Data Engineering
Emily Wang | July 2022

Creating a Tennis Betting Pipeline

Abstract

My goal is to build a match prediction tool for ATP (professional tennis) matches I'll be using the dataset of all ATP matches from 2000 to 2022 with roughly 57,000 matches and 23 variables. By creating a pipeline that periodically updates match predictions based on recent match data, the tool can empower many different use cases including determining future match draws, sports betting opportunities, and tournament tactics for players

Design

1. Data ingestion: The data is sourced from a tennis betting website Tennis-Data. I used Python to aggregate the CVS files into one combined dataset, then created a database using SQLite to store the aggregated data. My code pulls the aggregated data from the local database to create new visualizations and conduct the modeling exercises
2. Data Processing: Because my dataset has 57,000 data points, it wasn't necessary to use Google Cloud, so I ran the ML models on my machine
3. Deployment: I created a Streamlit Web app to present my findings

Data

The dataset contains roughly 57,000 matches. There are 35% upsets and 65% not upset matches. Some notable variables include:

- ATP = Tournament number (men bracket)
- Date = Date of match
- Series = Name of ATP tennis series (Grand Slam, Masters, International or International Gold)
- Court = Type of court (outdoors or indoors)
- Surface = Type of surface (clay, hard, carpet or grass)
- Round = Round of match
- Best of = Maximum number of sets playable in match
- Winner = Match winner
- Loser = Match loser
- elo_loser= The Elo Model ranking of the loser calculated *before* the match based on the playing history of the two players

- elo_winner= The Elo Model ranking of the winner calculated *before* the match based on the playing history of the two players

Algorithms

Model selection: Decision tree, random forest, and XGBoost classifiers were used before settling on the decision tree as the model with an equal balance of performance and interpretability. All models were evaluated using ROC, as well as a specific breakdown of recall and precision to see the tradeoff between the two metrics for each model
Data Split and Validation: The entire training dataset was split into 80/20 train vs. holdout, and all scores reported below were calculated with 4-fold cross validation on the training portion only. Predictions on the 20% holdout were limited to the very end, so this split was only used and scores seen just once.

Feature Engineering

1. Creating the target variable “upset”
   a. Calculate Rank Delta= winner_rank - loser_rank
   b. 1 if If Rank Delta > 0 , else 0
2. Converting categorical features to binary dummy variables, and dropping one category to avoid multicollinearity
3. Creating elo_winner and elo_loser: The Elo Rating is a well known probability calculation that takes into account player ranking to determine a match outcome. In terms of interpretation, if a player has an Elo rating of 1,800 and his opponent has a rating of 2,000, the probability of the lower Elo rating player winning becomes is 24.1%. Learn more about it here. I utilized a helper functions created by Edouard Thom to calculate the elo_probabilities

Communication

The Streamlit web application allows users to see different visualizations of the dataset and the model results through a web app. Some of the main visualizations were the different upsets occurring over the period of the dataset (2000-2022), which tournaments had the most upsets of a given year, and different performance metrics of the model. Since the XGBoost is the best performing model, I plotted the ROC Curve with a sliding threshold of the decision probability and plotting the feature importance. In order to deploy the app, I would need to utilize cloud based data storage which is something to work on for future considerations.

Future Considerations

While much of the data pipeline for the project can be executed within a few lines of code, there are additional considerations for future execution that can further improve the data engineering pipeline. For data ingestion, the beautifulsoup package could be use to automatically scraping website for updated CSV files when new tournament data is provided. As the data size and model complexity increases, PySpark would be a useful tool that could be easily integrated with the existing Python codebase to manage big data and extensive model training such as hyperparameter tuning or neural networks. For Streamlit app deploying, Google/AWS could be used as an online data storage solution and deploy the app via Github.