دانشگاه کردستان
University of Kurdistan
زانکۆی کوردستان

Apprenticeship project report

Project title: Federated learning for traffic prediction

Professor name: Dr. Sadon Azizi

Student name: Arash Ahmadi

Summer 2021

Introduction

In this project, the traffic prediction system has been implemented with Federated learning machine learning.

Federated learning is a machine learning technique that instead of a central unit such as a server performing the learning operation, this operation is divided between different devices in order to divide the heavy load of this type of calculation between different devices and also the delay of sending and receiving data from the sensor to the computing center is reduced.

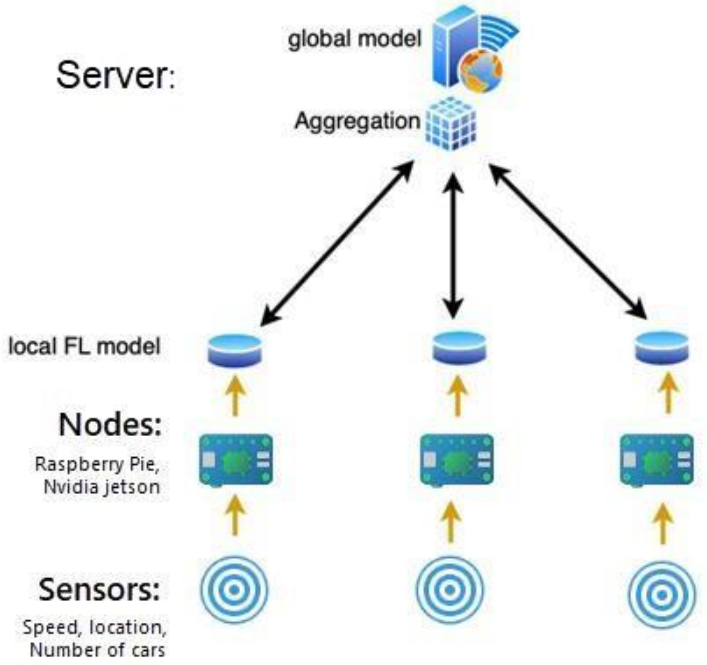

Figure 1: In this figure, Nodes take information from sensors, perform learning on the received data so that they can predict traffic for the future. Then they get the final data (Model) locally and send it to the server. The server combines these models during the Aggregation operation so that different users can receive traffic forecasts from the server.

Another advantage of this type of learning is greater security of input data for learning. For example, in a traffic forecasting project, a sensor should be placed in a place that is calculated by the client in that area, the number of cars that are on that street at a certain hour. The process of obtaining this data includes obtaining the geographic location of the cars, the ID of each car, their speed, etc. But the server does not need all this information and in this project it will only need information such as the number of cars in a one-hour period and the street ID. In this way, comprehensive and sensitive information will not be sent to the server, and the bandwidth consumed by the server will be reduced, and the server will be able to respond to a larger number of users.

In the next parts, we will talk about the execution environment, codes and algorithms of the project.

Project execution environment

This project is written in Python and can be run in any operating system that has Python 3 or higher.

The following libraries need to be installed to run Python code on the server:

The final model received from the client is received as a data frame from this library (Numpy and Pandas)

This library is used to display data graphically (Matplotlib and Seaborn).

This project can be run on the client both directly or on Nodes through Docker. Docker is a software platform for building applications based on containers that are small and light operating environments that share the operating system kernel and run separately on Linux, Windows, and Mac operating systems. Docker helps developers to develop their software. Build easier than ever and basically "build once and run anywhere".

Figure 2: Libraries of the client version of this project

Through Docker, you can store and install all the files and apps needed to run the project, such as the Python library, the Alpine Linux distribution image, and all the libraries mentioned above in a container and to all devices that have Docker installed without doing extra steps and installing prerequisites.

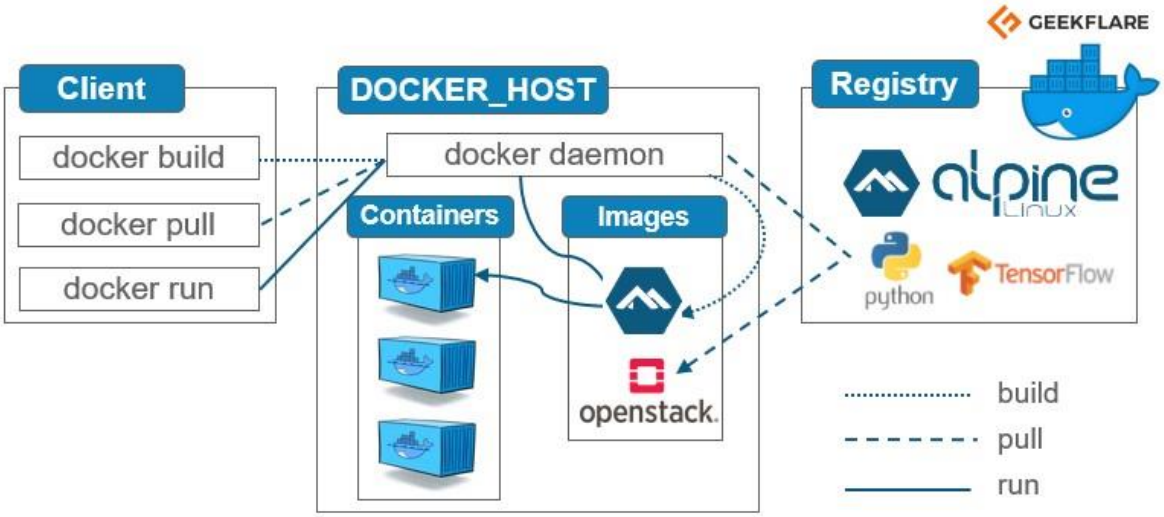

Figure 3: This figure shows how the project is executed through Docker

Docker does not run directly on Windows and can be run via wsl2 or Hyper-v. To use gpu in Docker, Windows 11 with build number 22000 and above and Windows 10 with build number 21364 and above and cuda software are required.

Description of code and algorithms used in the project

The client first tries to connect to the server socket. After connecting to the server, the server displays a message that it is ready to receive the trained model from the target client.

Then the data related to the traffic of each region is read from the csv file through the pandas library and placed inside a data frame. Dataframes are data structures that can store data in two dimensions and label each row and column.

Before the Normalize function, we separate the number of desired Junction machines (intersection where traffic is possible) from the data frame and place it in the column section and remove the ID part from the data frame.

Now, in the next step, we have to prepare this raw data for learning. The data we have is called Time series. Sequences of data collected in a time limit form a time series. These data reflect the changes that the phenomenon has undergone over time. Therefore, we can consider these values as a time-dependent vector. In the dataset of this project, the number of cars at different hours is considered a time series according to this definition.

When it is possible to forecast for a time series that has become stationary. A stationary time series means sequences of time-dependent values whose mean and variance do not depend on time.

In fact, in a static time series, the laws governing the changes of values are not dependent on time.

Trends (the tendency of a time series to increase, decrease, or even be constant) can have different averages over time, while seasonal variations (changes that occur repeatedly over periods shorter than one cycle) can lead to change of variance over time, both of which

define a time series as non-stationary. Stationary data sets are sets that have a stable mean and variance, and in turn, they are much easier to model.

Therefore, in this project, an attempt has been made to make the dataset static before learning by stabilizing the mean and variance and differentiating using Normalize and Difference functions. After this, we use the Stationary_check function to make sure that the initial data is stationary.

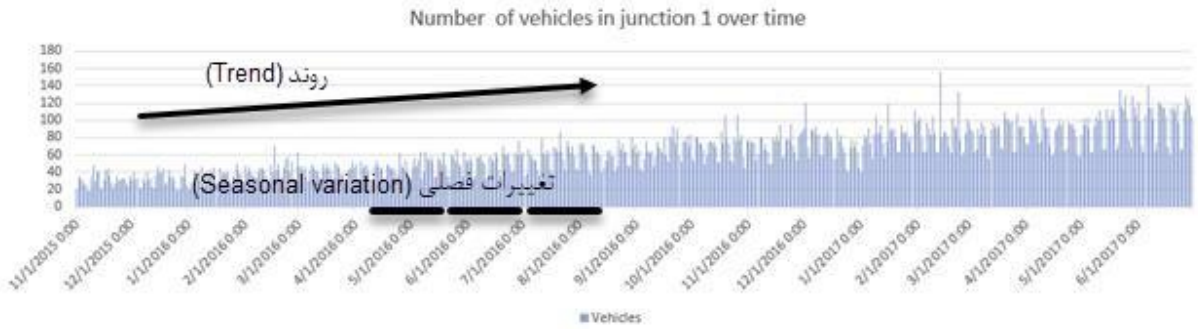

Figure 4: The number of cars related to the first Junction is shown in the period of 1 year and 8 months. As we can see, the graph has an increasing trend over time and we see seasonal changes in each time period.

Now, in the next step, we split 90% of the initial dataset, and in the next step, we perform training on them.

We use Recurrent Neural Network (RNN) to train the dataset. First, let's talk about the neural network itself and say why we use this type of learning in this project.

A neural network is like a brain, except that we want to implement this concept in a computer. which has input (eyes), output (hands) and senses whether the outcome is good or bad (eating cake/staying hungry.) Like a brain, it tries to keep doing good things and what it feels it is bad not to do.

However, unlike humans, it cannot see things, move, or know what is good and what is bad. A programmer must give it eyes, hands, and skin in order for it to do something.

Let's take an example: a programmer wants to force a neural network to play Mario. When a programmer gives the neural network eyes (to see the game) and hands (to play the game), the programmer tells the network to try to go as far to the right as possible. If it stops, it feels hungry.

If its level increases and wins the game, it will feel like it has eaten a piece of cake. So, the neural network will try to keep up its level and not fail.

But how can this concept be implemented? The neural network consists of many small "roads" and "intersections" starting from the eyes and ending with the hands. Every intersection has a light that says:

"Go left" or "forward" or "right".

Every time the eye sees something, a car is sent down the road. Cars then turn to each intersection as indicated by the lights. Then, at the end of the road, it reaches the hands and the hand moves depending on the end of the road it reaches (for example, the left hand if the end is left, the right hand is the right end)

If the hand moves and the neural network feels "good", it sends more cars through using these intersections. If he feels bad, it avoids sending the car in this particular way and goes to other four ways.

After playing for a while, the crossroads lights are all in the best direction for the neural network to defeat Mario.

RNNs are just neural networks, except that cars can also reverse if a light says so. That way, the machine can spend more time or take a different route to get to the "feel good" hand and not the other hand.

GRU gating mechanism is used for this project. GRU is like long short-term memory (LSTM) with “forget gate”, but it has less parameters than LSTM, because it has no output gate. TensorFlow library has been used to perform this type of learning.

The way this type of learning works has more details, which is beyond the scope of this report because the focus of this report is on the implementation of the concept of federated learning. To prepare the dataset to perform this type of learning, we will use the TnF function to obtain Feature and Target and give the output data from these functions to the GRU_model method to start the learning process.

After performing the learning process, finally, the GRU_model method returns the trained dataframe to us. However, we made this trained dataframe stationary in the previous steps in order to have a more accurate prediction.

So, in order to show the data in its real scale, we reverse the Normalization and Difference done (the reverse of the calculations done to make it stationary)

Now, we store the data frame that contains the prediction result and real data in the Result list, and then we convert the result into a string with the pickle library and the dumps method and send this string to the server through the socket. (We send the real data to compare it with the predicted data)

Now on the server, we decode the data received from the client through the pickle library with the loads method and put it in

We save a list called trained_data so that later every client sends a traffic prediction request to the server, we extract the result by referring to this list and send it to the desired user.

References

https://docs.docker.com/get-started/

https://www.positronx.io/create-socket-server-with-multiple-clients-in-python/

https://towardsdatascience.com/why-does-stationarity-matter-in-time-series-analysis-e2fb7be74454

https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

https://blog.faradars.org/time-series/

https://www.kaggle.com/fedesoriano/traffic-prediction-dataset

https://www.kaggle.com/karnikakapoor/traffic-prediction-gru