Low Level Design (LLD)

Phishing Domain Detection

April 1, 2022

Revision number: 1.0

Last date of revision: April 1, 2022

Contents

Chapter 1

Introduction

1.1 Purpose of LLD

Return to table of contents

LLD provides internal design of actual program code for the phishing detection solution. This document describes class diagrams with methods and relations between classes and program specs. LLD is constructed in a way that a programmer can code the program from it.

1.2 Scope

Return to table of contents

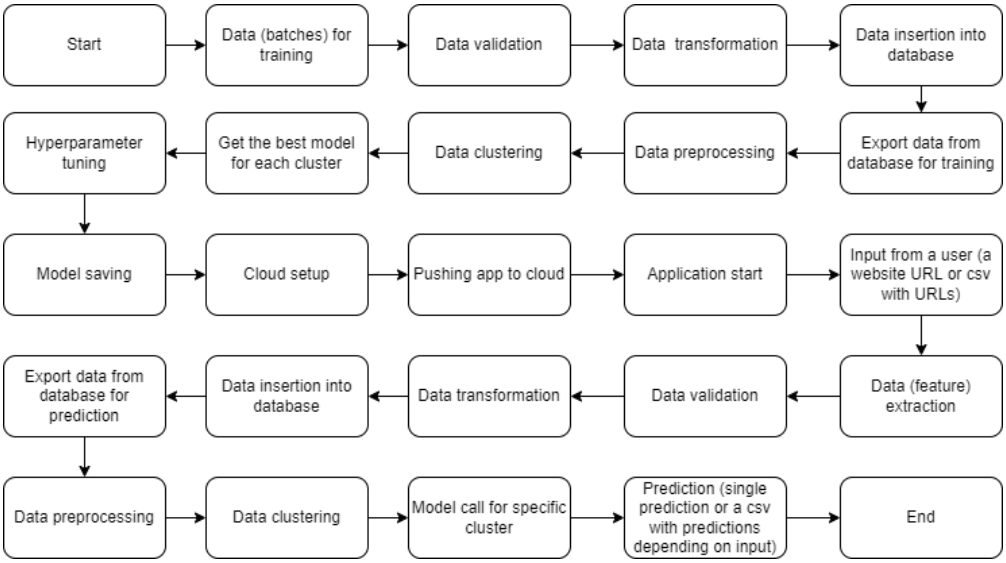
LLD is a component level design process which passes through a step-by-step refinement. The process can be used for designing data structures, required software architecture, source code, and performance algorithms. Data organization may be defined during requirement analysis and then refined during data design work.

Chapter 2

Architecture

2.1 Architecture diagram

Return to table of contents



Chapter 3

Architecture description

3.1 Data description

Return to table of contents

The data contains 88,647 observations of websites. Specifically 58,000 legitimate and 30,647 phishing website instances with 111 features. The data comes as a csv file.

3.2 Data validation

Return to table of contents

In this step, we perform different sets of validation on the given set of training files.

1. Name validation - we validate the name of the files based on the given name in a schema file provided. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to 'Good_Data_Folder' else we move such files to 'Bad_Data_Folder.'

2. Number of columns - we validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to 'Bad_Data_Folder.'

3. Name of columns - the name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to 'Bad_Data_Folder'.

4. The datatype of columns - the datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to 'Bad_Data_Folder'.

5. Null values in columns - if any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to 'Bad_Data_Folder'.

3.3 Data insertion into database

Return to table of contents

1. Database creation with the given name passed and connection (if the database exists, open connection to it)

2. Table creation in the database - tables with names 'phishing1' and 'phishing2', are created in

the database for inserting the files from the 'Good_Data_Folder' based on given column names and datatype in the schema file. We have created 2 tables because AstraDB has no capacity to save 111 features for a table, at least for free users. If the table is already present, then the new table is not created and new files are inserted into the already present table as we want training to be done on new as well as old training files. We have also created 'prediction_phishing1' and 'prediction_phishing2' tables to host data for prediction purpose. The prediction data has been created using website feature extraction codes that can be found in extract_features folder.

3. Insertion of files in the table - all the files in the 'Good_Data_Folder' are inserted in the above-created tables. If any file has invalid data type in any of the columns, the file is not loaded in the tables and is moved to 'Bad_Data_Folder'.

3.4 Export data from database for training

Return to table of contents

The data is exported from the database for data preprocessing and model building.

3.5 Data preprocessing

Return to table of contents

1. Replace the invalid values with numpy "nan" so we can use imputer on such values.
2. Check for null values in the columns. If present, impute the null values using the KNN imputer.
3. Replace -1 values with -999 for continuous/numeric features.
4. Use one-hot encoding for categorical columns.

3.6 Data clustering

Return to table of contents

KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using 'KneeLocator' function. The idea behind clustering is to implement different algorithms.

The Kmeans model is trained over preprocessed data and the model is saved for further use in prediction.

3.7 Model building

Return to table of contents

Model building includes finding the best model for each cluster. We are using Support Vector Machine, XGBoost, Naive Bayes, Logistic Regression, Random Forest, and LightGBM algorithms. For each cluster, the algorithms use the best parameters derived from GridSearch/BayesSearch. We calculate the AUC scores for the models and select the model with the best score for each cluster. The best models for each cluster are saved for further use in prediction.

3.8 Input from a user

Return to table of contents

A user will input a website URL or a csv/xlsx file with URLs in it. Then 111 features are derived from each website URL.

3.9 Model call

Return to table of contents

To make a prediction a model will be called based on a cluster of a datapoint.

3.10 Deployment

Return to table of contents

The solution will be deployed to Heroku. CircleCI is used for CI-CD.

Chapter 4

Unit tests

4.1 Unit test cases

Return to table of contents

Testing has not been conducted.