
**THIS PROJECT REPORT HAS BEEN WRITTEN AND SUBMITTED AS PARTIAL FULLFILLMENT FOR THE BACHELOR OF INFORMATION TECHNOLOGY**

**KCA UNIVERSITY**

**FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY**

# SYSTEM IMPLEMENTATION PLAN: DISEASE PREDICTION SYSTEM

**Submitted by:**

OMONDI SANTOS OKELLO

REG: 19/05036

**Supervised by: MR COLLINS ONDIEK**

**BSc. INFORMATION TECHNOLOGY**

# TABLE OF CONTENT



# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

Disease prediction is a critical aspect of healthcare that has the potential to save lives, reduce healthcare costs, and improve patient outcomes. With the increasing amount of medical data being generated, the need for accurate and efficient disease prediction systems has never been greater. However, despite advances in medical technology, traditional disease prediction methods still face several limitations, including a lack of accuracy and inefficiency. This is why we have developed a cutting-edge disease prediction system that leverages the power of machine learning algorithms to analyze patient data and accurately predict the likelihood of disease. Our system integrates with existing healthcare infrastructure, including electronic health record (EHR) systems, and provides real-time predictions to healthcare providers. The purpose of this system

is to provide an accurate and efficient tool that can help healthcare providers prevent and predict diseases, leading to improved patient outcomes and reduced healthcare costs. In this paper, we will provide a comprehensive overview of the disease prediction system, including its key components, features, and benefits. Our aim is to demonstrate the significance of this system and its potential impact on healthcare. We believe that our disease prediction system has the potential to revolutionize the way healthcare providers approach disease prevention and prediction, ultimately improving patient outcomes and saving lives.

## 1.2 Goals and objectives

The goals and objectives of this test plan is to measure testing progress and verify that testing activity is consistent with project objectives.

Goals and objectives are grouped into the following categories:

**Functional correctness.**

Validation that the application correctly supports required Hive design application processes and transactions. List all the smart health application processes that the application is required to support. Also list any standards for which there is required compliance. Some of the objectives under this category includes the following

- Ability to identify a duplicate user that is about to be created by the system.
- Ability to make accurate disease prediction-based symptoms submitted by patient.
- Ability to find the correct doctor based on the doctor's profile that is searched
- Ability for system users to give their feedback on how they fill about the system's experience

**Authorization.**

Verification that actions and data are available only to those users with correct authorization. Here the goals and objectives are to identify key authorization requirements that must be satisfied, including access to functionality and data. In detail they include the following

- Ability to sign in only users that are registered by the system.
- Ability to register new users and determine their access levels.

**Service level.**

This is the verification that the system will support the required service levels of the business. This includes system availability, load, and responsiveness.

**Usability.**

This the validation that the application meets required levels of usability. This include the required training level needed for users to be able to use the system effectively.

### 1.3 Statement of scope

The scope of this test plan is to define what areas the disease prediction system is supposed to be tested what functionalities to mainly focus on, what types of bugs customers are interested in and what features of the system that will not be tested by any means

#### I. Features

Each test contains at least one feature, each feature describes an area of a product, it describes a process of the disease prediction system, particular to a functionality to be tested. The features to be tested here include

- Data sources: The types of patient data that the system is designed to analyze, such as EHR data, demographic data, and lifestyle data.
- Prediction algorithms: The specific algorithms used to analyze patient data and generate predictions, including machine learning algorithms and statistical models.
- Integration with existing systems: The existing healthcare systems that the disease prediction system is designed to integrate with, such as EHR systems.
- User interface: The type of interface that will be provided to healthcare providers to access and interpret the predictions.
- User access and authorization: The levels of user access and authorization that will be required to access the disease prediction system and patient data.
- Data privacy and security: The measures that will be taken to ensure the privacy and security of patient data, including secure data storage and transmission methods.
- Predictive accuracy: The target accuracy level for disease predictions, including the acceptable error rate.

### 1.4 Major constraints

- Data availability: The availability and quality of patient data, including the need for comprehensive and up-to-date data to ensure accurate predictions.
- Data privacy and security: The need to ensure the privacy and security of patient data, including compliance with regulations such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA).
- Integration with existing systems: The need to integrate the disease prediction system with existing healthcare infrastructure, including EHR systems, and the challenges of integrating with legacy systems.
- User adoption: The need to ensure that healthcare providers are trained and willing to use the disease prediction system and trust its predictions.
- Predictive accuracy: The need to achieve a high level of accuracy in disease predictions, while also balancing the risk of false positives and false negatives.
- Technical expertise: The need for technical expertise in machine learning and healthcare to develop, implement, and operate the disease prediction system.
- Funding and resources: The need for funding and resources, including hardware, software, and personnel, to develop and operate the disease prediction system.

# CHAPTER 2: IMPLEMENTATION PLAN

### 2.1 Software to be Implemented

The software to be Implemented here will be a disease prediction system. This system will be web based, that is able to be accessed from any device that has internet connection.

Name of the website:  Hivedesign.de

### 2.2 Model Assumptions

Here I outline of model design of the approach I will take in the software implementation life cycle that clearly shows the implementation objectives of the system. This is to guide to define the test coverage and testing scope to clear the picture of the project at any instance, to ensure a implementation activity is not missed.

#### 2.2.1 Choose algorithm (NAÏVE BAYES)

Naive Bayes is a type of machine learning algorithm that is based on Bayes' theorem, which states that the probability of a hypothesis (such as a disease diagnosis) given some evidence (such as patient data) is proportional to the prior probability of the hypothesis and the likelihood of the evidence given the hypothesis.

In the context of disease prediction, Naive Bayes can be used to predict the probability of a disease diagnosis based on patient data, such as demographic information, medical history, and laboratory test results. The algorithm assumes that the features in the patient data are independent of each other, which is known as the "naive" assumption.

Naive Bayes can be a useful algorithm for disease prediction in certain scenarios, such as when the dataset is small or the relationship between the features is not well understood. It is also relatively fast and simple to implement, making it a good choice for applications with limited computational resources.

However, it is important to keep in mind that the performance of Naive Bayes can be impacted by the validity of the "naive" assumption, and it may not be the best choice for datasets with complex relationships between the features. The development team may need to evaluate multiple algorithms to determine which one is best suited for the system's goals and objectives.

#### 2.2.2 Train model

Involves the following steps

- Prepare the data: The first step is to gather and preprocess the patient data that will be used to train the model. This may involve cleaning the data, imputing missing values, and normalizing the features.
- Establish the prior probabilities: The next step is to establish the prior probabilities of each disease class in the training data. This can be done by counting the number of instances of each class in the training data and dividing by the total number of instances.
- Establish the likelihood probabilities: For each feature in the patient data, the likelihood probabilities of each feature given each disease class need to be calculated. This can be done

by counting the number of instances of each feature in the training data for each class and dividing by the total number of instances of each class.

- Train the model: The model is then trained using the prior and likelihood probabilities. This involves inputting new patient data into the model and using the probabilities to calculate the posterior probabilities of each disease class.
- Evaluate the model: Once the model has been trained, it is important to evaluate its performance on a separate dataset, such as a validation or test set. This can be done by comparing the model's predictions to the actual disease outcomes and calculating metrics such as accuracy, precision, and recall.

It is important to note that the performance of a Naive Bayes model can be impacted by the quality of the training data and the validity of the "naive" assumption. The development team may need to iterate on the model and the training data to improve its performance.

### 2.2.3 Evaluate the model

Evaluating a Naive Bayes model involves comparing its predictions to the actual disease outcomes in a separate dataset, such as a validation or test set. The following metrics can be used to evaluate the performance of the model:

- Accuracy: The accuracy of the model is the proportion of correct predictions made by the model. It is calculated as the number of correct predictions divided by the total number of predictions.
- Precision: Precision is the proportion of true positive predictions made by the model, i.e. the proportion of instances that the model correctly predicted as having the disease.
- Recall: Recall is the proportion of actual positive instances that the model correctly predicted, i.e. the proportion of instances with the disease that the model correctly identified.
- F1 Score: The F1 score is a composite metric that combines precision and recall into a single value. It is the harmonic mean of precision and recall, and is a good measure of the overall performance of the model.

It is also important to consider the confusion matrix when evaluating the performance of a Naive Bayes model. The confusion matrix shows the distribution of actual and predicted disease outcomes, and can help identify areas where the model may be underperforming.

### 2.2.4 Refine Model

- Improve the training data: The quality of the training data can have a significant impact on the performance of the model. The development team may need to gather additional data or preprocess the data further to improve its quality.
- Experiment with different prior probabilities: The prior probabilities used to train the model can impact its performance. The development team may need to experiment with different prior probabilities to see which ones result in the best performance.

- Modify the "naive" assumption: The "naive" assumption of independence between features can impact the performance of the model. The development team may need to modify the assumption, for example by incorporating interactions between features into the model.
- Try different algorithms: The Naive Bayes algorithm may not be the best choice for the dataset and the system's goals and objectives. The development team may need to evaluate other machine learning algorithms to determine which one is best suited for the task.
- Regularize the model: Overfitting can occur when the model is too complex and has too many parameters. The development team may need to regularize the model to prevent overfitting, for example by adding a regularization term to the loss function used to train the model.

.

EXAMPLE OF NAÏVE BAYES OUTLAW:

An example of using Naive Bayes for disease prediction is a system that predicts the likelihood of a patient having a specific disease based on their symptoms. The system can use Naive Bayes to make predictions based on a set of binary features that represent the presence or absence of a particular symptom.

For example, the system might use features such as "cough," "fever," "headache," "shortness of breath," and so on. The Naive Bayes algorithm would then use the probabilities of these symptoms occurring given the presence or absence of the disease to make a prediction.

In this example, the development team would first gather a dataset of patients with known disease outcomes and their symptoms. The dataset would be used to train the Naive Bayes model and estimate the probabilities required for making predictions.

Once the model is trained, it can be used to make predictions for new patients. For example, if a patient reports symptoms of a cough and a fever, the system could use the Naive Bayes algorithm to estimate the probability of the patient having a specific disease based on these symptoms.

### 2.3 Implementation Timeline

1. Week 1-2: Requirements gathering and data collection
- goals, objectives, and requirements of the system
- Collecting and preprocessing the data that will be used to train and test the Naive Bayes model
2. Week 3-4: Model training
- Training the Naive Bayes model using the preprocessed data
- Store the trained model for later use
3. Week 5-6: Model evaluation
- Evaluating the performance of the trained Naive Bayes model using metrics such as accuracy, precision, recall, and the confusion matrix
- Refining the model as needed to improve its performance
4. Week 7-8: System integration
- Integrating the trained Naive Bayes model into the overall system architecture
- Developing a user interface for the system
5. Week 9-10: Deployment and testing
- Testing the end-to-end system to ensure it meets the requirements and is working as expected
- Deploying the system into a production environment
6. Week 11-12: Ongoing maintenance and monitoring
- Monitoring the system's performance to ensure it is operating as expected
- Updating the model as needed to improve its accuracy
- Fixing any bugs that arise.

### 2.4 Implementation Plan Milestone

- Data Collection and Preprocessing: Collect and preprocess the medical data that will be used to train and test the Naive Bayes model.
- Model Development: Develop and train the Naive Bayes model using the preprocessed data.
- Model Evaluation: Evaluate the performance of the trained Naive Bayes model using metrics such as accuracy, precision, recall, and the confusion matrix. Refine the model as needed to improve its performance.
- System Integration: Integrate the trained Naive Bayes model into the overall system architecture, including the development of a user interface.
- Testing and Deployment: Test the end-to-end system to ensure it meets the requirements and is working as expected. Deploy the system into a production environment.
- Ongoing Maintenance: Monitor the system's performance to ensure it is operating as expected. Update the model as needed to improve its accuracy and fix any bugs that arise.
- User Training: Provide training for users on how to use the system effectively.
- Launch: Launch the disease prediction system to the public.

## 2.5 Team Role and Responsibilities

- Project Manager: The project manager is responsible for overseeing the overall implementation plan, ensuring that it stays on track, and managing the resources needed to carry out the plan.
- Data Scientist: The data scientist is responsible for collecting, preprocessing, and analyzing the medical data that will be used to train and test the Naive Bayes model. They are also responsible for developing and refining the model as needed.
- Software Developer: The software developer is responsible for integrating the trained Naive Bayes model into the overall system architecture and developing the user interface. They are also responsible for testing the end-to-end system and fixing any bugs that arise.
- User Trainer: The user trainer is responsible for providing training to users on how to use the system effectively.
- Technical Writer: The technical writer is responsible for documenting the implementation plan, including the requirements, milestones, and roles and responsibilities of the team members.

## 2.6 Implementation Plan Metrics

- Accuracy: The accuracy of the Naive Bayes model is a key metric, as it measures how well the model is able to correctly predict the disease.
- Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model.
- Recall: Recall measures the proportion of true positive predictions among all actual positive cases.
- F1 Score: The F1 Score is a weighted average of precision and recall and provides a single metric that balances both.
- User Satisfaction: User satisfaction is another important metric, as it measures the level of satisfaction among users of the system. This can be measured through user surveys or through monitoring usage patterns.
- Time to Predict: Time to predict is the amount of time it takes for the model to make a prediction, and it is an important metric for assessing the performance of the system in real-world use.
- False Positive and False Negative Rates: The false positive and false negative rates are important metrics as they measure the proportion of incorrect predictions made by the model.
- Model Robustness: Model robustness is a measure of the model's ability to perform well on unseen data and can be evaluated by testing the model on a validation set.

## 2.7 Budget

| | Data collection | Hardware | Software | Personnel | 3rd party services | Testing & Validation | Maintanace & Upgrade |
|---|---|---|---|---|---|---|---|
| Name | Public H | HP | Postgresql, Pycharm | 3technical individuals | Hosting etc | Collecting user info | Domain renewal etc |
| Cost | ***** | 50,000 | 30,000 | 60,000 | 15,000 | 5,000 | 20,000 |
| | | | | | | | |

## 2.8 Monitor progress

- Cross-validation: Cross-validation is a method to evaluate the performance of the model on new data by partitioning the data into training and validation sets. This method helps to prevent overfitting and provides a measure of the model's generalization performance. By using cross-validation, we can measure the accuracy of the model and track its progress over time.
- Performance metrics: Performance metrics such as precision, recall, and F1-score can be used to evaluate the accuracy of the model. These metrics provide a quantitative measure of the model's performance and help to identify areas that need improvement.
- Monitoring user feedback: Monitoring user feedback can provide valuable insights into the performance of the system. User feedback can be collected through surveys, feedback forms, or user testing sessions. By analyzing user feedback, we can identify areas where the system needs improvement and make necessary changes to improve the system's performance.
- Monitoring data quality: The accuracy and reliability of the system depend on the quality of the data used to train the model. Therefore, it is essential to monitor the quality of the data used and ensure that it is up-to-date and relevant. Regularly updating the data and cleaning the data can help to improve the accuracy and reliability of the system.
- Monitoring system downtime and errors: Monitoring system downtime and errors can help to identify and address issues that may affect the system's performance. Regularly monitoring the system's logs and error messages can help to identify and fix issues quickly, thereby improving the system's uptime and performance.

## 2.9 Evaluate the project

- Accuracy: The accuracy of the model is the most critical factor in evaluating the project. We can measure the accuracy of the system by comparing the predicted results with the actual results. If the predicted results are accurate, it indicates that the model is reliable.
- Precision and Recall: Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positive cases. Precision and recall are important measures of the system's ability to correctly identify patients who have the disease.
- User feedback: User feedback can provide valuable insights into the system's usability and effectiveness. Feedback can be collected through surveys or interviews with users of the system. The feedback can help identify areas of improvement, such as the user interface, the system's performance, and the quality of the predictions.
- Performance metrics: Performance metrics such as the F1-score, Area Under the Receiver Operating Characteristic Curve (AUC-ROC), and Mean Absolute Error (MAE) can provide a quantitative measure of the system's performance. These metrics can help identify areas that require improvement.
- Data quality: The quality of the data used to train the model is crucial for the accuracy of the system. Therefore, we need to ensure that the data used for training the model is relevant, up-to-date, and accurate.
- Deployment: The system's ability to be deployed and integrated into the existing healthcare infrastructure is essential for its success. Therefore, we need to evaluate the system's deployment process, including installation, maintenance, and support.

### 2.9.1 Conclusion

In conclusion, disease prediction systems have the potential to revolutionize healthcare by providing early detection and intervention for various diseases. These systems use various algorithms, including Naive Bayes, to analyze patient data and predict the likelihood of a disease. Naive Bayes algorithm is a simple and efficient algorithm that can be used for disease prediction. It is particularly suitable for large datasets with many features and has been shown to be effective in various disease prediction tasks. However, its assumptions regarding feature independence may limit its performance in some cases.

The accuracy and reliability of a disease prediction system that uses Naive Bayes algorithm can be monitored by regularly evaluating its performance on new data, collecting user feedback, monitoring data quality, and assessing its deployment process.

Overall, disease prediction systems have the potential to improve healthcare outcomes by enabling early detection and intervention for various diseases. Naive Bayes algorithm is one of the algorithms that can be used in disease prediction systems, and its performance can be monitored by regularly evaluating its accuracy and reliability. However, it is important to carefully consider the features used in the model and its assumptions, and to use it in conjunction with other algorithms and methods for disease prediction to improve its accuracy and reliability.