AWS Machine Learning Engineer Nanodegree - Capstone Project

I. Proposal

I.I. Project Overview

RMS Titanic was designed to be the more luxurious and safest ship built in 20th century. On the night of April 20, the Titanic hit an iceberg and sink in the middle on its journey. Unfortunately, due to the low number of rescue boats, more than a half of the passengers have died. The survive number was only 722 of 2224 in total.

The project proposal is to build a predictor model that recieves, as input, passenger information (like name, age, gender, socio-economic and class) and return if this fictitious passenger would survive or not in Titanic tragedy. To go further and receive theses passenger information, an endpoint will be develop using Python Frameworks in order to demonstrate another away to create endpoints, instead of those shown during the Nanodegree Program using AWS.

I.II. Problem Statement

The problem is Kaggle challenge and can be access here (https://www.kaggle.com/c/titanic). Based on the passenger data, the challenge is to build a predictive model that answers the question:

> “what sorts of people were more likely to survive?”

In others words. The idea is to use the provided dataset, which contains all informations about the passenger aboard Titanic in 1912, and build a machine learning model that predicts if the passenger would survive, based on new data received.

I.III. Datasets and Inputs

As it was mentioned above, the dataset was provided by Kaggle on a machine learning competition. The original dataset can be downloaded here (https://www.kaggle.com/c/titanic/data) and comes with 3 files:

- **train.csv** - file which contains data that have to use as model training input.
- **test.csv** - file which contains data to check the accuracy of the model created.
- **gender_submission.csv** - file which contains examples of what a submission file should look like. This file will not be used.

All features in the dataset in based on passenger informations, and they are described below.

- **PassengerId** - a unique value to identify the passenger (integer).
- **Survived** - indicates if the passenger survived or not (integer).
- **Name** - passenger name (string).
- **Pclass** - passenger ticket class (interger).
- **Sex** - passenger sex (string).
- **Age** - passenger ager (integer).
- **SibSp** - number of passenger siblings and spouses aboard (integer).
- **Parch** - number of passenger parents / children aboard (integer).
- **Ticket** - passenger ticket number (string).
- **Fare** - paid fare (float).
- **Cabin** - passenger cabin number (string).
- **Embarked** - port of Embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

I.IV. Solution Statement

The solution is a classification model that predicts if the passenger would survive based on the passenger information. To make predictions through HTTP requests, an endpoint will be developed using Python Frameworks. Before modeling the predictor, python libraries like Numpy, Pandas and Matplotlib will be used to data visualization and data preprocessing. Then, the model will be built using Sklearn Estimators and Sklearn Metrics Functions to measure model quality. Lastly, Python Flask and Docker will be used to create an endpoint to access the model and make predictions.

I.V. Benchmark Model

First, it will be user Sklearn Estimator without any special parameters (default hyperparameters). Then, to compare the results, a grid search model selection will be used to select the same model with betters hyperparameters and compare with the default model.

I.VI. Evaluation Metrics

For the model evaluation, a accuracy score and confusion matrix will be used.

I.VII. Project Design

Data visualization

- this is the step to get more information about the dataset, visualizing them.

Data preprocessing

- fill empty rows with median (or other metrics).
- remove remaining empty rows.
- encode non-numerical features, if necessary.
- remove remaining non-numerical features.
- normalize numerical features with MinMaxNormalization.
- balance data per target class.

Data splitting

- split the train.csv dataset into 85% for training and 15% for testing.
- **OBS:** the test.csv file will be used to evaluation.

Model selection

- training different kind of Estimator with default hyperparameters.
- choose the best one

Hyperparameter tuning

- using grid search to select the best hyperparameter.

Endpoint

- create an REST API to make predictions.