Paul Yaginuma
MSDS434 Winter 2025

Golf Prediction Application - Overview

Introduction

This is a cloud-based prediction application that takes in a golfer's stats for a round of golf and predicts their strokes gained tee-to-green (SG T2G) for that round. The strokes gained statistic captures how much of an advantage a player gained compared to the field in that particular round of play. Strokes gained tee-to-green specifically captures how well a player hit their tee shots and approach shots relative to the competition.

This is a valued statistic for golf analysts and is often referred to as ball striking since tee shots and approach shots are hit harder and farther than chips and putts. The conventional wisdom is that ball striking is more consistent tournament-to-tournament than other aspects of the game like chipping or putting, which are more affected by things like weather and grass type. However, tournaments often only record a statistic for strokes gained total and do not capture or publish results for each individual strokes gained category. Being able to accurately predict a players SG-T2G based on their other stats is useful for feature engineering for a follow-on performance prediction model.

Application Design

This application uses several AWS cloud based tools to automate the process of collecting data, training a machine learning model, and serving predictions to users, which could be other automated systems. For the collection of data, we use AWS event bridge to trigger a lambda function weekly on Tuesdays (after weekend golf tournaments have finished). This event update function pulls a list of new tournament metadata—date, name, tour, etc.—from datagolf.com and inserts the information into a database hosted on RDS. It then looks for any new pga tour tournaments that have strokes gained and traditional stats that were not already in the database. If it finds any, it calls a different API endpoint on data golf and downloads the actual player-round statistics for that tournament in CSV format. Finally, it keeps only the columns needed for the ML model and saves the file to an S3 bucket.

For ML model we use a stochastic linear regression model that can be fine tuned using new data. We initially trained our model on approximately 100,000 rounds of historical data. We used the following statistics to predict SG-T2G: SG-Total, driving distance, driving accuracy, greens in regulation, scrambling, proximity rough, proximity fairway, and poor shots. To keep our model up to date, we automated the process of fine tuning the model using new data that is updated each time a new PGA tour tournament is held that captures strokes gained data. To do so we use another lambda function that runs after the weekly data ingestion occurs. This lambda function in turn triggers a Sagemanker instance that pulls any data that has been updated in the past 5 days, fine tunes the model, then saves the updated model back to the S3 bucket.

Finally, we used docker to setup a containerized flask server hosted on an ec2 instance to serve predictions to users. Whenever the container is run, it pulls the most recent model from the S3 bucket and uses that for predictions. With the current setup, users can SSH into the ec2, input stats, and receive a prediction. Since this application will only be used to populate internal datasets—where the SG-T2G stat is missing—we don't plan to leave it publicly accessible in the future as we have for the demo.

Advantages / Disadvantages

The main advantages of this system are the scalability and cost savings, both due to using several serverless cloud tools. The lambda functions and sagemaker instances are all temporary, infrequently called, and only charge per use. This saves money compared to other approaches like having a hosted sagemaker prediction endpoint running constantly. We achieve this by using a cheap S3 bucket to store the data, the model, and the script that is used by sagemaker to perform the fine tuning. The only other costs are storage in the database, which is negligible due to the small amount of data, and the cost of having an ec2 instance running. However, we can shut down the ec2 instance as soon as we are finished populating the SG-T2G stat for the rest of the data. Finally, the use of modular serverless tools means that this application is very scalable—from our current use case which fine tunes on hundreds of datapoints—to anywhere on the order of millions of datapoints should we ever need to handle larger datasets.

The main disadvantages are the security issues with having an open e2 instance—which we addressed above—and the diminishing performance returns that we expect with continually fine tuning a predictive model. Continually fine tuning a model will eventually lead to poorer results if characteristics of the underlying data change, as they often do in sports. However, we can easily remedy this by occasionally retraining the entire model using all of the historical data.

Areas of Improvement

There are three main areas of improvement for this project. First, we will work to gather more data and host it all in normalized tables in a database. While we used csv files stored in an S3 bucket for this expedient solution, using a database is a more scalable and cleaner approach. A database is more scalable, less prone to errors and duplicates, and can be queried better than a collection of CSVs. Second, we will improve the security of the server by introducing access requirements and/or rate limiting to prevent unauthorized access, overloading the system, or attacks. Finally, we will implement more monitoring and testing for the various components of the application. While we have basic monitoring setup to count the number of predictions served and have demonstrated the use of Github actions for basic CI/CD functionality, we will add testing and deployment to the automated pipeline to further streamline application development and maintenance.

New Technologies

Two technologies that we could incorporate into this application are serverless API tools and LLMs. For a follow on performance prediction model—that predicts how well a roster of players is predicted to perform in a future tournament—we would incorporate tools like AWS API gateway to stand between the suer and our prediction server. This tool would allow us to implement features like access limitations and rate limiting using a simple scalable interface. If access to the future model requires user accounts or payment, this would be an effective way to implement those features.

Further, we could use generative AI and LLMs to offer a feature where users can submit a slate of golfers for a tournament and get a human readable summary in return. By using tools like PCA or SHAP values that assess feature importance, we could generate text that explains why the model predicts certain golfers will perform better than others on specific courses or in certain conditions. This would be useful for casual users who value text over raw prediction values.