Breast Cancer Detection

Contents

[1 Technical Design Document 2](#_Toc44177265)

[1. Introduction 5](#_Toc44177266)

[High level objectives 5](#_Toc44177267)

[2 Workflow Overall 6](#_Toc44177268)

[Application Flow 6](#_Toc44177269)

[Exception Scenarios Overall 6](#_Toc44177270)

[3 Workflow Data Ingestion and File Conversion 8](#_Toc44177271)

[3.1 Technical solution design 10](#_Toc44177272)

[3.2 Exceptions Scenarios 10](#_Toc44177273)

[4 Graph-Based EDA 12](#_Toc44177278)

[4.1 Technical solution design 12](#_Toc44177279)

[4.2 Exceptions Scenarios Module Wise 13](#_Toc44177280)

[6 EDA using Sweetviz library 14](#_Toc44177281)

[6.1 Technical solution design 14](#_Toc44177282)

[6.2 Exceptions Scenarios Module Wise 14](#_Toc44177283)

[7 Data Transformers( Pre-processing steps) 15](#_Toc44177284)

[7.1 Technical solution design 15](#_Toc44177285)

[7.2 Exceptions Scenarios Module Wise 15](#_Toc44177286)

[8 ML Model Selection 16](#_Toc44177287)

[8.1 Technical solution design 16](#_Toc44177288)

[8.2 Exceptions Scenarios Module Wise 16](#_Toc44177289)

[9 Model Tuning and Optimization 17](#_Toc44177290)

[9.1 Technical solution design 18](#_Toc44177291)

[9.2 Exceptions Scenarios Module Wise 18](#_Toc44177292)

[10 Testing Modules 19](#_Toc44177293)

[10.1 Technical solution design 19](#_Toc44177294)

[10.2 Exceptions Scenarios Module Wise 20](#_Toc44177295)

[11 Prediction Pipeline 21](#_Toc44177296)

[11.1 Technical solution design 22](#_Toc44177297)

[11.2 Exceptions Scenarios Module Wise 23](#_Toc44177298)

[12 Deployment Strategy 24](#_Toc44177299)

[12.1 Technical solution design 25](#_Toc44177300)

[12.2 Exceptions Scenarios Module Wise 26](#_Toc44177301)

[13 Monitoring 27](#_Toc44177302)

[13.1 Technical solution design 27](#_Toc44177303)

[13.2 Exceptions Scenarios Module Wise 27](#_Toc44177304)

[14 Logging 28](#_Toc44177305)

[14.1 Technical solution design 28](#_Toc44177306)

[14.2 Exceptions Scenarios Module Wise 28](#_Toc44177307)

[15 Hardware Requirements 29](#_Toc44177308)

[15.1 Requirements for model training 29](#_Toc44177309)

[15.2 Requirements for model testing 29](#_Toc44177310)

# Technical Design Document

Version 1.0

Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 18th July 2020 | 1.0 | Initial Draft | Jane Alam  Hemant Gautam |

Contributors

The content of this document has been authored with the combined input of the following group of key individuals.

|  |  |
| --- | --- |
| Name | Section Worked Upon |
| Jane Alam  Hemant Gauam | Initial Draft |

# Introduction

The goal here is to build an end to end Machine Learning solution for breast cancer detection where the user will only give the data(either by adding single patient data or file upload) and the result will be the shown wheather the cancer type is M(1) = malignant or B(0) = benign.

This project shall be delivered in one phase:

Phase1: Integration of UI to all the functionalities(predicting the result).

The technical design document gives a design blueprint of the Breast Cancer detection project. This document communicates the technical details of the solution proposed.

In addition, this document also captures the different workflows involved to build the solution, exceptions in the workflows and any assumptions that have been considered.

Once agreed as the basis for the building of the project, the flowchart and assumptions will be used as a platform from which the solution will be designed.

**Note: All the code will be written in python version 3.7**

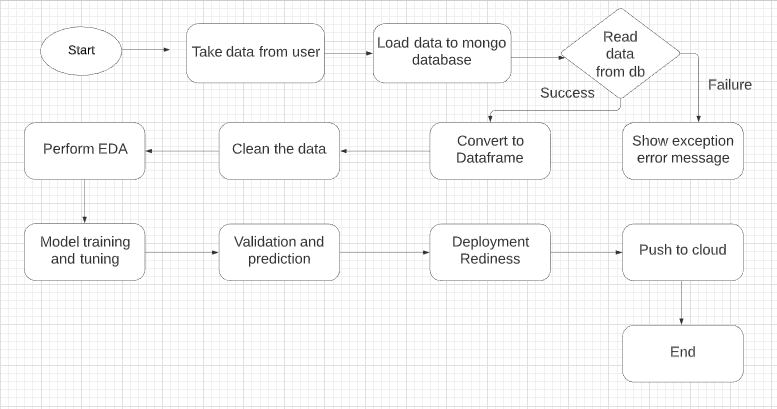
## High level objectives

The high-level objectives are:

1. Enable loading of csv data into mongo db database. And convert that data into dataframe.
2. Perform graphical analysis for the data and Showcase the results (graphs) on the screen.
3. Perform data cleaning operation with all the steps required and showcase a report on screen.
4. After data cleaning showcase the graphical analysis once again for comparison.
5. Choose the appropriate ML model for training.
6. Perform model Tuning.
7. Create a list of top 3 models and show multiple metrics for them.

# Workflow Overall

## Application Flow



## Exception Scenarios Overall

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Unable to read data from database | Give proper error message | Database credentials needs to be checked |

# Workflow Data Ingestion and File Conversion

**Data Sources:**

|  |  |
| --- | --- |
| Data Connector Utils | File Conversion Utils |
| [Mongo](https://help.tableau.com/current/pro/desktop/en-us/examples_access.htm) DB | CSV |

## Technical solution design



## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataGetter** |  |
| Method Name | read\_data\_from\_mgdb |  |
|  | Method Description | This method will be used to read data from a csv file |
|  | Input parameter names | self,username, password, dbname |
|  | Input Parameter Description | username: admin username of the database  password: admin password of the database  dbname: name of the db to connect with |
|  | ouptput | DB connection object |
|  | On Exception | Log connection failure exception message in log file |

## Exceptions Scenarios

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Unable to read data from database | Give proper error message | Database credentials needs to be checked |

# Data Profiling

After reading the data, automatically the following details should be shown:

1. The number of rows
2. The number of columns
3. Total missing values and it’s percentage
4. Number of categorical columns and their list
5. Number of numerical columns and their list
6. Number of duplicate rows

## Method Definition

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataProfiler** |  |
| Method Name | get\_data\_profile |  |
|  | Method Description | This method will be used to give various insighst about data. |
|  | Input parameter names | self, dataframe |
|  | Input Parameter Description | dataframe: the inpt data just loaded from source |
|  | ouptput | 1. The number of rows 2. The number of columns 3. Total missing values and it’s percentage 4. Number of categorical columns and their list 5. Number of numerical columns and their list 6. Number of duplicate rows |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

# EDA

Create the following graphs:

**MVP:**

Correlation Heatmaps

Check for balance/imbalance

**Phase1:**

Count plots

Boxplot for outliers

Piecharts for categories

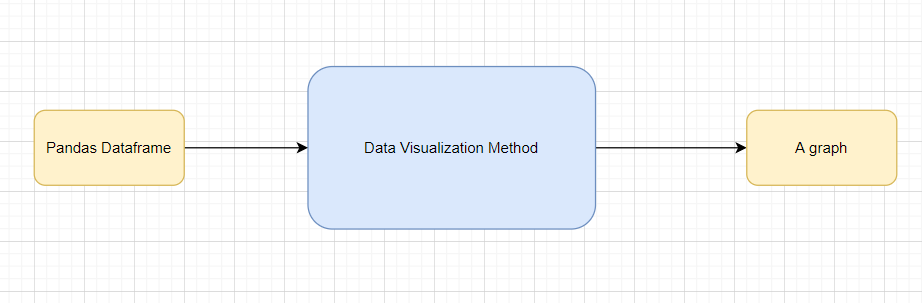
Line charts for trends

Barplots

KDE Plots

Scatterplot

## Technical solution design



## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataVisualization** |  |
| Method Name | read\_data\_from\_csv |  |
|  | Method Description | This method will be used to read data from a csv file or a flat file |
|  | Input parameter names | self,file\_name, header,names, use\_cols, separator |
|  | Input Parameter Description | file\_name: name of the file to be read  header: Row number(s) to be used as column names  names : array-like, optional  List of column names to use. If file contains no header row, then you  should explicitly pass ``header=None``.  Use\_cols: To load a subset of columns  Separator: Delimiter to use |
|  | ouptput | A pandas Dataframe |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong input to the methods | Handle Internally | Code should never give a wrong input |

# EDA using sweetviz library

# Data Transformers( Pre-processing steps)

**MVP:**

Null value handling

Categorical to numerical

Imbalanced data set handling

Handling columns with std deviation zero or below a threshold

Normalisation

PCA

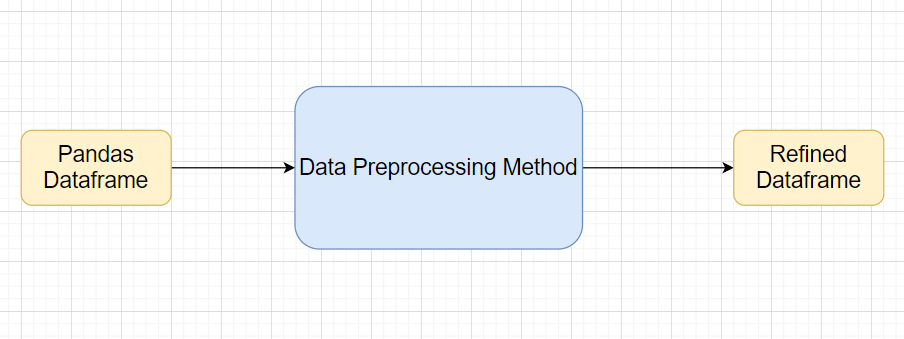
**Phase1:**

Outlier detection

Data Scaling/ Normalisation

Feature Selection: <https://scikit-learn.org/stable/auto_examples/index.html#feature-selection>

## Technical solution design



## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataPreprocessor** |  |
| Method Name | impute\_missing\_values |  |
|  | Method Description | This method will be used to read data from a csv file or a flat file |
|  | Input parameter names | self,file\_name, header,names, use\_cols, separator |
|  | Input Parameter Description | file\_name: name of the file to be read  header: Row number(s) to be used as column names  names : array-like, optional  List of column names to use. If file contains no header row, then you  should explicitly pass ``header=None``.  Use\_cols: To load a subset of columns  Separator: Delimiter to use |
|  | ouptput | A pandas Dataframe |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# ML Model Selection

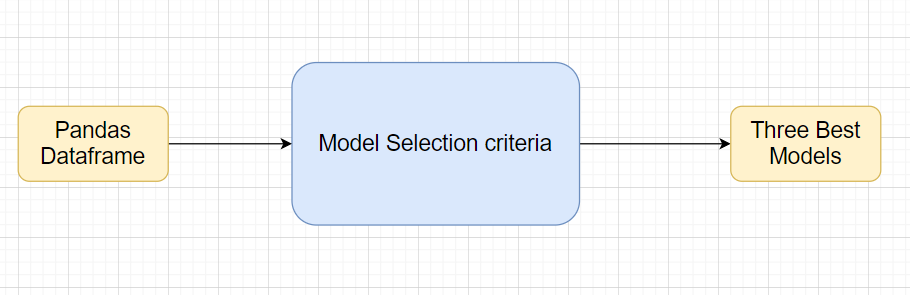
**MVP:**

3 Models— **Logistic Regression**, **Random Forest**, **SVM, KNN**

**Phase1:**

Model Selection criteria

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# Model Tuning and Optimization

**Note:** The data should have been divided into train and validation set before this.

**Classification:**

Logistic Regression

Decision Tree

Random Forest

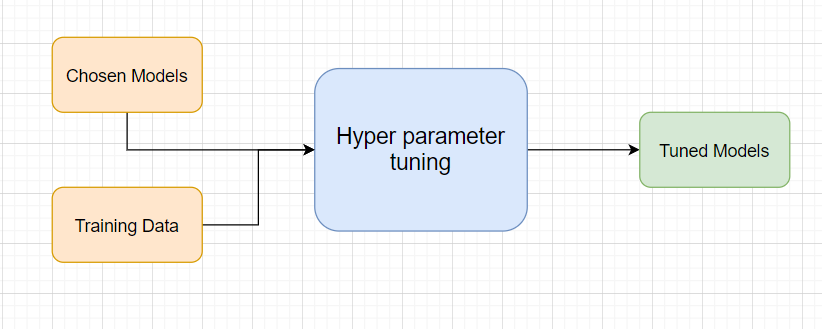
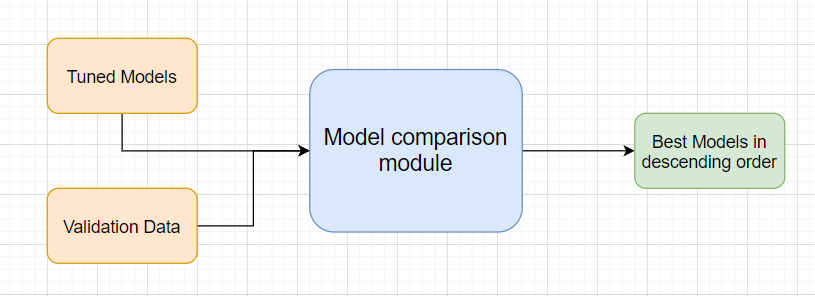
SVM

KNN Classifier

Model selection criteria:

Accuracy, AUC, Precision, Recall, F Beta

## Technical solution design

1. 
2. 

# Testing Modules

Divide the training data itself into train and test sets

Use test data to have tests run on the three best models

Give the test report

1. Accuracy
2. Precision
3. Recall
4. F Beta
5. Confusion matrix

**Note**: Save the best model after validation is completed.

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Number of Parameters do not match | Handle internally | Check the test data creation and verify the columns |
| Only once class present in test data | Handle Internally |  |

# Prediction Pipeline

Use the existing data read modules

Use the existing pre-processing module

Load the model into memory

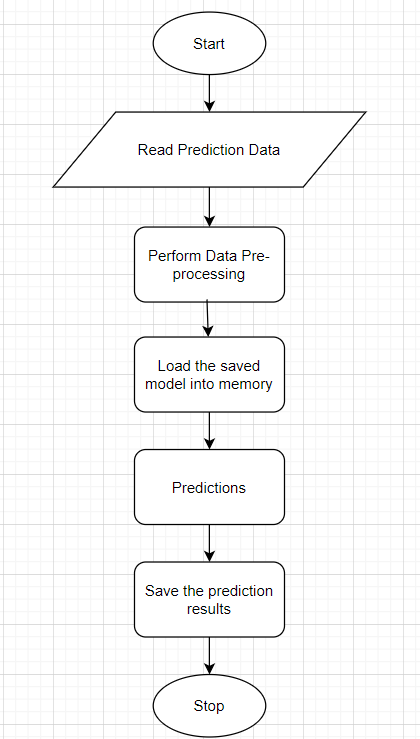
Do predictions

Store prediction results(show sample predictions)

Phase 2:

UI for predictions

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Columns don’t match in training and Prediction data | Show error message | The user enters the correct data |
|  |  |  |

# Deployment Strategy

Take the cloud name as input

Prepare the metadata files based on cloud

Phase 2:

Accept the user credentials

Prepare a script file to push changes

Docker instance

Push of the docker instance to cloud

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
|  |  |  |

# Logging

Separate Folder for logs

Logging of every step

Entry to the methods

Exit from the methods with success/ failure message

Error message Logging

Model comparisons

Training start and end

Prediction start and end

Achieve asynchronous logging

**Phase 2:**

Options for Logging in DB

Options for Log Publish

## Technical solution design



## Common Logging Framework Code

|  |  |
| --- | --- |
| Class Name | App Logger |
| Method Name | log |
| Method Description | This method will be used for logging all the information to the file. |
| Input parameter names | self,file\_object, log\_message |
| Input Parameter Description | file\_object: the file where the logs will be written  log\_message: the message to be logged |
| ouptput | A log file with messages |

## Exceptions Scenarios Module Wise

Ideally, the logging should never fail.

# Hardware Requirements

## Requirements for model training

The minimum configuration should be:

* 8 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor

## Requirements for model testing

The minimum configuration should be:

* 4 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor

# Sample code and standard to be followed:

Sample Code:

class Data\_Getter:  
 *"""  
 This class shall be used for obtaining the data from the source for training.  
  
 Written By: iNeuron Intelligence  
 Version: 1.0  
 Revisions: None  
  
 """* def \_\_init\_\_(self, file\_object, logger\_object):  
 self.training\_file='Training\_FileFromDB/InputFile.csv'  
 self.file\_object=file\_object  
 self.logger\_object=logger\_object  
  
 def get\_data(self):  
 *"""  
 Method Name: get\_data  
 Description: This method reads the data from source.  
 Output: A pandas DataFrame.  
 On Failure: Raise Exception  
  
 Written By: iNeuron Intelligence  
 Version: 1.0  
 Revisions: None  
  
 """* self.logger\_object.log(self.file\_object,'Entered the get\_data method of the Data\_Getter class') # Logging entry to the method  
 try:  
 self.data= pd.read\_csv(self.training\_file) # reading the data file  
 self.logger\_object.log(self.file\_object,'Data Load Successful.Exited the get\_data method of the Data\_Getter class') # Logging exit from the method  
 return self.data # return the read data to the calling method  
 except Exception as e:  
 self.logger\_object.log(self.file\_object,'Exception occured in get\_data method of the Data\_Getter class. Exception message: '+str(e)) # Logging the exception message  
 self.logger\_object.log(self.file\_object,  
 'Data Load Unsuccessful.Exited the get\_data method of the Data\_Getter class') # Logging unsuccessful load of data  
 raise Exception() # raising exception and exiting

Coding Standard:

1. Imports should usually be on separate lines
2. Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing.
3. Compound statements (multiple statements on the same line) are generally discouraged
4. Comments should be complete sentences. Always make a priority of keeping the comments up-to-date when the code changes. Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
5. Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
6. The name of the variables should start with small case capital letters and a multi word variable should be named as: word1\_word2\_word3.
7. The variable name should be appropriate based on the things that they do. DO NOT USE NAMES LIKE x, k, y etc. Always use a meaningful English word. For example, customer\_name, nearest\_neighbour etc.
8. Method names should start with small case characters. They should start with a verb and make a meaningful sense of what they are supposed to accomplish. For e.g.: load\_data\_from\_sql()
9. Always use self for the first argument to instance methods.
10. Class names should normally use the CapWords convention. Class name should also represent the functionality of the class. For e.g. DataLoader()
11. Modules/Packages/Folders should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. For e.g.: data\_ingestion
12. Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL.
13. Comparisons to singletons like None should always be done with is or is not, never the equality operators
14. The code should be properly enclosed withing try and exception blocks and the exceptions should be handled with proper error messages.
15. Additionally, for all try/except clauses, limit the try clause to the absolute minimum amount of code necessary. Again, this avoids masking bugs
16. When a resource is local to a particular section of code, use a with statement to ensure it is cleaned up promptly and reliably after use.
17. Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. If any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable)
18. Object type comparisons should always use isinstance() instead of comparing types directly
19. Don't compare boolean values to True or False using ==