CSCE 585: Machine Learning Systems: Course Project: Progress Report

Shima Oruji and Zach Thomas

October 3, 2022

Contents

1 Introduction

Diabetes Mellitus is among critical diseases and lots of people are suffering from it in recent years. According to the recent studies, age, obesity, lack of exercise, hereditary diabetes, living style, bad diet, high blood pressure, etc. can cause Diabetes Mellitus. People having diabetes have high risk of diseases such as heart disease, kidney disease, stroke, eye problem, and nerve damage. With the recent advancements in the field of machine learning (ML), several researchers have tried to apply ML models to perform Diabetes prediction in patients based on various factors. However, there is no rigorous and comprehensive study on the evaluation of different ML models to determine the best practices in this specific problem.

In this project, we aim at implementing and evaluating different classification methods (e.g., decision tree, random forest, support vector machine, and neural network) on the given dataset and determine which methods perform better and under which conditions. We will use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years. We will also develop a website and API, which can be leveraged by the users to perform the online inferrence from the best trained model. We are also planning to implement a website and API which can be used by the clients to perform online inference from the best trained model. The overall structure can be extended as a machine learning pipeline which gets updated and deployed regularly.

2 Problem Statement

The project has two main parts: 1) Machine learning, and 2) Website and API development. The first part is a binary classification problem (onset of diabetes as 1 or not as 0). As mentioned in the previous section, we will use the Pima Indians onset of diabetes dataset, which is a standard benchmark dataset for these studies. For the evaluation, we will use the accuracy, precision, and recall as general indicators of the performance of different classifiers. We are planning to tune each classifier with the best hyperparameters using the ROC curve and use it for comparison with other classifiers. In the second part, we will use an API and website to make the trained model publicly available to clients whre they can directly enteract with the best trained model to predict Diabetes.

3 Technical Approach

For the first part of the problem, we performed an extensive research on the available classification libraries and tools available in the literature. Given the assumptions and project goals, we think that `pycaret` would be the best choice (please see here). `pycaret` is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that exponentially speeds up the experiment cycle and makes you more productive. Compared with the other open-source machine learning libraries, `pycaret` is an alternate low-code library that can be used to replace hundreds of lines of code with a few lines only. This makes experiments exponentially fast and efficient. `pycaret` is essentially a Python wrapper around several machine learning libraries and frameworks, such as `scikit-learn`, `XGBoost`, `LightGBM`, `CatBoost`, `spaCy`, `Optuna`, `Hyperopt`, `Ray`, and a few more. The design and simplicity of PyCaret are inspired by the emerging role of citizen data scientists, a term first used by Gartner. Citizen Data Scientists are power users who can perform both simple and moderately sophisticated analytical tasks that would previously have required more technical expertise.

We started the implementation of the binary classification code in Jupyter Notebook. The detailed code is available here. We first load the dataset into `pandas` dataframe and verify the data types in different columns as well as whole dataset table. Then, we define a `pycaret` experiment with the given dataset and the target classes. The simple command `compare_models` in pycaret trains different classification models and shows their performance in a resulting table. The command resturns the best performing model based on the evaluation metrics such as precision, recall, F-1 score, etc. The best trained model can then be used for the inference using a simple command `predict_model`. To save the trained model, we use the `save_model` command which saves the resulting model as a `pickle` file which can be then deserialized using `load_model` command to be used for inference.

Our plan for the API includes creating a detailed and interactive website via a Flask-RESTful API for the model comparisons we have developed for our dataset, and allowing users to both observe and compare the performances for the Pima Indians dataset. The idea behind using a Flask-RESTful API is the simplicity and versatility of Flask combined with a very consistent and flexible method of a REST type API. The website will be self made, simplistic, but having all functionality of the program fully implemented. We plan to continue referring to work from the class github, class lectures, and other available academic resources online to complete the API and fully integrate it with our program.

4 Intermediate/Preliminary Results

So far, we noticed that by running the code repatitively, the best model differs between the Random Forrest Classifier, Ridge Classifier, and Logistic Regression. The evaluation criteria for these models are very close to each other and due to the random nature of the model training, every time, we get a better result from various models. The prelimnary results can be found in the Jupyter Notebook in here. In summary, the best result pertains to the Random Forrest Classifier with the accuracy, precision, recall, and F-1 score of 76.89%, 69.11%, 59.74%, and 63.81%, respectively. For the next week, we are planning to dive deep into the issue and improve the performance of the trained model by changing the hyperparameters of the model.

We currently have begun early work on the website, downloading required tools for the API such as Flask, Flask-RESTful, and Postman. Our index website is currently a very simple skeleton of what sites the API will generate, that will eventually fill in with information about model performance, with preliminary work done on developing the API itself as well. Currently that is also a skeleton, but with functionality of receiving and processing HTTP GET requests, and creating and filling a dictionary, returning it as a response to a GET request. This kind of functionality will prove crucial for the eventual implementation of ML model prediction, analysis and comparison. Initial performance has shown that the GET requests are successfully handled, and return a generated dictionary in the form "x":x, where x is a number from one to ten. The complexity of the API is only going to scale from here, but this kind of work is a good step, considering our inexperience with APIs at this scale.