you can just skip this.

# 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.

# Try Machine Learning

This part is just a try. And didn’t works, so skip it as you will.

Discussion: 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.

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%).

Result: This model only learned to predict every review as score 5. This is useless. Need more implement to models. FAILED

Improve: Above is using cross entropy as cost function. Should change it.