method:

这部分不知道是不是要算 data preprocessing:

extract train and test files out of original hotel files:

Because we want to choose the best matched hotel in the same city with the given test hotel, we would have to make sure that there are indeed some hotel files in that location. In our current implementation, we create the test files also from the TripAdvisor dataset as simplification. The way we created train and test files is: we go over all the original hotel files, and add the location with its current appearance times into a map. If the location already appeared once in one hotel file, when it appears again, we then choose this file and modify it (extract location, hotel name, comments, etc.) then put it into the test folder. All the other hotel files which locate in that city will be modified in the same way but put into the train folder. That means, in the end of this process, we have exactly one test file for every location, which appeared more than once in our dataset, and all the other hotel files will be used for training.

Though the test file also contain a label, we assume that the overall rating of test file is unknown. Like later stated in the further work, if we use some other data as test files then there will be no such label. So this is also a simulation for using other data.

Our process to choose the best matching hotel will be separated into two steps:

In the first step we will try to figure out the most similar hotel to the given test hotel. We assume that the similarity between two hotel files could be related to the overall rating. In this case we take the overall rating as the label, and we want to use libsvm to train a regression model by using the tf-idf values of feature words as vectors. For calculating the tf-idf values for one hotel file, we take each paragraph within this file as a document, so basically we have for every feature word a tf-idf value for each paragraph after the first calculation. Then these values will be summed up and we take the sum then as the tf-idf value for this feature word for the whole hotel file.

feature words evolutions:

To decide whether a comment is positive or negative and how positive it is depends on many factors like context. So it is a hard task to make decision by just using several feature words, so the selection of feature words is important for the performance.

The TripAdvisor dataset already provided more than thousand feature words but not in a formal feature words file format. After ordering the feature words by their indexes, we choose the first 400 feature words from the original file as our first feature words file. After the using these vectors to train the regression model with libsvm, we found that the predicted values are not accurate. Although the size of feature words is large, it does not mean that all of the feature words are relevant for the label. For example, the words like “staff” or “service” does not imply any emotion of the comment. This kind of neutral words may be useful with a context detector, but is not a good choice for a tf-idf calculation.

So the first evolution of feature words is to delete those neutral words. After the first deletion we still have around 300 feature words, which showed no big improvement after testing over a small part of the data. So the words like “sun” or “warm” will be also deleted because these words don’t have a clear

relation with the label. However these words may be important for the user because they are important features of a hotel, thus we will use some of them again later in the second step. For current step we only need to find out the features that do have an clear impact over the label. After this deletion we have around 40 features.

The synonyms or words that indicate similar emotion levels, for example, “love” and “enjoy”, were defined as two different features. Intuitively we know that this is actually unnecessary. Making those words as separate features will make the tf-idf values of such words vibrate and reduce the relevance between those features and the label. This will make the regression harder and inaccurate. So for the next evolution we collapse those words into one feature. Once a word from the hotel file is detected to be the same with one of these words, the hit of this feature will added by one. With help of this change, the size of features is reduced again, and the values of vector are more relevant to the label.

Though we won’t include context detection here, some simple work could be done over the tf-idf calculation together with the improvement over feature words. For example, “don’t like” means totally different as the expression “like”. Until now the two expression will be both seen as hit on “like”, and this made the calculation inaccurate. So we try to recognize some simple negative patterns. If we found that the last word is “not” or ended with “n’t” we will then reduce the hit of this feature by one if the overall hits is not zero. Besides we found some typical typos in the text like “dont”, this will also be included as a pattern to be detecting. Because we have to finish the calculation for each word before we read the next word, so we could not detect the pattern which include the words after current word, “at all” for example. However those patterns we found already helped to improve the performance of calculation.

Because the size of feature words is already quite small, we could directly observe the feature words with the label. Then we found some feature words, which were thought intuitively to be relevant to the label, like “like” or “nice”, vibrated a lot and did not contribute much to the learning. Words like “like” or “well” could have another meanings which do not indicate emotions, and because we don’t have context detection, we cannot figure out whether these are the case. So we simply discarded these words. And for words like “good” or “nice”, the reason why they don’t work well is more complex. Firstly they are too common, even with the idf calculation, they still don’t have much impact over the label. Secondly, some customers may like to list the pro and cons in the comment, and this comment will definitely contain words like that, but the overall emotion is still unclear. Thus we discarded those words and this is the last evolution of feature words.

There is another change in the vector file. We created a new feature which indicate the overall positive feature rates in the comments. We know that the overall rating of a hotel, which is the label, mainly depends on how “positive” the comments are. And from our current feature words, they are all clearly positive or negative. So we could calculate the ratio of positive feature words over all feature words for each paragragh, and then get the average rate over all paragraphs, which will be introduced as our last feature. And this new feature does have a clear relation with the label and helped improving the performance.

libsvm:

All the values of feature together with the label will be formatted as a vector. And all those vectors will be saved into a vector file. After generating the vector file, we have to put it into the libsvm to train a model. The train mode we have chose is nu-srv, which does a regression train with all the features we give, which is exactly what we want. The other parameters will be set with default values. After the learning we then use the trained model to predict the label of our test files. The MSE

(mean-squared-error) of prediction is chosen as our evaluation value. The MSE of the original feature words file and the latest modified feature words file will be compared with each other in the section evaluation.

After we finished predicting the label of test files, we will let the user typed in a location, so that exactly one test hotel will be chosen from the test hotels. After that we will find all the train hotel files which locate in the same city with this test hotel and put them into a list. This list will be reordered so that the hotel with the most similar label to the predicted label of test hotel will come first. Because the label of train hotels is discrete, we often have more than one hotels, whose labels are most similar to the predicted label of test hotel. These will be taken as candidates, and we will finally pick one of them out in the second step. (if there is only one candidate then we don't need the second step and could directly return the candidate as the best choice)

second step:

Now we have several candidates with the same label. It is already hard to use the svm again to figure out which is better in our case. So we will leave this to the user. In the second step the user will have to enter one or several feature words. If there are more than one feature words then they will be taken together as one feature, just like explained in the feature word evolutions. And we calculate again the tf-idf value of this new feature among all candidates. We will then return the one with the highest value for this new feature as the best matching hotel.

evaluation:

Running over the complete original hotel files will take more than six hours on laptop so we only did this twice, once for the original feature words file which includes 400 feature words, and once for the latest modified feature words file, which includes 6 feature words (but in vector file there is 7 features, because we created a new feature like explained in the feature words evolutions).

And we take the MSE (mean-squared-error) of svm as the evaluation value:

```
....*
optimization finished, #iter = 4931
epsilon = 3.4187707148314317E-4
obj = -175.78468014460978, rho = -3.972588876421037
nSV = 1345, nBSV = 62
Mean squared error = 0.33061602024265924 (regression)
Squared correlation coefficient = 0.22384799585925572 (regression)
```

Picture 1: MSE with original feature words

```
..........*.*
optimization finished, #iter = 10827
epsilon = 3.4409404237534247E-4
obj = -155.7340685260143, rho = -3.6423408245162148
nSV = 1310, nBSV = 170
Mean squared error = 0.2510052875657915 (regression)
Squared correlation coefficient = 0.37942126798750186 (regression)
```

Picture 2: MSE with modified feature words

As we can see, the MSE with modified feature words is clearly better than the MSE with the original feature words, which means the changes over feature words and the calculation of tf-idf have made improvement over the performance.

further work:

use more information besides the comment texts:
we still have other ratings like rating for room, which could also be used to help training the train model.
discard comments in non-english languages:
we should detect whether a comment is in non-english languages or contains unreadable characters so that these comments could be discarded by preprocessing.
test it with more datasets or general texts(restaurants):
in our implementation we extract the test file from the original datasets. In the future we could also use other data from other sources. The integration is easy with our current system, because what we need as a test file is just the location, hotel name and comments.

注意：对 tripadvisor 数据集需要做引用说明，libsvm 和 UIMA 可能也需要。