GEORGE BROWN COLLEGE
SCHOOL OF COMPUTER TECHNOLOGY
APPLIED A.I. SOLUTIONS

FULL STACK DATA SCIENCE SYSTEMS

Dr. Vejey Gandyer

Mohammad Ardalanasl

April 17, 2024

Project Overview

The objective of this project is to develop an LSTM-based stock value predictor and deploy it to the cloud. This deployment will enable users to access the predictor for free, allowing them to integrate it into their trading strategies.

Data Acquisition and Preparation

To acquire the necessary data, I utilized an API to retrieve historical stock values for a selection of symbols. Our training dataset comprised 100 symbols, and I collected data spanning two years. To prepare the data for training, I scaled the values between 0 and 1 and then stacked them together. This process ensured that the data was suitable for input into the LSTM model, facilitating effective training.

Feature Selection and Model Architecture

While there is a multitude of parameters that could be considered as features for training the model, I opted to focus solely on the stock value history due to constraints in time and budget. This decision allowed us to streamline the project and focus on creating a model that could be deployed to the cloud. Our primary objective was to gain hands-on experience in the model development and deployment process, laying the groundwork for a platform that could benefit end users in the future.

Training Model

The training process of this project consists of two main components: the training model and the predicting model. In the training model phase, I utilized the entire dataset comprising the stock values of 100 symbols stacked together. To define the architecture of the model, I needed to specify our hyperparameters. For hyperparameter tuning, I employed a Particle Swarm Optimization (PSO) method, which is an optimization technique. PSO facilitated the automated search for optimal hyperparameters. This metaheuristic method iteratively refines the training process, utilizing approximately 10% of our data. The hyperparameters tuned include those that define the model architecture, such as the number of look-back steps in the time series format, the number of units in the LSTM model, the activation type, and the number of layers in the model.

Model Evaluation

Following the determination of optimal hyperparameters, I constructed a model based on the recommended architecture and trained it using 95% of our dataset. The remaining 5% of the data was reserved for testing the trained model. To assess the model's performance, I compared the mean squared error (MSE) of the training and test sets. Additionally, to evaluate the model's ability to generalize to new data, I tested it using the AAPL symbol. I visualized the real and predicted values on a graph, demonstrating the model's ability to closely predict actual stock values.

Deployment Preparation

In the deployment phase, I aimed to prepare the trained model for deployment on the cloud, enabling users to test it in real-time. To achieve this, I converted the notebook model into a script, organizing the separate parts of the model into functions callable within the main function. This process allowed us to carefully review our code and structure it more efficiently. Within this script, I imported the trained model to predict stock values for randomly selected symbols and plotted the results on a graph.

API Development

In the subsequent phase, I developed an API using Flask based on our previously created script. This involved designing several HTML files, including an index page to welcome users and a prediction page where users could select different symbols for predicting their future values. I decided to offer two types of prediction: one for daily predictions and another for weekly predictions. Flask was instrumental in integrating our HTML pages with the predictor functions. To ensure real-time functionality, the model required the latest stock value data. To address this, I implemented a function to retrieve the latest data of predefined symbols from a free API and store it for use throughout the day. Although this process could be automated to occur daily, it was not within the scope of this project.

Model Deployment

Upon completing the Flask model, I conducted a preview test of the model on the local host and enhanced the visual appeal of the HTML files. I integrated images generated by AI into the background of the pages and ensured that the style of text, buttons, and containers was appropriately configured for a polished user interface.

Docker Containerization

To make our models accessible across various environments, including cloud servers, I containerized them using Docker. I began by installing the Docker extension and Docker app. Next, I created Dockerfiles containing the necessary commands to build the Docker image. Additionally, I created "dockerignore" files to specify which files should not be included in the Docker image. This approach ensured that the Docker image was created with an optimal size.

Environment Configuration

During the process of creating the Docker image, I encountered several errors. The primary issue was related to the TensorFlow library's inability to find its dependencies for installation in the Docker image. After extensive research, I discovered that when using TensorFlow in a Docker environment, a specific version of TensorFlow that includes the necessary dependencies must be used. To address this, I replaced the base Python model with the required TensorFlow version in the Dockerfile.

Subsequently, I encountered an error related to the Flask library. Although Flask worked well in the local environment, it was not compatible with the Docker environment. After testing different Flask versions, I identified one that was compatible with our Docker setup.

Another challenge I faced was configuring the Docker firewall to allow data transmission to the browser. Even though the Docker image was created successfully, the model did not send any data to the user interface locally. To resolve this, I configured the model to open all IPs for access and deactivated the firewalls of the operating systems. These adjustments enabled us to successfully test the model in the local environment.

Cloud Deployment

In the subsequent stage, I uploaded our Docker image to the Docker Hub platform to enable access in online platforms such as the Google Cloud Platform (GCP). Following this, I created an account on GCP and activated the Compute Engine service to establish a Virtual Machine (VM) instance.

After creating the VM instance, I configured the firewall rules to allow all IPs access to the API. I then proceeded to install Docker on the VM and pulled the Docker image from Docker Hub to the GCP environment. Upon completing these steps, I gained access to the online interface using the IP address and port of our VM instance.