CSC1052 A4

Jared Rushe
22489846
jared.rushe2@mail.dcu.ie

Introduction:

For this assignment I implemented one machine learning and two web application focused ideas based around classifying TripAdvisor hotel reviews into star ratings out of 5. I built a One vs Rest (OvR) multiclass classification model to break the task into 5 separate binary classification problems. The main steps involved preprocessing the data, designing the algorithm and evaluating its performance. I then expanded the OvR model into an interactive Flask web application, allowing the user to choose their preferred binary classifier from Logistic Regression, Random Forest and Multinomial Naive Bayes. The application processed text-based reviews as user inputs and outputted the models results, integrating HTML templates with CSS for enhanced navigation and usability. The third idea involved transforming the web app into a Gradio-based interface which could be accessed directly from a Jupyter Notebook. I employed Gradio's image to text capabilities by incorporating Tesseract OCR to extract text from review screenshots. This allowed users to receive predictions from their choice of model based on images instead of raw text. Each idea builds on the previous one, culminating in a versatile and interactive tool suitable for real-world application in the travel tech domain.

Idea 1: Multiclass Classification Model for Tripadvisor Hotel Review Star Ratings (Machine Learning)

Notebook: csc1052_a4_p1.ipynb

During a recent job interview for a placement in the travel tech industry, I was asked how I would go about designing a multiclass classification model for predicting the star rating of Tripadvisor reviews. I thought this would be an interesting way to progress my study of machine learning techniques beyond the binary classification tasks I had completed to date. I sourced a dataset from 'huggingface' of 201,295 real Tripadvisor hotel reviews. I chose to design an OvR classifier for my model. This would allow me to leverage my existing understanding of binary classification algorithms by simplifying the multiclass problem by breaking it down into multiple binary tasks.

I began by loading the dataset and extracting labels for the review text and associated overall star ratings. I split the data into training and test sets with an 80:20 ratio. I then created a function to change the labels of each class to binary values based on the target class. For example, when the model is evaluating whether a review belongs to the 4 star class, the label "4" is changed to 1 and all other labels are changed to "0". I then trained five separate binary classifiers, each responsible for predicting one of the 1-5 star ratings, using a scikit-learn pipeline that included a Tfidf vectorizer and linear regression model that I developed as part of A2. I fit the classifier to the training data and used it to predict the star ratings of the test set. Finally, I calculated the overall accuracy score of the model by finding the binary classifier with the highest probability for each review and comparing the predicted class to the actual labels.

I used the scikit-learn classification report to analyse the models results against the test data. Overall the accuracy for predicting each class was very high (1 star: 0.97, 2 star: 0.94, 3 star: 0.87, 4 star: 0.71, 5 star: 0.80). This shows that the model is strong at determining whether or not a review received a specific rating.The overall accuracy of the model was 0.65. Considering a score of 0.2 would be expected if you randomly guessed each rating, this value indicates that, although far from perfect, the algorithm does have a significant level of sophistication and understanding of the data. However, the classifier appears to perform much better for the majority class. For example, for the 1 star predictor there were 1566 positive cases compared to 38693 negative cases. The negative cases were categorised very effectively, reflected in the metrics: precision 0.98, recall 0.99. Contrastingly the model performed much more poorly for the positive cases: precision 0.75, recall 0.46. This pattern of higher effectiveness was present throughout the 5 binary classifiers. It highlights that experimenting with undersampling the majority classes of negative cases or oversampling the minority classes of positive cases could improve the overall success of the model. This is an idea that I will certainly consider in my future work.

Idea 2: Allowing a Choice of Models on Flask Web Application (Web App)

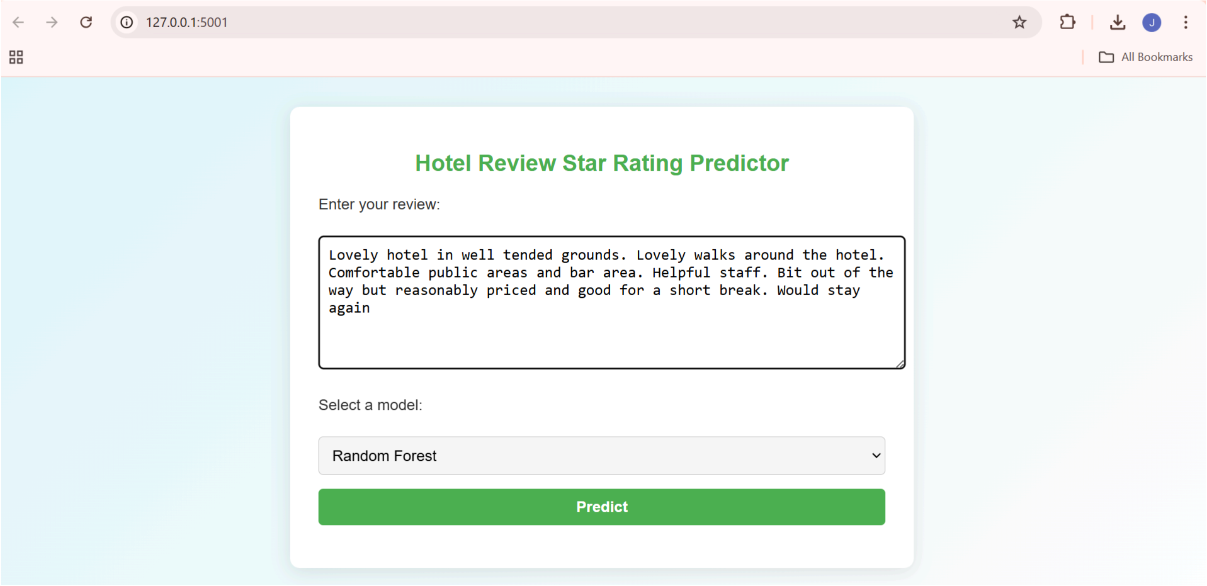
Notebook: csc1052_a4_p2.ipynb

I decided to create a Flask web application that allows a choice of which binary classifier to use within my OvR model. I reproduced my algorithm from Idea 1 two more times but changed the Logistic Regression model in the scikit-learn pipeline to Multinomial Naive-Bayes and Random Forest models. Despite the new models having a lower predictive accuracy over the test data (Multinomial Naive Bayes: 0.60, Random Forest: 0.55), there are still significant advantages to using multiple models for any prediction. Different models have unique strengths and weaknesses and relying on just one algorithm makes your conclusions vulnerable to the specific assumptions and limitations of that classifier.

I used the pickle module to save the three classifiers to .pkl files. I used pickle to load the models in the file app.py. This file contains the main Flask application that handles the web server and renders any HTML templates. I designed two HTML templates:

- Index.html: This is the home page template that allows users to input a hotel review, select a model, and submit it for prediction.
- Predict.html : This template displays the results of the prediction, including the predicted star rating, the chosen model, and a link back to the home page to submit another review.

I used CSS to enhance the visual appearance and layout of the html pages. I included a dropdown menu to select a model, replaced hyperlinks with buttons to navigate the website and varied text formatting through bolding, colours and font size to improve readability. These changes made the web application more user-friendly and aesthetically pleasing.



Hotel Review Star Rating Predictor

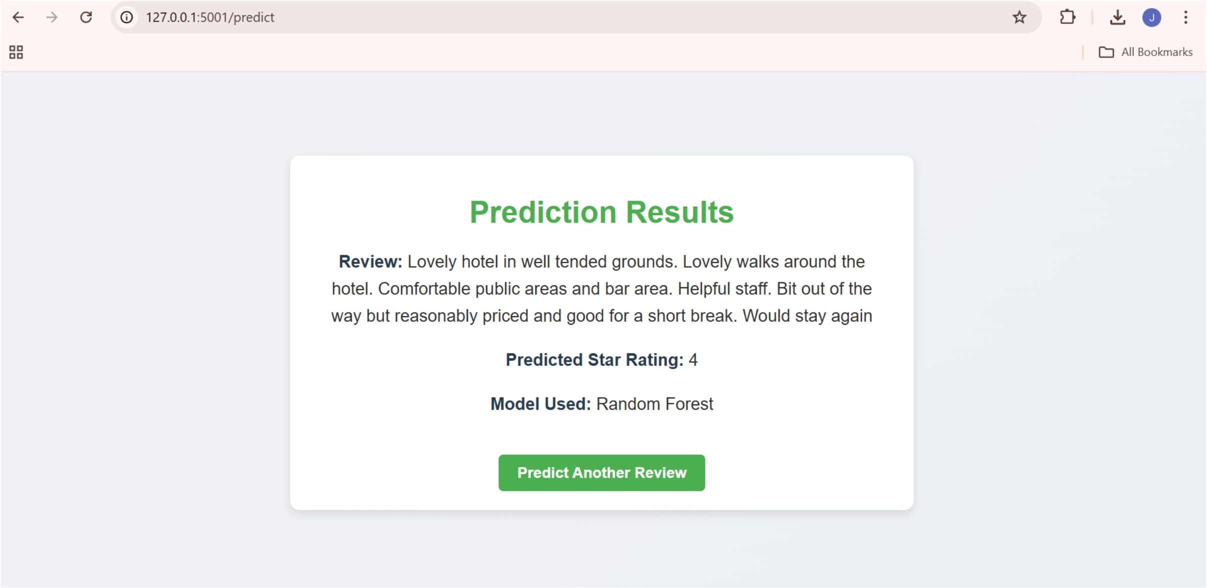
Enter your review:

```
Lovely hotel in well tended grounds. Lovely walks around the hotel.
Comfortable public areas and bar area. Helpful staff. Bit out of the
way but reasonably priced and good for a short break. Would stay
again
```

Select a model:

Random Forest

Figure 1: Running version of the Flask application displaying index.html template



Prediction Results

Review: Lovely hotel in well tended grounds. Lovely walks around the hotel. Comfortable public areas and bar area. Helpful staff. Bit out of the way but reasonably priced and good for a short break. Would stay again

Predicted Star Rating: 4

Model Used: Random Forest

Figure 2: Running version of the Flask application displaying predict.html template

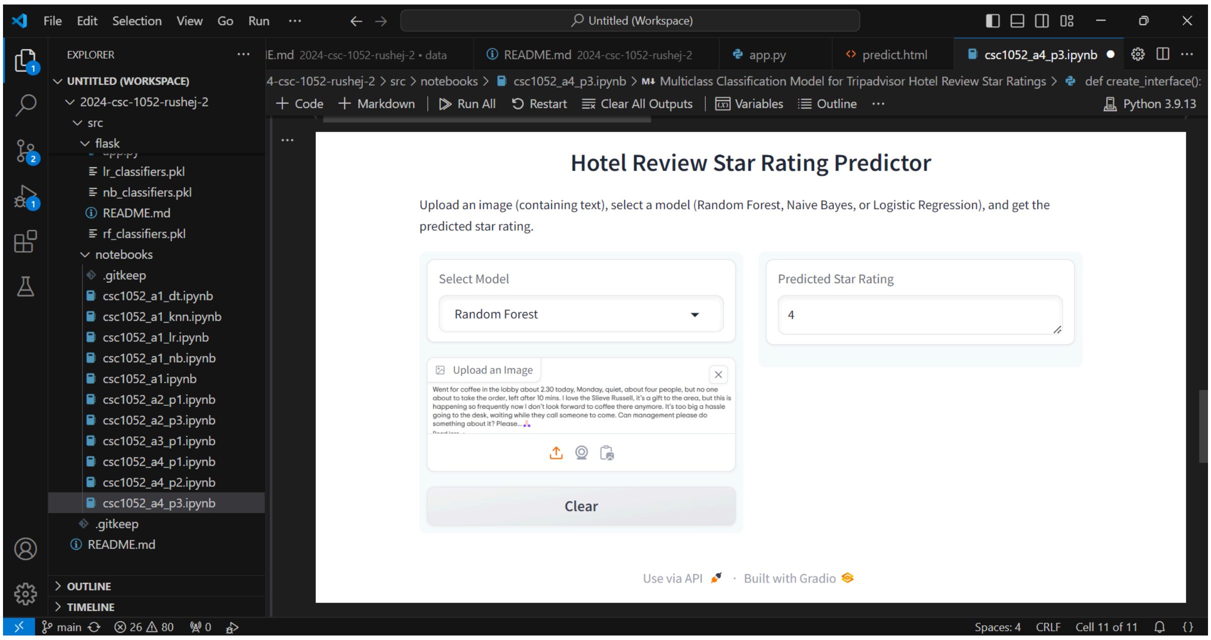
Idea 3: Image-to-Text Review Prediction via Gradio Application (Web App)

Notebook: csc1052_a4_p3.ipynb

I chose to adapt my Flask application from Idea 2 to a Gradio-based interface. This was because of Gradio's seamless integration of machine learning models with minimal code. I ran the application from a jupyter notebook, 'csc1052_a4_p3.ipynb' With pickle, I was able to smoothly load in the three predictive pipelines using their file paths from the Flask folder in my repository. I decided to change the input type for the app from text to images. This added realistic functionality to the model as many users would be more likely to upload a screenshot of a review posted online rather than the raw text itself. I used pytesseract, an optical character recognition (OCR) tool for python that recognises and converts the text in images.

I created a python function to “read” the text from the pictures uploaded to the app. The selected classification model was then applied to the text and the predicted star rating was returned. Next, I designed a Gradio interface integrating this function. It allowed users to upload an image, select their desired algorithm and view the output in an interactive and user-friendly manner.

Being able to run the Gradio app locally, directly within the notebook environment was very useful as I was able to iteratively refine the application while being able to see all my code and its impact in one place. This made it much easier to debug problems as I could see the output, error messages and code together without having to run a separate developmental server.



Hotel Review Star Rating Predictor

Upload an image (containing text), select a model (Random Forest, Naive Bayes, or Logistic Regression), and get the predicted star rating.

Select Model: Random Forest

Predicted Star Rating: 4

Upload an Image

Went for coffee in the lobby about 2.30 today, Monday, quiet, about four people, but no one about to take the order, left after 10 mins. I love the Slieve Russell, it’s a gift to the area, but this is happening so frequently now I don’t look forward to coffee there anymore. It’s too big a hassle going to the desk, waiting while they call someone to come. Can management please do something about it? Please...🙏

Clear

Figure 3: The functioning Gradio interface embedded in a Jupyter Notebook

Personal Retrospective

During my studies towards the CSC1052 Applications Domain module, I greatly furthered my knowledge in the areas of machine learning, web applications and general software development tools. My machine learning work was centred around designing predictive classification models. I learned how to effectively preprocess large datasets and partition their contents into appropriate training and test splits. I also gained great experience working with scikit-learn, leveraging its tools to vectorize text, train and fit data on various models and evaluate performance using a range of metrics. By combining these features I could experiment with what combinations of vectorizers, models and parameters were most suitable for different types of data. My web application development was primarily centred around the Flask framework. I learned how to structure and compile various file types such as python to initialise the application, html to create the user interface and pickle to import the machine learning models and vectorizers. I also practiced adapting applications across different frameworks such as Django and embedding them in python notebooks through Gradio. I conducted all of my assignments for this model through Jupyter notebooks in VSCode and uploaded my work to a Gitlab repository. Getting hands on experience with these development and collaboration tools greatly improved my documentation, version control and project organisation. I know these skills and concepts will stand to me in my future studies and as I enter professional environments.

Appendix:

- Trip Advisor Review Dataset: https://huggingface.co/datasets/argilla/tripadvisor-hotel-reviews
- Scikit-Learn Multiclass Documentation: https://scikit-learn.org/1.5/modules/multiclass.html#onevsrestclassifier
- Flask Machine Learning Model Implementation: https://medium.com/@qleedham/how-to-deploy-a-machine-learning-model-using-flask-922dc047ddd7
- Gradio Quick Start Tutorial: https://www.gradio.app/guides/quickstart
- Pytesseract Download and Description: https://pypi.org/project/pytesseract/