“UmbrellaAlert”: A Weather Forecast App for Specific Rain Prediction

Marzia Paschini
University of Trieste
MARZIA.PASCHINI@studenti.units.it

ABSTRACT
“UmbrellaAlert” is a web-based application that uses data-driven methods to provide accurate rain-specific weather forecasts and visualizations of historical weather data. The application employs a supervised machine learning model, specifically an information system based on binary classification has been developed to support its operation.

Keywords
weather; prediction; rainfall; numeric; categoric;

1. INTRODUCTION
The aim of “UmbrellaAlert” is to provide users with accurate rain forecasting using machine learning. The application allows users to input current weather parameters and receive a binary forecast indicating whether it will rain or not on the next day. In addition, users can visualize historical weather data, the ones used to build the prediction model. This paper outlines the development and features of “UmbrellaAlert”, highlighting its potential to offer valuable insights and improve decision-making in weather-related contexts.

2. BACKGROUND
2.1 Data Discovery and Exploration
The project started with data discovery. An open dataset available on Kaggle[1] that contained weather data was found and selected. The dataset consists of 25,000 weather observations collected in various locations across Australia, covering every day from 2008 to 2017. The idea of developing a weather prediction software emerged, along with tools for visualizing the underlying data.

The dataset consists of both categorical and numerical features, as can be observed in Table 1. Every observation is intended as collected in the respective location and date. It should be noted that the `Evaporation` and `Sunshine` columns were not accompanied by additional information, and therefore some assumptions had to be made regarding their interpretation.

Table 1. Names, types, and brief descriptions of the dataset's columns

Column	Type	Description
Unnamed: 0	Numeric	Index of observations.
Date	Numeric	Data collection date.
Location	Categoric	Location of data collection.
MinTemp	Numeric	Minimum temperature.
MaxTemp	Numeric	Maximum temperature.
Rainfall	Numeric	Amount in millimeters of rainfall.
Evaporation	Numeric	Amount in millimeters of the conversion between liquid water and water vapor.
Sunshine	Numeric	Amount in hours per day of direct solar radiation received.
WindGustDir	Categoric	Direction of wind gusts.
WindGustSpeed	Numeric	Speed of wind gusts.
WindDir9am	Categoric	Direction of wind gusts at 9am.
WindDir3pm	Categoric	Direction of wind gusts at 3pm.
WindSpeed9am	Numeric	Speed of wind gusts at 9am.
WindSpeed3pm	Numeric	Speed of wind gusts at 3pm.
Humidity9am	Numeric	Humidity at 9am.
Humidity3pm	Numeric	Humidity at 3pm.
Pressure9am	Numeric	Pressure at 9am.
Pressure3pm	Numeric	Pressure at 3pm.
Cloud9am	Numeric	Cloud at 9am.
Cloud3pm	Numeric	Cloud at 3pm.
Temp9am	Numeric	Temperature at 9am.
Temp3pm	Numeric	Temperature at 3pm.
RainToday	Binary	If it has rained that day or not.
RISK_MM	Numeric	Amount of rainfall in millimeters for the next day.
RainTomorrow	Binary	If it will rain the next day or not.

2.2 Software Development Method
As a generic product of small size, and due to the fact that the source dataset is fixed since it pertains to historical data, the requirements for the project were well-defined and did not require strict budget planning. Therefore, an Agile development methodology was deemed appropriate, specifically an incremental and iterative approach that interleaved the specification, design, and implementation stages.

Using an IDE and graphical tool-set for user interface design helped reduce overheads in the development process, leading to faster progress, more frequent feedback, and a more efficient development process. Additionally, the Scrum framework was used to break the project into smaller sprints and prioritize development tasks.

Approximately 80 hours were dedicated to the development of the “UmbrellaAlert” system. The initial phase of dataset acquisition and project structuring required 15 hours, while software development methodology and UML design took 26 hours. Data pre-processing and predictive model development each took 15 hours. A final round of checks and error fixing was completed over a period of 8 hours. Throughout the development process, progress was made in writing the accompanying report, which documented the various stages of the project.

2.3 Software Specifications

This section presents a detailed specification of the software developed in this paper, which is, as previously stated, a data-driven web application that employs a machine learning kernel. The application was built using the Streamlit web application framework, while the machine learning kernel was developed on a separate platform, such as a local computer by using the VSCodium IDE. The primary programming language used for development was Python, with the `numpy` and `pandas` libraries being particularly useful. As a Version Control System, GitHub has been used.

3. APPROACH

3.1 Software Requirements

To meet the needs of the users, the software must adhere to the following requirements. The application must allow the user to input current weather parameters, which must be validated before generating a prediction for the next day. Specifically, the inserted parameters include the minimum and maximum temperature for the day, a check for whether it has rained or not, and if so, the amount of rainfall in millimeters, as well as humidity and wind speed. The application should feature a user-friendly interface that is highly responsive, with a maximum response time of a few seconds. Additionally, the application must produce highly accurate weather predictions.

3.2 Software Organization

The software organization of the system is described using UML diagrams, including diagrams of components and classes.

The component diagram in Figure 1 illustrates the software system's decomposition into components and their dependencies. Two main subsystems have been identified: the *Application* and the *MLKernel* (*Machine Learning Kernel*). The *Application* subsystem comprises three pages that are mutually accessible, and the interfaces between the components enhance user interactions. The use of `pandas` library and `streamlit` framework were particularly enhanced in this diagram, due to their importance in the overall work.

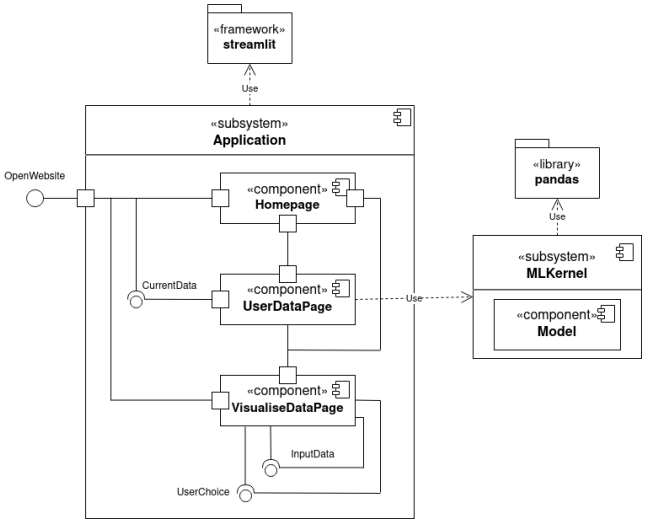

Figure 1. UML Component Diagram

The class diagram exposed in Figure 2 describes the structure of the system by showing the system’s classes and their attributes.

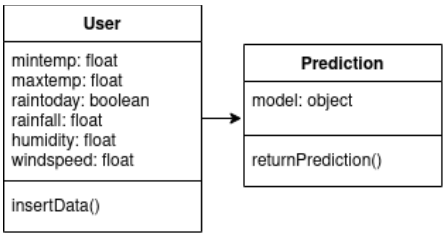

Figure 2. UML Class Diagram

The system's relatively simple nature led to the identification of two essential classes for representing the primary entities involved in the web application. `User` encapsulates the user's input data, used for prediction. The `insertData()` method enables users to insert or update their data. The `Prediction` class contains the machine learning model object and returns the prediction to the user, through the `returnPrediction()` method.

3.3 Software Functionalities

Two additional UML diagrams are used to showcase the software's functionalities.

The first is a high-level view of the system's capabilities, depicted in Figure 3 as a use case diagram. The diagram illustrates the relevant actors, which are the user and the machine learning kernel, and their corresponding interactions with the web application. Each actor has access to specific use cases that enable them to achieve their objectives. The user, for instance, can choose between inserting current data to generate predictions or viewing previous data. These two use cases involve filtering the original dataset and displaying the prediction results, respectively. The machine learning kernel, instead, only works about the prediction output.

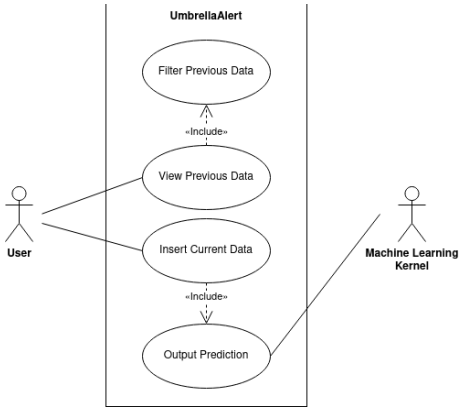

Figure 3. UML Use Case Diagram

The flow of activities in the system is represented through the activity diagram exposed in Figure 4. The diagram identifies three entities: the user, the application, and the machine learning kernel. The diagram illustrates the various actions that the user can perform within the system, and shows the corresponding responses of the application. The machine learning kernel is invoked only when a prediction needs to be made.

3.4 Data Analysis

To identify the most suitable algorithm for rain prediction, a data pre-processing phase was carried out. First of all, the correlation matrix for the numerical features reveals that there is little evidence of significant correlations between variables, except for the expected correlations between related variables such as `MinTemp` and `MaxTemp`, `Temp9am` and `Temp3pm`, `Pressure9am` and `Pressure3pm`, and so on. This implies that the numerical features may provide unique and independent information, and furthermore, the lack of correlation between the features could make the eventual feature selection more challenging.

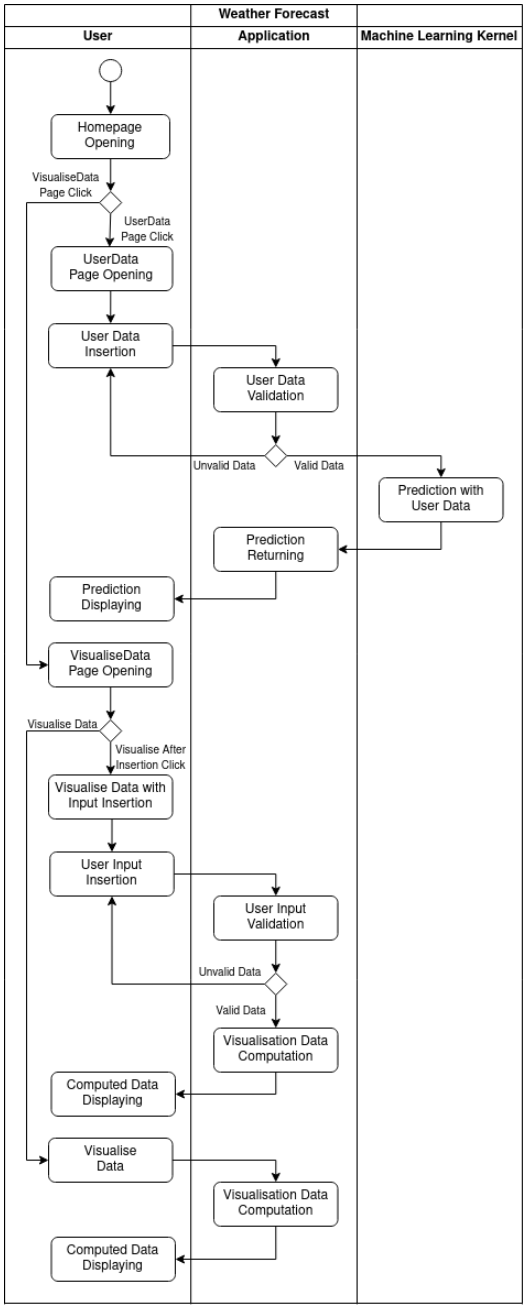

Figure 4. UML Activities Diagram

In the second step of the analysis, missing and null data points were identified, and columns with over 60% missing values, such as `Evaporation` and `Sunshine`, were removed due to their potential risks of falsifying the prediction of the target variable. To pre-process the other features, first the rows with missing values in the `RainToday` column were removed, which was deemed important for the prediction task. For the remaining missing values, they were filled in using mean or mode values, depending on the column type (i.e., categorical or numerical). To ensure the accuracy of the imputation, the filling operation by grouping the data by `Location` values was performed. This was done because the different geographic locations of Australian cities can have significant differences in meteorological data, and thus, grouping by `Location` was expected to improve the accuracy of the imputation.

Then, the `Date`, `Location` and `RISK_MM` columns were removed. The first two columns were deemed unnecessary for the analysis, while the `RISK_MM` column was excluded to avoid spurious predictions, as it is a prediction in itself. In this way, the amount of data has been reduced.

In order to prepare the data for machine learning, the columns containing 'Yes' and 'No' values (i.e. `RainToday` and `RainTomorrow`) were transformed into boolean values 'True' and 'False'. Additionally, a dummies transformation was applied to the categorical variables to obtain an overall numerical dataset. However, this resulted in a total of 58 columns, which is not practical for future user inputs and requires significant computation. From a user's perspective, the features that are intuitively and practically easier to understand, even by visiting a weather website, were selected. So, as a final data pre-processing step, only 5 numerical columns have been kept: `MinTemp`, `MaxTemp`, `Rainfall`, `Humidity`, `WindSpeed`. `Humidity` and `WindSpeed` were obtained as mean values respectively between `Humidity9am` and `Humidity3pm`, and between `Humidity9am` and `Humidity3pm`.

The final datasets contains the 5 numerical columns, as said before, and the categorical columns `RainToday` and `RainTomorrow`.

3.5 Prediction Model

After the dataset was pre-processed, the prediction model was constructed by comparing only supervised learning models, specifically basic ones such as *Logistic Regression*, *Decision Tree*, *Random Forest*, *Support Vector Machines*, and choosing the "best" one at the end. The objective was to find the most suitable model for binary classification, in order to answer the question: *will it rain tomorrow or not?* All of the models were trained and validated first with the split dataset in 80%-20%, then with *k-Fold Cross Validation*. The models were evaluated based on their F1-score and efficiency (in terms of time), and the best-performing model was selected. The decision to use the F1-score and efficiency as evaluation metrics was based on the fact that the dataset was imbalanced, with over 70% of observations belonging to the "No" class for the `RainToday` and `RainTomorrow` variables. In such cases, evaluating models based on accuracy alone could result in emphasizing the model's ability to predict the majority class, which was "No" in this case.

The four models were first trained and validated, and a comparison of their performance based on several evaluation metrics is presented in Table 2 (with a truncation to 2 decimal places after zero, to avoid verbosity in the paper). Based on the evaluation metrics presented in the results section, *Random Forest* achieved the highest F1-score. Additionally, the accuracy of *Random Forest* was found to be almost as good as that of *Logistic Regression*, which had the best accuracy among the four models. Taking these metrics into account, it can be concluded that Random Forest is the best-performing model among the four.

Table 2. Comparison of performance metrics for different predictive models

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.83	0.67	0.37	0.48
Decision Tree	0.75	0.42	0.45	0.43
Random Forest	0.82	0.64	0.39	0.48
Linear SVM	0.81	0.73	0.15	0.25

The same process was applied to the models but with *5-Fold Cross Validation*, as can be seen in Table 3. This validation technique should allow for better estimation of the model's performance on new, unseen data by repeatedly splitting the data into training and validation sets, and evaluating the model on multiple validation sets.

Table 3. Comparison of performance metrics for different predictive cross-validated models

CV Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.82	0.68	0.37	0.48
Decision Tree	0.75	0.43	0.45	0.44
Random Forest	0.75	0.43	0.45	0.44
Linear SVM	0.75	0.43	0.45	0.44

In this comparison, the best model seems to be *Logistic Regression* instead. However, it can be noticed that average model accuracy ranges from 75% to 82%

Support Vector Machines models with polynomial and RBF kernels achieved an accuracy of approximately 80%. However, the linear model outperformed them in terms of F1-score, with a score of 0.25 compared to over 0.40 for the other models. So, only linear SVM is considered among all SVM models.

Given the software requirement to provide fast response times to users, the execution times of the two models were compared. The *Random Forest* model demonstrated a faster performance, with an execution time of 2.01 seconds, compared to the *Logistic Regression* model with cross-validation, which had a longer execution time (i.e. 2.23 seconds). As a result, the *Random Forest* model was selected based on its faster performance.

4. RESULTS

The resulting application[2] is very user-friendly, in terms of its interface, and only takes a couple of seconds to load everything. In fact, users can easily find what they need with just a few clicks, using the simple sidebar on the left-hand side of the screen. The sidebar allows users to switch between the three basic pages:

- the homepage, which provides a clear and concise overview of the application;
- the rain-specific prediction page;
- the page for visualizing historical data, which also provides many graphical tools, such as line charts and pie charts.

In this way, users always know in which part of the website they are. Moreover, the minimalist design of the front-end has the advantage of avoiding confusion.

At the outset, the stated objectives have been ultimately achieved. However, it is worth mentioning that some aspects can certainly be improved.

4.1 Proposal Limitations

Weather forecasting is a challenging task due to the complexity and dynamic nature of the system being predicted. While a higher accuracy would always be desirable, it is important to acknowledge the difficulty of weather forecasting and consider the trade-offs between accuracy and model complexity when designing the prediction model.

To further improve the *Random Forest* predictive model, some potential areas to explore include optimizing hyperparameters and increasing the size and diversity of the training data, while keeping the lowest possible execution time.

Additionally, it may be worth considering other predictive models, both supervised and unsupervised, that may be better suited to the type of meteorological data at hand, but this would also imply a deeper knowledge in Machine Learning topics.

Also the Agile methodology presented some challenges, including the need for skilled Agile team members and the difficulty of maintaining simplicity in the development process.

Certainly, it is possible to build a more aesthetically pleasing and modern user interface that includes links to other related websites or data.

5. CONCLUSION

Working on this project has been an engaging experience, and there is certainly room for improvement with further work. However, some challenges were encountered with the Streamlit framework. While the `requirements.txt` file was created in the GitHub repository to enable the application to function on the Streamlit website, there is still an outstanding issue with the weather forecast component that needs to be addressed. Unfortunately, this issue prevents the weather forecast from being obtained through the web application. Therefore, it is recommended to download the repository and run the application locally. Please note that the issue will persist until it is resolved.

Also structuring the UML class diagram was initially a challenging task, and it evolved over time in response to changes in the project.

As the author, I found focusing on the software functionalities to be the most interesting aspect of the project, particularly the construction of UML activity and use case diagrams, as well as the data pre-processing phase.

6. REFERENCES

[1] Kaggle, 2023, Weather dataset for data analysis (Version 1.0) [Dataset]. https://www.kaggle.com/datasets/rever3nd/weather-data

[2] https://marziapaschini-spproject-streamlit-app-h32tni.streamlit.app/