Prodigal Final Documentation

Balaji Baskaran, Gabrielle Chen, Sean Lin, Jamie Paterson, Wonwoo Seo, Htut Khine Win

Table Of Contents

Description

Trading stocks has been a lucrative way to make money in the past century. Trading experts such as Warren Buffet have gained profits of tens of billions of dollars while beginners in the field have a hard time understanding the rules of the trading game. In recent years, stock trading has become more quantitative with algorithms and computer programs dominating trading activity, and many such strategies have had a better track record than that of successful investors. In particular, machine learning technology has been proved in practice to have a comparable learning ability to successful investors such as Warren Buffet. Thus, our team has developed a product to assist novices in stock trading to help them make informed trading decisions and narrow the gap between expert and novice stock traders. The goal of our platform is to provide stock prediction values derived from linear regression models with the goal of providing users with useful insights that can generate profits.

Prodigal is an open-source web application we created in order to facilitate the aforementioned goals. With machine learning models and market price data fetched from the Alpha Vantage API, Prodigal is able to provide prediction values based on a linear regression analysis of historical price data. Prodigal has three major components; front end, back end, and machine learning. Front-end work involved setting up dynamic web pages, data rendering, data visualization, company search, and user interface design. Back-end work were the creation of a MySQL database and a REST API offering both historical price values and predicted price values. Then, in conjunction with services offered by Microsoft Azure, the machine learning group extracted useful data from the databases, implemented our own linear regression model, ran the experiments, and fed our back-end API with predicted prices.

Process

XP breaks down the timeline of the project into iterations, and developers work on selected use cases in each iteration. In this project, we had a total of six iterations, with each iteration being two weeks long. At the start of each iteration, we first planned on which use cases to implement in that iteration and assigned selected use cases to subgroups. This gave us flexibility in balancing our workload with our teammates' schedules; more use cases were selected and assigned in iterations in which members had more time to spend on working, while less work was assigned during exam weeks.

XP promotes meetings throughout the development phase, so our group frequently had meetings throughout the project. These meetings were helpful in keeping every member on the same page and ensuring development progress. To report current

progress and receive feedback, team members and our assigned TA had an iteration meeting every two weeks. To check the progress of each subgroup, review code, and prepare for the iteration meetings, our group also held regular weekly meetings. To work on use cases in pairs, each subgroup had its own work meeting outside of the regular team meetings. Each subgroup was required to spend a certain amount of time working every week to ensure progress and archive the agenda of every meeting on the team Wiki to keep track of the group's progress. By making the progress of each subgroup transparent, members could check if their group was falling behind or not and provide or ask for help to ensure every assigned work was done before the iteration ended.

From the development aspect, XP requires developers to practice pair programming. This was the main purpose of subgroup coding meetings; we required members of each subgroup to be present at their coding meeting and write code in pairs. The benefit of pair programming we experienced is quality of code. Since partners provided ideas from different viewpoints and corrected code smells or snippets that did not follow coding standards in real time, we wrote better code compared to solo work. This gave us confidence to write code and prevented us from repeatedly worrying about whether our code works and looks good. Pair programming allowed our team to create quality code in a shorter time period, ensuring efficiency of the development process in the project.

Another important development concept in XP is refactoring and collective code ownership. Team members practiced refactoring and achieved collective code ownership through the code review process. After each subgroup finished a use case, said subgroup would create a merge request to master branch on project repository, notifying the rest of the team that a use case has been completed. This allowed the whole team to examine the new code and suggest possible refactoring or exchange opinions on part of the code. Since members other than the writers had unbiased viewpoints on the code, they easily identified bugs and code smells that the writers could not see. The code review process increased the overall quality and stability of the project and gave members feelings of responsibility and ownership toward written code.

For testing, we required both the front end and the back end to achieve coverage higher than 80 percent for the unit tests. For the front end, we also used Selenium to simulate user inputs and test our user interfaces. In addition to automated tests, we also dedicated a significant amount of time in the last iteration on manually testing our project on varying environment to find bugs which our automated tests could not detect.

Requirements & Specifications

Our platform requires the following features to satisfy all of the goals outlined in our actor-goal list: account creation/login/logout/deletion, view profile, email verification and notification for accounts, company search, view search results, view media/news, sector search, company comparison, favorites, and search history data for each user. We implemented all of these user goals throughout our six iterations.

For the front-end, we used regular JavaScript. For the back-end, we used Django, MySQL, the Alpha Vantage API, and sklearn. The app is hosted and deployed as an Azure web app.

Iteration 1	Estimated	Actual
Write a proposal	5	10
Form the team	5	5
Decide roles and setup development environment	5	7

Iteration 2:

The web app team installed Django on Azure and deployed a simple working Azure web app, while the machine learning team deployed a simple machine learning model in the Azure ML studio.

Iteration 2	Estimated	Actual
Setup Django framework on Azure	2	3
Build basic HTML hierarchy	3	5
Create index.html (user can navigate to the home page)	10	10

Iteration 3:

The user can now search for companies and get redirected to the specific web page about the company searched. We also established a REST API endpoint that can return market data, which is used for the chart generation in addition to initializing a training set for our machine learning models.

Iteration 3	Estimated	Actual
Setup CI/CD pipeline for machine learning app	5	12
Integrate CI/CD into web app	5	14
Create logo image	5	2
Decide color theme	3	3
Build search bar (user can search for companies)	10	4
Build search results page (user can get search results from their query)	10	12
Create REST API for historical data	15	18
Create MySQL DB for machine learning data	15	15
Add new machine learning models	10	8

Iteration 4:

The user now has a profile page with unique information about their account displayed on the browser, such as username, history, and favorites. Each company page contains a favorites button which can be toggled on/off. Favorites are stored in a database table and are loaded on the profile page. A proprietary authentication system exists that no longer requires users to use their Google accounts.

Iteration 4	Estimated	Actual
Build profile page (user can create a profile)	10	8

Create MySQL DB for web app data	5	5
Build proprietary login/signup functionality (user can sign in/login without any third-party OAuth)	10	6
Search history (user can view their search history)	15	15
Favorites (user can add/remove favorites to their profile)	10	13
Password hashing	10	6
Create REST API for stock price data	10	5
Integrate machine learning app with web app	10	8
Create prediction models using sklearn	15	10
Create REST API for prediction data	15	12

Iteration 5:

The user is able to find companies they were previously unfamiliar with through the autocomplete feature. This feature works by filtering the list of companies displayed on the front-end based on the user input. The large quantity of companies displayed by the autocomplete is facilitated by a scrollbar, where only 6 companies are displayed at a time, preventing the page from being too populated with entries. The user is able to search for an entire sector rather than individual companies, which displays a list of companies within the searched sector to the front-end. The user is also able to compare two companies by three different metrics: absolute value, daily percentage change, and tendency. All of the pricing data is displayed on charts provided by the Google Charts API. Chart lines are also color coded to provide unique identification, with the prediction line as red and the historical price line as blue.

Iteration 5	Estimated	Actual
Autocomplete on search	10	15
Search by company name	8	8
Search by sector	10	7
Compare companies	10	12
Graph prediction data	10	15
Migrate databases and API server to Azure	15	20

Iteration 6:

The user is now able to receive email alerts pertaining to activity on their account and receives a welcome verification email upon signing up. These features are done using Django send_mail(). Signing in can also be done with email rather than username. The user is also able to automatically sign in when loading the page on their browser by clicking the “Remember me” box. Finally, we migrated the HTML templates to the bootstrap framework, which provides a more visually appealing interface in addition to facilitating responsive design when the pages are re-sized.

Iteration 6	Estimated	Actual
Polish UI	12	10
Email verification	10	10
Email notification	10	5
Migrate front end to Bootstrap	25	23
Remember me	5	4
Secure API with API key	5	5

Architecture & Design

Prodigal has two main architectural components - the web front end for rendering the web pages and interacting with the users, and the machine learning back-end framework to provide the front end with necessary computations and data. Since the purpose of our application is to provide users with ease of access to stock prices in real-time and help them make better investment choices by supplying them with a machine learning prediction model, we, as a team, decided to use the web as a medium to implement our design.

UML Diagram for back end

When we started to decide on the languages and frameworks, we emphasized three core attributes: support for rapid prototyping that aligns with the “fail-fast” system design, a stable framework that has been around for some time so that we don’t have to reinvent the wheel, and team members’ familiarity with the language framework. With those attributes in mind, we came up with the clear winner for our project - Python and the Django framework. One of the reasons Python is well-suited for our type of project is its relatively easy syntax for rapid prototyping. Moreover, its dynamic typing mechanism

suffices for its slower processing time compared to its statically-typed siblings such as C++ or Java. To complement Python, Django as a framework, is a popular and stable framework that works well with Python.

UML Diagram for front end (web application)

Choosing Django as our main framework to supply data and computations for both the front end and the machine learning back end had significant benefits throughout our whole project. Since our project relies heavily on data and computational work in the machine learning model, data storage, access, and preservation has been our primary concern. Django meets our particular requirement through its ORM (Object Relational Mapping) system for databases. ORM is a mapping from an object in programming languages to database tables/rows/columns. With ORM, our project can move from relational databases such as MySQL to non-relational databases such as Postgres or MongoDB in one key-value switch in the configuration file without needing to edit all of our code to reflect the database change.

Along with ORM, our project has benefited greatly from decoupling our project into two applications - a website front end for user interactions and a machine learning back end for computations and data storage. We could, in theory, combine those two into one application. But, one major advantage to our approach is that if one day, we want to provide a hostless (without a website) service to help users with stock price prediction, we can just host our machine learning module anywhere on the web and users can still have access to our core implementations.

Another important add-in for our system is the Django Rest Framework (DRF) that provides REST API endpoints for our system. Although ORM is very useful in its own aspect, REST API endpoints provide another layer of abstraction for the communication between our front end and machine learning back end. Now, whenever our front-end developers need access to data such as the daily closing prices of tickers, they don't need to query the database directly. They can, instead, send an HTTP GET request to one of our endpoints and get information quickly. Moreover, having a REST API ensures the security and integrity of our data. Everyone who wants to update or get our data has to be granted a security token, and as developers of Prodigal, we can monitor the accounts that have recently accessed our data.

After describing the core software packages and the reasons behind each library, it is time to provide descriptions of the entry point into our system and how our machine learning model and front end interact. It is important to remember that the core functionality of our software is predicting stock prices given the history data. To provide users with such functionality, the Prodigal website has a search bar. When users type in the search "keyword", the website displays a graph showing the historical data and at the tail of the historical data, we provide future data in a different color to show the prediction for this particular ticker.

It seems easy enough when users see the prediction from this search button click. Yet, behind the scenes, the setup is a fairly complicated system design. When users search for a particular ticker, front end website sends a HTTP request to the machine learning (ML) backend. Then, the ML backend queries the historical data of particular ticker to the ML database. The ML database first searches the ticker in its on-premise database. If the data points are found, it runs the machine learning model and sends the prediction and history data back to front-end. If the ticker data point is not found inside the on-premise database, ML backend pulls the data points from Alpha Vantage (AV) API. If the AV API returns history data, the ML backend uses those values to run an on-demand machine learning model and returns the historical data and prediction values to the web front end to display the data. An important thing to note

from this design is that our design frees us from the dependency on any kind of prepopulated database of tickers. If we can't find the ticker, we'll populate it when user searches for it.

Reflections and Lessons Learned

Balaji Pandurangan Baskaran (baskarn2):
This project was interesting, and I learned many useful traits by working on this project. These include using and developing on the Django framework and using Azure Web Services. I worked primarily on the front end and the necessary back-end components related to the front end. In the aforementioned parts, I was able to gain experience using the Django framework, creating and using the MVC model. I was also able to work on setting up the CI/CD pipeline for the project. I really like the details and intricacies involved in creating a CI/CD pipeline and the different cloud services that went into supporting it. Finally, I was greatly involved in constructing, updating, and maintaining the test suites for the project. This includes using both Selenium and PyUnit. As the initial project envisioner, I additionally learned a lot about the non-technical aspects of working with teams, including time management and clear communication.

Wonwoo Seo (wseo2):
This project was the most challenging out of all the team projects I have participated in, and I learned a lot from this experience. From doing front-end work, I learned how to use the Django framework and was able to get a better understanding of MVC. I also learned Bootstrap, which will be helpful in the front-end work I will partake in in the future. From doing back-end work, I believe implementing the REST API was the most helpful experience. From the development aspect, I caught a glimpse of what being a full-stack developer will be like in the real world.

As a team leader, I also learned lessons regarding project coordination; the leader should have a clear vision of the team's goal, know the members' skills and capabilities, and be responsible to check the details. It was my first time being a leader of a team project, and these lessons I learned from my mistakes will help me if I have the opportunity to manage projects in future.

Sean Lin (xunlin2):
This project made me apply technology I learned in practice, integrate them together, and improve my skills. For example, I have worked with the Django framework in other projects but was not so familiar with the logic behind its structure. However, by working

on Prodigal, I learned even more, such as static, template, request, and models. In addition, I now have experience with web hosting, launching, and connecting to the REST API. I also learned to care about both the front end and the back end and being a full-stack developer. The other important thing that impressed me is the development process and teamwork. I didn't take CS 427 and thus, was not familiar with XP. This project and my teammates help me code correctly, work efficiently, and finally give me exposure to industry-level production development and coding standards.

Jamie Paterson (jlp3):
I was unfamiliar with Python before this semester, so this project provided me with an enjoyable and worthwhile learning experience in that programming language. Besides learning general Python syntax and coding standards, I was also able to learn how to use specific tools such as the sklearn library and the Django framework, with the latter being surprisingly easy to use and understand; I particularly liked the MVT architecture of Django due to its simplicity as a web framework. This project has given me useful full-stack web development skills that I will definitely apply in future Python-related projects. I also improved my JavaScript skills through the autocomplete feature that I developed and became familiar with the Azure platform which I had hitherto used. This project also gave me further exposure to the Agile development methodology which is widely used in industry. Overall, this project helped me learn a new programming language, and how to work effectively on collaborative software projects.

Htut Khine Win (hwin16):
This project is really significant in the development of my understanding on how websites work. Before this project, I was mainly a software engineer who worked on system design used for building a telescope. I don't have any prior knowledge working with websites. Yet, in this project, I learnt how to setup a website, join databases to it and create API endpoints. I learnt that API endpoints are really powerful constructs that build up on top of databases. By learning to use API endpoints, I didn't have to interact directly with databases or learn the syntax of particular database to build particular database query. Another aspect worth mentioning is I learnt about Docker containers, how to setup dockers, how to bring up dockers, and most importantly how to deploy those docker containers. In the first few meetings, I was the team leader to bring the machine learning back end together. Being the team leader, I faced many challenges, one being how to design a system and the other being how to give directions to each team member so that he gets a hands-on learning experience and enjoy being part of this learning process. In this aspect, I somewhat failed and struggled. But at the end, each team member came together and contributed much more than I would have

expected. It was a challenging project to work on for a semester and I thoroughly enjoyed working with these guys.

Gabrielle Chen (gchen46):

In the first five iterations, I worked on creating and improving the machine learning model our application uses to predict the closing price of a given company. Initially, the model was built using Azure ML studio. This was a mistake as we could only feed our model static CSV data. In addition, we would need to create a model for each ticker symbol because each static CSV file fed into the model only contained information for one ticker symbol. What we wanted, however, was an efficient model that takes in real-time data and outputs a list of five prediction values for any ticker symbol. I then decided to move our model out of Azure ML studio, opting to use the standard Python libraries instead. To ensure the model uses real-time data, I created a JSON to CSV converter with which I could feed the resulting CSV file into the model. Interacting with JSON was new to me; I had only used CSV files for data-related projects before this. In the sixth iteration, I implemented email validation and confirmation (the welcome email). From this, I learned about type checking and the email functionalities Django has to offer.

More importantly, however, I learned what a good team is. Prior to this project, I was always the leader of all of the group projects I have worked on. Those projects were stressful as communication was lacking and some team members refused to contribute. The Prodigal team is a different story. Everyone contributed, and I must say that I will miss working with everyone once this project is completed.

Appendix

Code Documentation

1. **Get Doxygen**
    a. Get Doxygen binaries from http://www.stack.nl/~dimitri/doxygen/download.html
    b. For Linux/macOS, you can also get Doxygen with following command: wget http://ftp.stack.nl/pub/users/dimitri/doxygen-1.8.2.linux.bin.tar.gz
2. **Place Doxygen binaries at root of the project**
3. **Generate Doxygen**
    - Frontend: doxygen Doxygen-Prodigal-frontend
    - Backend: doxygen Doxygen-Prodigal-ml

Docker Image Deployment Instruction

1. **Run docker containers**
2. **To deploy backend image:**
    - docker pull hwin16/prodigal-ml:v1.0.6
    - docker run -p 8000:8000 hwin16/prodigal-ml:v1.0.6
3. **To deploy frontend image:**
    - docker pull hwin16/pro-frontend:v1.0.3
    - docker run -p 8000:8000 hwin16/pro-frontend:v1.0.3