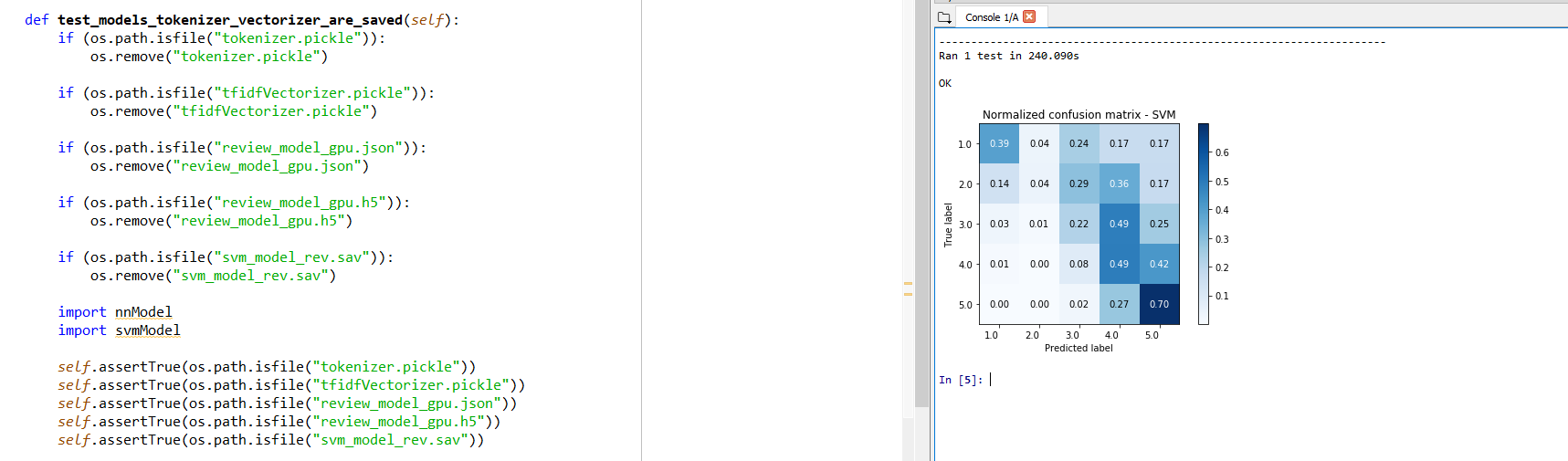
**Test report**

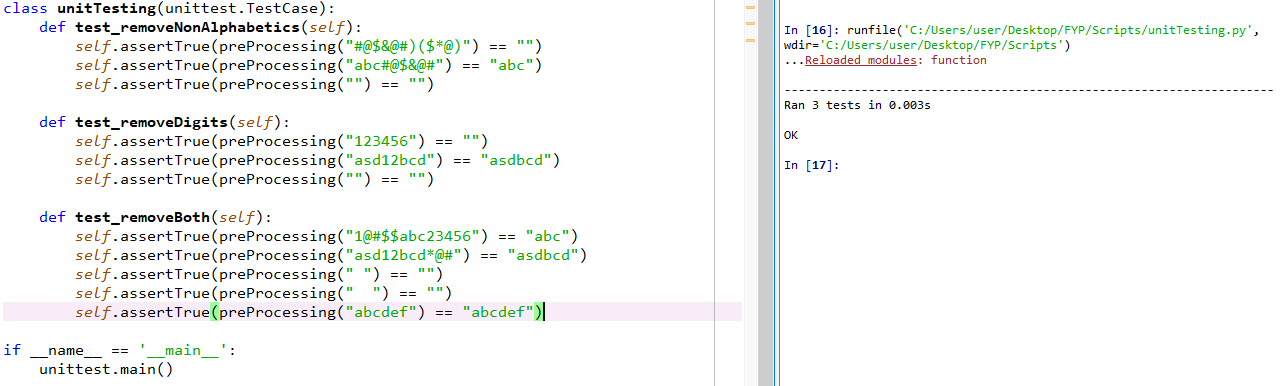
**Unit Test**

Test 1: Models and their relevant files (tokenizer, tfidf vectorizer, model weights) are saved into folder



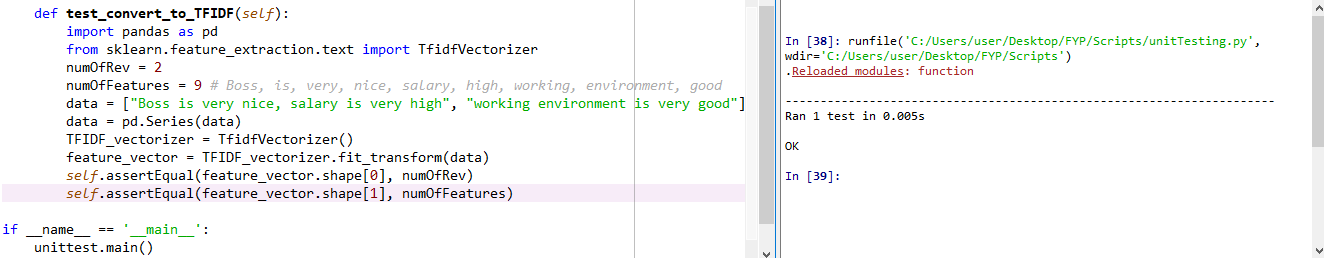
Our main.py file allows user to enter reviews or other ratings (work-balance, culture values, company benefit and so on) and subsequently generate an overall ratings associated with the user input. Our LSTM-CNN and SVM models, weights of these models, tokenizer and TFIDF vectorizer are required to be present in the folder to make main.py file to run successfully. Thus, we need to test the code that are used to generate these files. We first delete our models and relevant files saved previously, then we use assertion to ensure the relevant files are always saved into the folder (just ignore the confusion matrix as both nnModel.py and svmModel.py will generate confusion matrix for better visualization of our result).

Test 2: Remove non-alphabetic characters (including space and tab) and digits and convert review to lowercase



Since we have used pre-trained word embeddings in our experiment, we need to ensure that every punctuation, non-alphabetic character, digits are removed from each review and reviews are converted to lowercase. One of the reason is that they did not carry important meaning. Another reason is that pre-trained word embeddings treat the words “Hello”, “hello” and “hello!” are all totally different words as they aren’t exactly the same characters. Converting all reviews to lowercase makes our pre-trained word embeddings easier to pick up on statistical patterns in the data.

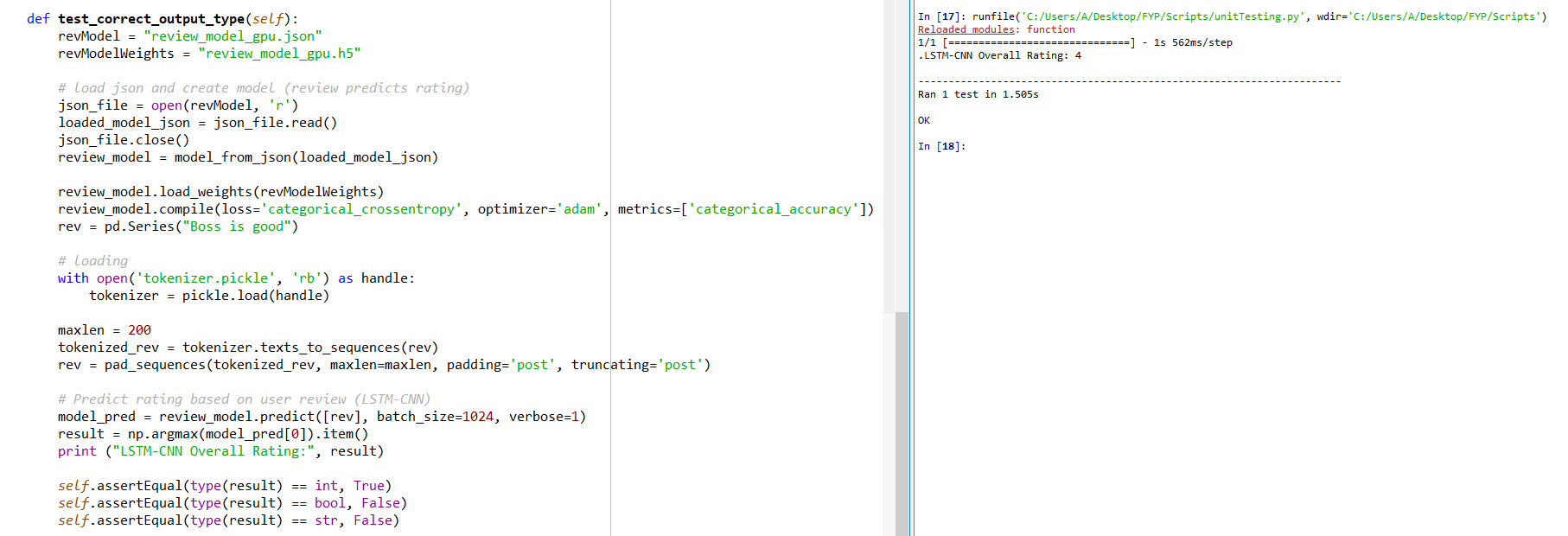
Test 3: Reviews are successfully converted to a matrix of TFIDF features



This testing ensures that TFIDF vectorizer converts reviews to a matrix of TFIDF features. In this example, we are having 2 reviews with 9 unique words (Boss, is, very, nice, salary, high, working, environment, good). Therefore, when we feed these reviews to the TFIDF vectorizer, the matrix generated by the vectorizer should have the shape of (2, 9), where 2 is the number of reviews we have and 9 is the total number of unique words in these reviews.

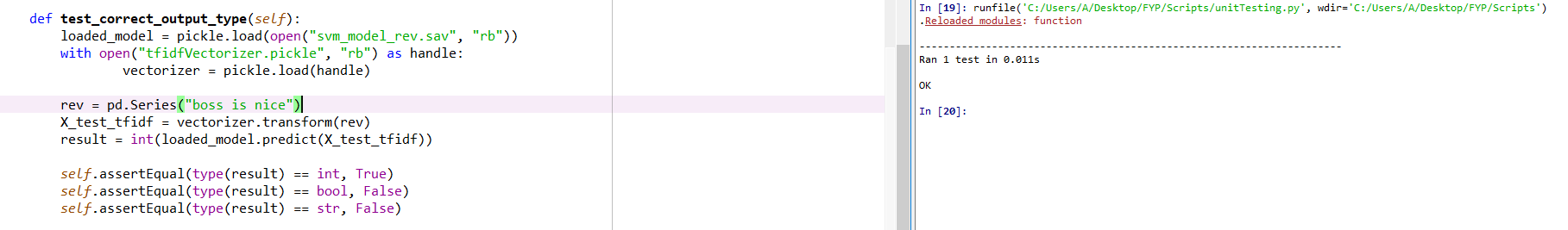
**Integration testing**

Test 1: LSTM-CNN’s predicted output has correct type



Before testing our output has the correct python native type, we need to load our model by using tokenizer, model and model weights files. This testing ensures that our predicted output type is always int as rating is in range of 1 to 5.

Test 2: SVM’s predicted output has correct type

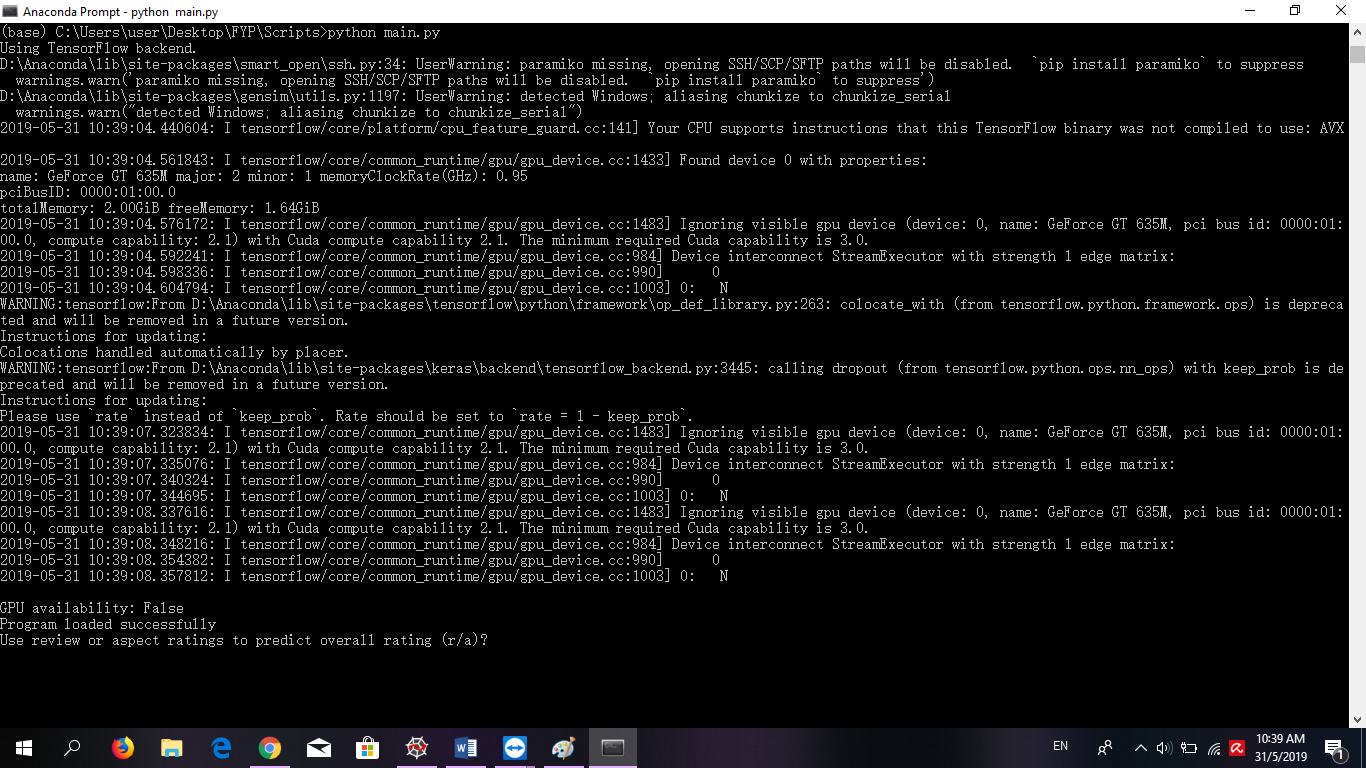


Same as above. Since we have two classifiers, we also need to test the predicted output generated by SVM is of int type.

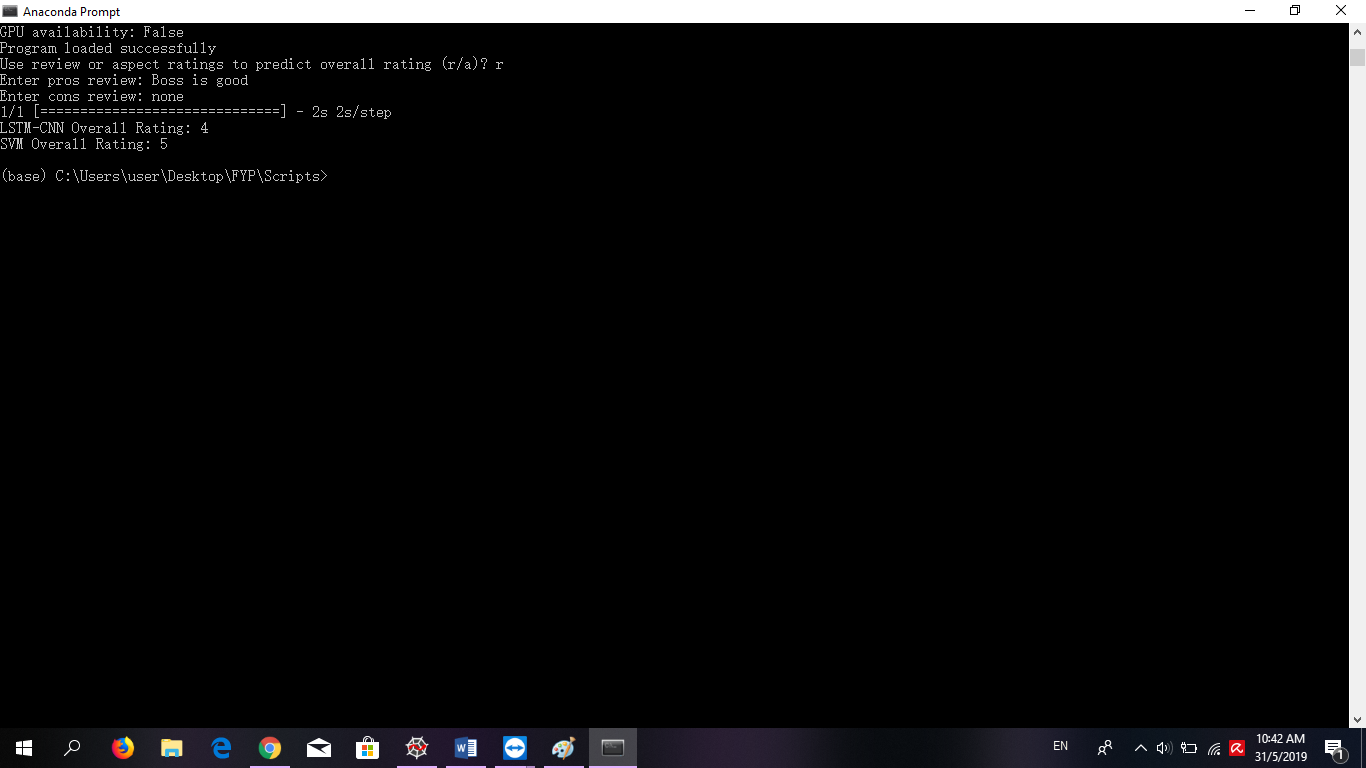
**System testing**

Device’s hardware(CPU or GPU) is detected (important, if the device does not have GPU, we have to load model built by regular LSTM rather than CudNNLSTM, otherwise the program will crash)

Program loaded successfully

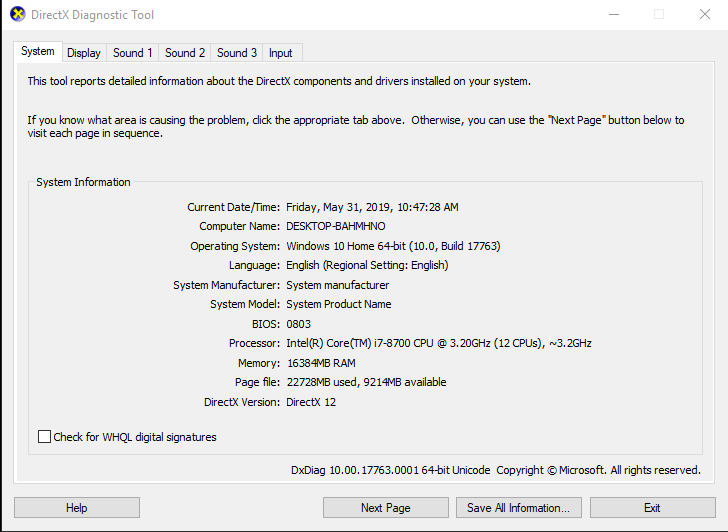
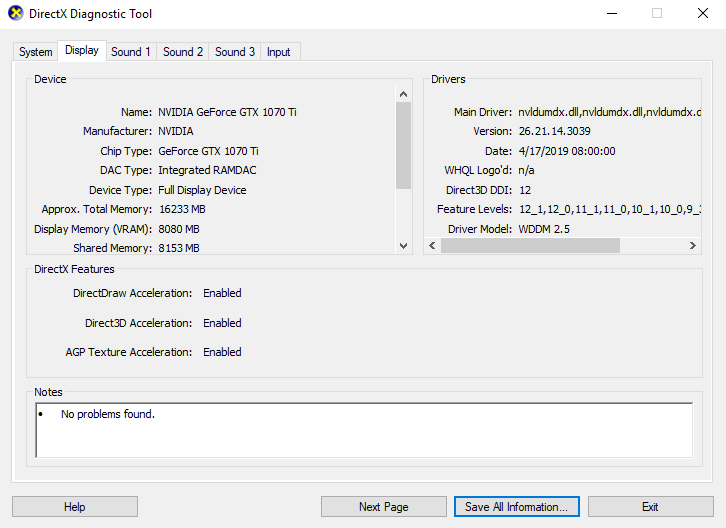


Input and output received and generated successfully



**Performance**

A desktop with the following specifications is used to record the training time of both LSTM-CNN and SVM classifiers.

Note that we only time the training time of these classifiers instead of running time of the whole process (extract columns, train test split, load embedding, generate embedding matrix, compile model and so on).

LSTM-CNN (review predicts overall rating, batch size =128, epochs = 20, early stopping patience = 2)



LSTM-CNN (aspect ratings predicts overall rating, epochs = 20, early stopping patience = 2)



SVM (review predicts overall rating, LinearSVC(C=0.1))



SVM (aspect ratings predicts overall rating, LinearSVC(C=0.1))



All functions we used is from third party libraries, therefore we could not know the internal structures of those built in functions. However, we believe those built in functions have been well-tested and thus their performance are guaranteed.

**Scalability**

For any user that would like to use the program, the user needs to clone our program and install the relevant libraries before the user can start to play with our program. If we ever update the codebase, the user would have no idea what has been updated unless the user checks our GitLab repository.

Hence, for user that have zero knowledge in using Git control, the user will face problem in cloning our program. If we can export our program to a website and host it with a server, the user would just need to access the website link and more people can access to our program too. Besides, we can also show important notice about an update to the program on the website.

**Usability**

The code that we have written follow some of the standard software design principles. First, we make sure our code follow DRY- Dun’t Repeat Yourself principle by refactoring duplicate code into a common function. As most of the code for preprocessing steps of each of the classifiers have a lot of similarities, we created a utility file which serves the main purpose to store the common functions that are used by multiple files.

We also follow the SRP - Single Responsible Principle by ensuring each of the functions only have 1 job and does not produce any side effect. We also wrote formal documentation for any important functions that is difficult to understand. If we ever open source the codebase, any beginner user that would like to modify or expand our codebase should have no problem in doing so.

**User interface**

Since we are doing research based problem, we don’t actually have a sophisticated user interface. Our program’s user interface is just the terminal. However, it is very simple to use. User do not need to care about whether their devices have GPU or not. Our program will automatically check for availability of GPU of the running device. If GPU is not found on the running device, we will load the model built by regular LSTM rather than the model built by CuDNNLSTM to ensure user can run our program successfully. Also, user only requires to enter pros and cons reviews or aspect ratings such as work balance rating, culture values rating and so on, to generate the predicted overall rating. Simplicity of the design of our program allows users have a good time using our program.

**Limitations and Recommendations for improvement**

|  |  |  |
| --- | --- | --- |
| No | Limitations | Improvements |
| 1 | Every time a model is created, it will rewrite the existing files, including the model, model weights, tokenizer and vectorizer saved previously. User cannot choose to save in other folder or rename the model he/she wants to save to prevent overwriting models built previously | Modify our existing codes to allow users to save the new model into another folder or rename it. |
| 2 | Only have two classifiers, users may want to look for more classifiers for comparisons | Include more classifiers like random forest, naïve bayes, and so on. |
| 3 | Does not validate user input values. Our context is employee review but user may enter review that is irrelevant to this context. For example, “The chicken burger is so tasty” | Add more cores to ensure user’s input has the same context as our dataset. |
| 4 | Enhancement of the design of our user interface is required | Create a sophisticated and user friendly UI |