Chen Dong
Lance Galletti
CS506
Oct 27, 2021

Report for Kaggle Competition

Preliminary analysis:

After I received this assignment, I conducted basic research and have some interesting findings. First, I found out that the scores range from 1-5, which is a piece of valuable information. Because when I tried to implement one method on my extract feature section, I found out that the outcome actually became more accurate regarding percentages, but some of the scores exceeded five, which is considered invalid. The second thing I discover is that the mean score is 4.1, which is high. So I have to pay more attention to the method that I try to implement because I had to try to keep the mean about the same so that the prediction can be more accurate. The third thing I have noticed is that the importance of adversative. I found out that if an adversative is used, the score tends to be lower than the average. The fourth thing, which is also the most important finding, is that if summary and text contain more positive words, the score tends to be high, and if more negative words are contained, the score tends to below. The fifth thing I notice is that a linear regression model maybe not be a good choice since it does not fit practical situations. The last thing I notice is that this runs very slow! I have to start early!

Feature extraction:

1. Regarding the text, I use an emotional dictionary-based method to measure the positive/negative degree of the text expressing emotions. SentiWordNet assigns three sentiment scores to each synset of WordNet: positivity, negativity, and objectivity. In this midterm, I used the SentiWordNet interface of the nltk toolkit to calculate the emotional scores of Text and Summary words quickly. The specific method is as follows: first, segment the text, remove the stop words, then construct words, part of the text to query in Sentiwordnet. If it exists in Sentiwordnet, the score will be calculated automatically(if it does not exist, the score will be zero), and finally, add the total score of all words as the text sentiment score. This is implemented, and the result proves it to be successfully because when people leave their comments if they love this product, they are more willing to use positive words and vice versa.
2. Text richness and length score. The length of Summary and Text and the number of entity words can also help predict the score. The text length of Summary and Text and the length of the text after removing the stop words measure their length and richness. This works well on the testing base where the RSME decreased after this feature was applied. However, when I submitted this on Kaggle, the overall score did not indicate this is a well-extracted feature. Considering this, I conducted some other research and tests on this feature and finally discovered the fact that this does not necessarily improve the performance. However, I still keep this in feature extraction because this does not increase RSME.
3. I also used normalization to analyze my data better because the range of values is quite different, which will often negatively affect model training and fitting. Using it can limit the score and make individual scores more reasonable.
4. I used another package that comes from sklearn, which can extract the features based on variance. However, this feature selector will remove all low-variance features that I do

not want, and applying it to the model does not decrease RSME, so I decided not to use it.

Flow, decisions, and techniques tried:
The first thing I did was observe the data. I went through the Kaggle website and acquired all information. Then, I took a close look at the text and summary as well as the length of data, which is about 1.7 million. So, I know I must start this early to leave enough time for my machine to run the feature extraction. The first model I tried to use is knn, the given model. However, since knn is not model-based, it has a low bias, but that also means it can have high variance, which is called the Bias-Variance tradeoff. There's no guarantee that just because it has a low bias, it will have a good "testing performance ."In contrast, it could easily overfit the data and have very low testing performance. Also, knn is slow when if there is a lot of observations. Since it does not generalize over data in advance, it scans the historical database each time a prediction is needed. Based on these reasons, I decided not to use knn as my model.

The second model I use is linear regression. I use Linear Regression because it is simple to implement and perfectly fits linearly separable datasets, and is often used to find the relationship between variables. However, this does not output a satisfying result. I think it may be due to two reasons: Linear Regression assumes that the data is independent, while in our case, it is not. Also, it is not a good model for most practical applications because it over-simplifies real-world problems by assuming a linear relationship among the variables. Since our data come from real life, it is reasonable that this does not perform a good job.

The third model I used is the Decision Tree. I tried this because the decision tree does not require normalization of data and scaling of data. And more importantly, missing values in the data also do not affect the process of building a decision tree to any considerable extent. Decision trees perform a much better job compared to knn and linear regression.

The fourth model, which is also the model I decided to use, is Gradient Boosting Decision Tree. This is a branch of the decision tree that I think best fits our dataset by far. The advantage is undeniable because it often provides predictive accuracy that cannot be trumped. Also, it handles missing data, which imputation is not required. In addition, it can optimize on different loss functions and provides several hyper-parameter tuning options that make the function fit very flexible. The only disadvantage is that it takes lots of time and space. However, since we were given 12 days, I think it is the right thing to use a method that gives the most accurate predictions.

During the model selection, I also test a few feature extraction, especially with one-hot and nltk. I implemented one-hot for the first time. I considered using one-hot because it has the advantage that the result is binary rather than ordinal and that everything sits in an orthogonal vector space in which we clearly know what words appear or not. Besides that, it is also relatively easy to implement. However, the major factor that I decide not to use is that it doesn't scale well when the number of output labels is large. Since we have 1.7 million of data, I believe the cost of building one-hot is too expensive (it requires too much space). So, I tried sentiwordnet in nltk. Sentiwordnet has its own libraries of words and a scoring system. After I implement this, the

RSME score decreases significantly. Sentiwordnet includes a wide range of words which I think covers most of the text and summary.

When tuning the model, I used K-fold cross-validation and grid search in model testing. In this midterm, I can obtain the performance of the model on the test set according to the submitted results and then adjust the model on the test set. But I have noticed that in real applications, it is difficult for our model to find a single test set to adjust, and this approach can easily lead to overfitting of the model. Therefore, I used K-fold cross-validation, divided out 1/5 of the training set as the validation set, and used the grid search method to search for the hyperparameters of the model within a certain range. The model trained in this way can better balance accuracy and robustness. The specific implementation uses GridSearchCV in sklearn.

Model validation/testing:
I think a good model should produce a relatively accurate prediction, and that is the reason why I use the gradient boosting decision tree. For instance, linear regression did not give very accurate data, which is not considered a good approach. Also, a good model should not overfit one particular dataset. I believe that a good model should apply to a more general setting. If the model only works in one situation, it is not efficient at all because we cannot develop new models for every single dataset. In addition, a good model should not be too complicated to implement. If a model requires too many preparations, then it is not a good approach overall. Too many settings before testing can also lead to more problems. Moreover, a good model should also use relatively less time and space. This is important because sometimes, we need to use a model in a timely manner. As a result, I chose the gradient boosting decision tree since it dissatisfies the last one, but we are given plenty of time, so it is not a serious problem.

Still, testing is really important, and I know this is a good model until I see the RSME decreases because of the implementation of the model. I mainly just apply all models and test them several times to see whether they can give an accurate prediction or not.

Creativity/challenges/effort:

1. The challenges are various. The biggest challenge for me is that I have to decide which model I should apply because I know if the model is not selected properly, then the efforts dedicated to the model may not actually provide a better prediction.
2. I try to extract a feature called adjective because I think the way people describe a product also makes a lot of difference, but I do not have enough time to implement it.
3. I start this assignment after it has been released. I dedicated a lot since one extract feature run will take about 10 hours to do so.
4. I also think about extracting punctuation from the text and summary. However, I think it is more complex and also do not have a clear pattern to conclude just from what punctuation people use.

Future:

1. If I can redo this midterm, I would try more models since model selection is so important.
2. Also, I should discuss with my friends more about strategies we apply so that we all get more ideas from each other.

Thank you so much for reading the report!