Data Preprocessing and Feature selection:

Id is similar to an index and is not useful for prediction, so it is discarded.

For product id and user id, I calculated average score for each product and each user; ids themselves are useless and are discarded.

Helpfulness nominator and denominator are kept because reviews with score 1 tend to have higher nominators and denominators. Time does not seem to be a useful indicator: all years have close average scores and standard deviation (except 1997), and the statistics for different months are even closer.

According to my observations, Summary and Text are mostly related to the scores. I processed them in several different approaches.

> First approach is to calculate their lengths. However, it turns out that lengths are not that useful because there are no concrete different between the length of a 5 star and that of a 1 star.
>
> So, I switched to use **Natural Language Process** (luckily, I am taking an NLP course this semester). The first NLP technique I used is term-document matrix. After lowercasing and removing stop words, I divided the summaries and texts from training data into 5 categories according to their scores. For each score, I calculated the vector of each word. By adding the word vectors, taking average and normalizing, the vector for a summary or a text is generated. For example, a sentence(s) vector is [0.1, 0.1, 0.1, 0.1, 0.6], so the sentence(s) will have a 0.6 probability of being score 5 and 0.1 probability for each of the other scores. I did this for both Summary and Text, so in total 10 new features are generated, and the columns containing summaries and texts are removed.
>
> After some experiments, I transformed those probabilities into the score with the highest probability. For example, the above [0.1, 0.1, 0.1, 0.1, 0.6] is directly transformed into 5. Therefore, 10 columns are transformed into 2 columns. At that point, I decided to try something more aggressive: instead of a term-document matrix of each score, a term-document matrix for each text is chosen. In that matrix, the number of rows equal to the number of texts and the number of columns is the number of words (70,000+ in my case!) I no longer need to calculate vectors; the built-in sklearn vectorizer comes handy. In order to make training faster, I also used SVD to extract features from the 70,000+ columns and the final dimension is reduced to 100+. I took advantage of the scipy sparse matrix to complete this job.
>
> *Note that I did not calculate the term-document matrix for each summary due to considerations about available RAM and training time. I chose texts because they are longer and should contain more information. However, term-document matrix for each summary maybe helpful to improve prediction and can be experimented in the future. Additionally, I tried to stem the words but the improvement is not apparent.*

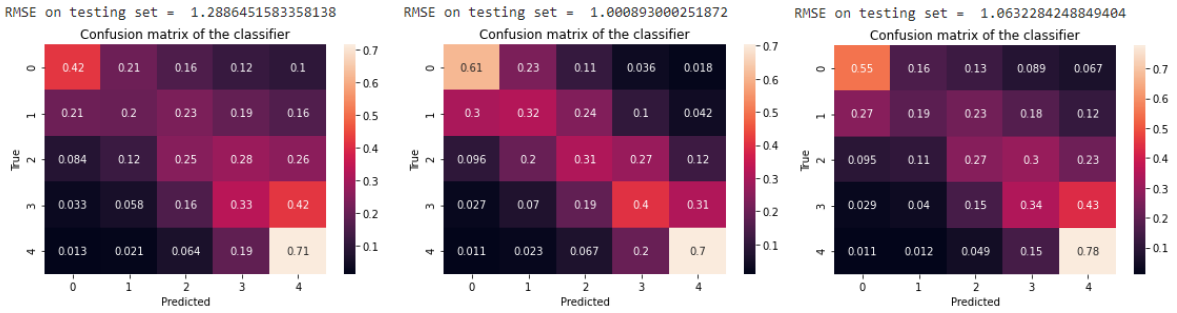
The final dataset should be something like this:

	Unnamed: 0	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Text	film_avg	user_avg	S	T
0	0	0	0	0	4.0	this is a charm version of the classic dicken ...	4.483871	4.333333	4	2
1	1	1	0	0	3.0	it was good but not as emot move as the the ch...	4.483871	3.600000	3	2
2	2	2	0	0	3.0	dont get me wrong winkler is a wonder charact ...	4.483871	3.800000	5	2

Model Selection and Testing:

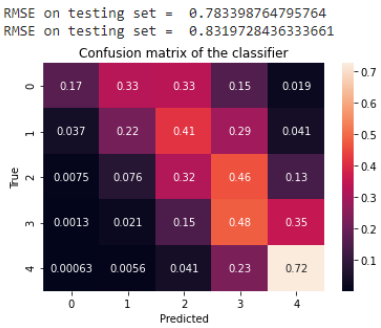
Here are several alternative models from sklearn that I tried: k-neighbors classifier, native bayes classifier, SVM, logistic regression, decision tree and linear regression.

By using a validation set and comparing root mean square error, I find that decision tree and native bayes have higher error than k-neighbors and logistic regression. K-neighbors classifier and logistic regression tend to have a RMSE slightly above 1.



From left to right: decision tree, logistic regression and k-neighbors classifier.

When it comes to linear regression, because it does not require a "hard classification" (for example, the output can be 3.5 instead of 3 or 4), the RMSE is significantly lower. Even if I manually classify the results into integers (for example, any number greater than 4.5 will be mapped to 5), though RMSE increased slightly, it is still much better than other models.



Note that all those RMSE are experimentally trained on the old dataset, which looks like below:

	HelpfulnessNumerator	HelpfulnessDenominator	film_avg	user_avg	S1	S2	S3	S4	S5	t1	t2	t3	t4	t5
28716	0.080232	0.033409	3.631579	4.400000	0.219503	0.134178	0.123222	0.160949	0.362149	0.164412	0.162329	0.182663	0.214131	0.276465
800543	-0.144842	-0.158993	4.333333	3.739130	0.202111	0.331081	0.234729	0.140305	0.091774	0.134707	0.187669	0.226090	0.253220	0.198314
488587	0.080232	0.803016	4.606852	3.865248	0.232465	0.206707	0.178003	0.184050	0.198775	0.119129	0.133421	0.176120	0.225189	0.346142

Note that SVM is excluded from the discussion because it takes very long time to run thus being unsuitable for this task.

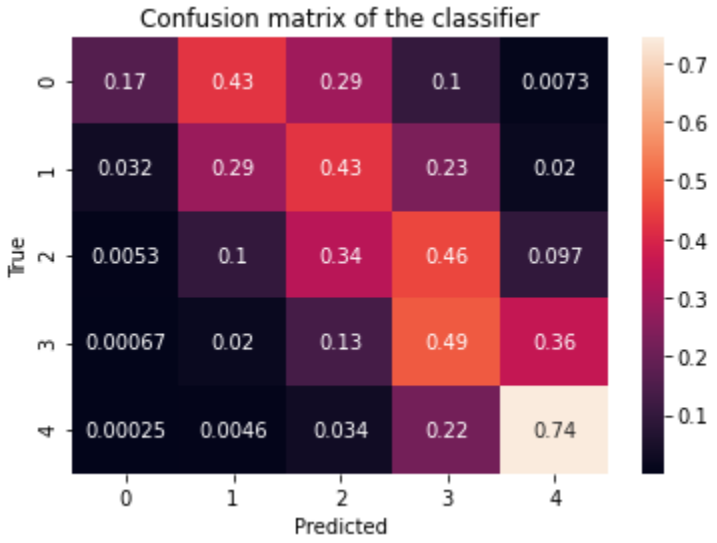
Moreover, I find that training the model on the whole training dataset (without a validation set) will slightly decrease the error (one proof is the slightly better score on Kaggle). I calculated the RMSE in the next section based on the true labels and the prediction on the whole training set.

Final Result:

On the final dataset, I used linear regression and mapped different output ranges into integer scores from 1 to 5.

The RMSE on linear regression is 0.7012115894867887; after mapping, it increases to 0.7168360246233899

Below is a screen shot of the confusion matrix:



My best score (lowest RMSE) on Kaggle public leaderboard is 0.93571