PredictStroke: stroke prediction based on biological and social factors

Rita Ortese
Università di Trieste
Trieste, Italy
Software Engineer/Project Manager
rita.ortese@studenti.units.it

Contents

1 Introduction

Modern machine learning techniques show great results in different fields of application such as engineering, physics, medical research, and so on.
PredictStroke is a project that has the aim to develop a Machine Learning-based system designed to assist specialists, such that neurologists, in identifying the leading causes of strokes, what to focus on in order to prevent them, and how likely an individual to suffer from a stroke.
Our idea moreover is to allow common people to individually check how likely they are to suffer from a stroke due to their conditions without the need of being an expert in the medical field. After the prediction is obtained, it is largely suggested to talk with an expert to receive a more specific diagnosis.
In this framework our prediction are obtained through machine learning models that are trained using continuous and categorical variables, allowing users to enter their data in such a way as to have a quick answer about the possibility of suffering from stroke in the short term. The project involves collecting and processing sensitive medical data. Specifically, the system requires:

- Demographic data of the user (age, bmi, hypertension, glucose level, etc.).
- Social data of the user (work type, marital status, residency type, etc.).

With this project, we want then to offer a fully functional web service powered by modern machine learning tools with the specific aim to help specialists or simple users identify their risk of suffering from a stroke given their current status.

The PredictStroke project was developed by a team of four members, each responsible for a specific aspect of the system:

- **Software Engineers / Project Managers**: Two members coordinated the project, managed development tasks, and ensured the machine learning model was properly integrated into the web application;
- **Software Developer**: One member developed the interactive interface and application logic using *Streamlit*, ensuring a smooth user experience;
- **Data Engineer**: One member handled data processing, model training, and ensured the dataset was well-structured for accurate predictions.

The project utilized **Git** for version control, ensuring efficient collaboration and code management. In the **Github** repository, a structured branching strategy was adopted to keep the main code stable while developing new features and fixing issues.

Regular team meetings helped coordinate work, review progress, and integrate components efficiently. As mentioned earlier, the application was built using **Streamlit** [1]. The trained models were saved as pickle files and loaded directly into the web app when needed, enabling smooth interaction and immediate use of the model for predictions.

2 Background

2.1 Software Process Model and Development Method

For the development of the PredictStroke project, we adopted the Agile process model [2] with an incremental development approach. Given the need for flexibility in refining our machine learning models, Agile allowed us to iteratively improve the system while continuously integrating feedback. This methodology was chosen due to the nature of our project, which required adaptability to evolving data, continuous refinement of predictive models, and seamless integration of new insights without disrupting the overall development process. Agile emphasizes adaptability, collaboration, and continuous refinement. Instead of following a rigid sequential approach, we divided the project into smaller increments, each developed, tested, and improved iteratively. This enabled us to rapidly adjust to emerging challenges, refine the machine learning models, and enhance the web interface based on real-time evaluations. The incremental approach was particularly beneficial for handling the uncertainty associated with medical data and ensuring that each component of the system was validated before proceeding to the next stage. Development was structured in short cycles, ensuring that each functionality was validated before moving forward. Frequent testing and evaluation allowed us to detect and resolve issues early, preventing major failures. Continuous integration and feedback loops ensured that improvements were seamlessly incorporated throughout the project. The use of Agile also enabled the team to better coordinate between different aspects of development, including machine learning, web deployment, and user interface design. By following Agile principles, we maintained a high degree of responsiveness, ensuring that the final system was both robust and adaptable to user needs. This iterative process also facilitated collaboration among team members, reducing development risks and ensuring that the final product met both technical and user requirements.

2.2 Gantt Chart

To manage and track project progress, we utilized a **Gantt Chart** [3]. This project management tool helped us visualize task dependencies, deadlines, and team member responsibilities over time. By structuring the workflow into overlapping tasks, we maintained flexibility and ensured continuous progress throughout the development process. The main benefits of using a Gantt Chart in our project included:

- Clear timeline representation of continuous and parallel development tasks;
- Continuous iteration and integration of components;
- Efficient collaboration between team members working on different aspects simultaneously;
- Quick adaptation to feedback, allowing for improvements during implementation.

The Gantt chart [1] was regularly updated to reflect task completion status, ensuring that we stayed on track with our initial development plan. The project activities were distributed across a two-week period (February 3rd to 14th, excluding the weekend, 8th and 9th), ensuring an incremental approach:

- **Day 1**: Team meeting, discussion of objectives, and dataset selection;
- **Day 2-3**: Data preprocessing – cleaning, feature selection, and transformation;
- **Day 3-4**: Initial model implementation–training, evaluation, and iterative improvements;
- **From Day 4 onward**:
  - Development of the web application using Streamlit, running in parallel with model refinements (adjusting hyperparameters, testing different architectures, and optimizing performance);
  - Continuous integration, testing, and debugging, refining machine learning models while ensuring seamless interaction with the web interface;
- **Final Day**: Refinements, final testing, deployment, and validation to ensure a seamless user experience.

By adopting Agile with incremental development, we ensured that workstreams overlapped, allowing early testing and continuous refinements throughout the project. This structured yet flexible approach facilitated collaboration, efficiency, and a high-quality final implementation of the PredictStroke project.

3 Methodology

3.1 Software Requirements

The PredictStroke project is designed to provide a web-based machine learning tool for stroke risk assessment. The software requirements are categorized into functional and non-functional requirements:

3.1.1 Functional Requirements.

- The system must allow users to input personal and medical data such as age, BMI, hypertension, glucose level, work type, and other relevant factors.
- The machine learning model should process the input data and return a likelihood for stroke risk.
- The web application must present results in an intuitive and comprehensible manner.

3.1.2 Non-Functional Requirements.

- The application must ensure data security and privacy compliance, avoiding sensitive data exposure;
- The web service should be accessible through multiple devices (desktop, tablet, mobile);
- The system must be scalable to handle an increasing number of users;
- The application should deliver predictions within 2 seconds, ensuring smoothness.

3.2 Software Development Methods

The project followed an incremental development approach with Agile principles to ensure continuous improvement and flexibility.

3.2.1 Development Strategy. The system was developed in parallel by different team members, with distinct responsibilities such as web development, data engineering, and machine learning model implementation. Python was used for system development, enabling efficient implementation of various components; modular design was adopted, allowing seamless integration of components.

Risk management played a crucial role in maintaining project stability and efficiency. To mitigate potential obstacles, tasks were broken into smaller, structured units, reducing bottlenecks and ensuring continuous progress. Regular team meetings were held to identify risks early and make necessary adjustments. One of the primary challenges was managing task delays and integration issues. These risks were addressed through contingency planning, workload reallocation, and flexible scheduling. Version control strategies ensured that changes were systematically reviewed, minimizing conflicts during integration. Additionally, frequent testing helped identify errors at an early stage, preventing critical failures. Data quality and model performance risks, such as class imbalance, were carefully managed. Techniques like SMOTE were applied for data balancing, while cross-validation helped mitigate overfitting. Emphasis was placed on recall as a performance metric to improve model reliability. Input validation measures were also implemented to maintain the clinical relevance of predictions and enhance system stability. Through proactive risk management and a structured development strategy, we ensured consistent progress toward a robust and dependable stroke prediction system.

3.2.2 Continuous Integration and Deployment. To maintain an efficient workflow, Continuous Integration (CI) and Continuous Deployment (CD) practices were used:

- **CI:** Team members committed their changes to a shared GitHub repository, following a structured branching strategy. Code was reviewed via pull requests before merging into the main branch. Manual testing ensured integration stability;
- **CD:** After successful integration, the latest version of the application was manually deployed to Streamlit for continuous user testing and feedback.

3.2.3 Version Control. Git and GitHub were used for version control, ensuring structured and collaborative development. A branching strategy was followed:

- `main` branch: Stable version of the project;
- `data_engineer` branch: Data preprocessing, model training, and evaluation;
- `website` branch: web development.

Feature-specific branches were created and merged into the appropriate branches after review.

3.3 Software Test

Testing was a crucial step in ensuring the system's reliability and accuracy. During development and evaluation, Key Performance Indicators (KPIs) were established to measure success, track progress, and identify potential issues early. Particular emphasis was placed on recall, as minimizing false negatives was essential to prevent the misclassification of stroke cases. A high recall ensured that at-risk patients were correctly identified, reducing the likelihood of overlooking urgent cases. Additionally, key metrics such as model accuracy, F1-score, and response time were continuously monitored to assess overall performance. The Area Under the Curve (AUC) was also considered to evaluate the balance between sensitivity and specificity. By consistently tracking these KPIs, we were able to detect performance issues early and implement necessary optimizations to enhance the system.

3.3.1 Testing Plan.

- **Integration Testing:** Verified the seamless interaction between the web interface and the machine learning model, through Streamlit;
- **Performance Testing**: Assessed system response time and load handling capacity to ensure a smooth user experience;
- **Model Validation**: Evaluated the effectiveness of predictions using standard metrics like precision, recall, and F1-score.

A specific anomaly detection check was not implemented, as users can only select values within a predefined range, making additional error handling for extreme or invalid inputs unnecessary.

3.3.2 Testing Strategy.

- **Manual test**: it was implemented where possible, particularly for model performance evaluation;
- **Manual testing**: it was performed for UI components and user interactions;
- **Unit testing**: Core functions, such as data normalization, feature extraction, and inference computations, were unit tested to maintain prediction accuracy across different input variations and deployment environments.

Through a structured development approach, rigorous version control, and thorough testing strategies, we ensured the robustness, performance, and reliability of the PredictStroke application, meeting both functional and non-functional requirements.

4 MLOps Approaches and Results

The project adhered to MLOps best practices to ensure the robustness of model training, evaluation, and deployment. The dataset used, Stroke Prediction Dataset [4], consisted of a total of 5109 samples, with 1278 allocated for testing, representing 25% of the data, while the remaining samples were used for training. To address class imbalance and improve model performance, oversampling [5] was applied exclusively to the training set, increasing the number of training samples to 7284. A total of 10 features were selected for training, ensuring that the model was trained on the most relevant data. Before proceeding with model evaluation, **K-Fold Cross Validation** [6] was conducted to minimize errors and ensure reliable generalization across different datasets. Various models, including Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine (SVM), were implemented and tested, and their performance metrics were carefully analyzed. Without applying oversampling, key metrics such as Recall and F1-score were found to be near zero, indicating an inability to correctly classify the minority class. However, when oversampling was incorporated, these metrics showed significant improvement, justifying its use in the training process. Following an extensive evaluation of the models, the **Random Forest** model with oversampling emerged as the best-performing approach, achieving an accuracy of 96%. The final model results were stored for future use in the application. The trained models were subsequently integrated into an application designed for user interaction. Users can input relevant data into the system and choose the model they wish to use for prediction. The output is a binary classification, indicating whether an individual, based on their provided characteristics, is at risk or not at risk of stroke. This approach ensures transparency and usability, allowing end-users to make informed decisions based on the model's predictions.

5 Conclusion and Limitations

In PredictStroke project, we developed a predictive model for stroke risk assessment using an MLOps approach. Through data preprocessing, oversampling, and rigorous model evaluation, we built a system that allows users to input data and select a model, providing a binary classification of stroke risk. This tool has the potential to assist in early stroke detection and could be integrated into medical environments for further use.

However, the project has several limitations. Due to privacy concerns, real patient data input is not available, and external user testing has not been conducted. Implementing the system in a real-world medical setting would require patient consent and a structured data collection process. Another major limitation is the lack of interpretability; while the model predicts stroke risk, it does not provide insights into the underlying causes or patient-specific factors. Future work could explore more complex approaches, such as CNNs or additional predictive features, to further improve accuracy. Despite these challenges, this system serves as a foundational tool that can be expanded and refined for potential integration into clinical decision-making.

References

[1] Streamlit. Streamlit - the fastest way to build and share data apps, 2024.
[2] Agile Alliance. Manifesto for agile software development, 2001.
[3] HubSpot. The ultimate gantt chart cheat sheet, 2017.
[4] Stroke prediction dataset, kaggle, 2021.
[5] Medium, random over sampler.
[6] Medium, k fold cross validation explained.

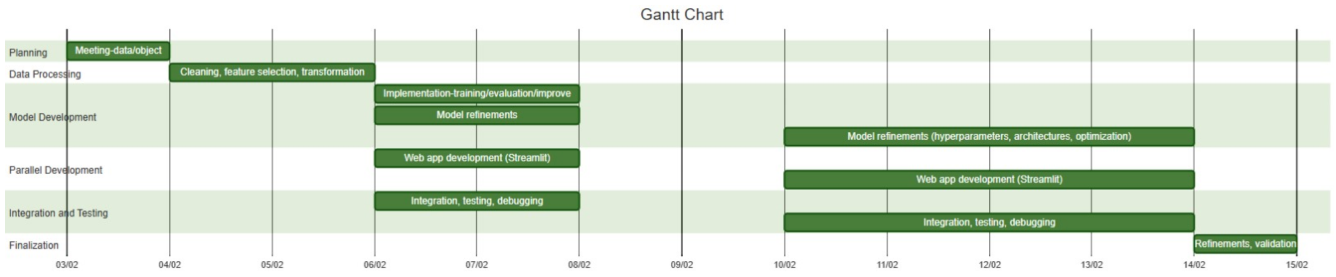

Figure 1. Gantt chart