Premier Markets

Data Driven Market-Making for Sports

Iram Liu, Sam Meisner, Noah Plant, Rohan Shah, Alice Um, Cody Torgovnik, Corey Wang, Jerry Wang

TABLE OF CONTENTS

Team Members

Iram Liu

Samuel Meisner

Rohan Shah

Jerry Wang

Corey Wang

Cody Torgovnik

Alice Um

Noah Plant

PROBLEM STATEMENT

01

INTRODUCTION

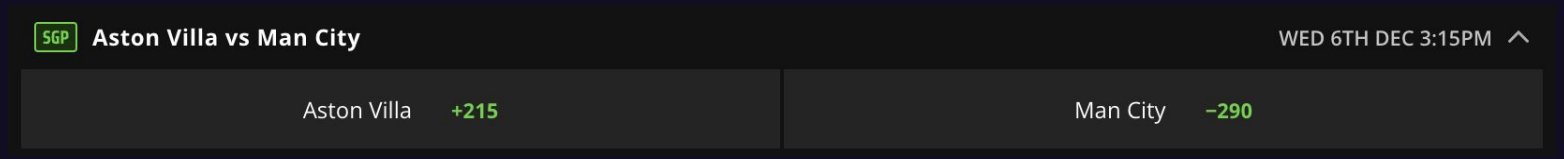
PROBLEM

Unlike financial markets, sports markets are drastically inefficient, with market makers typically charging around a **10% fee,** or *vig* for a “coin toss” event.

OUR IDEA

We believe this inefficiency creates an opportunity for a new market participant. We we can leverage data to profitably market make while **charging a lower vig.**

Example (ignores draw)



31.7%

74.4%

= 106.1%

DraftKings charges 6.1% VIG here

We think we can beat this

MACHINE LEARNING

02

Logistic Regression Model

Data

Features: Multiple bookmaker pre-game odds

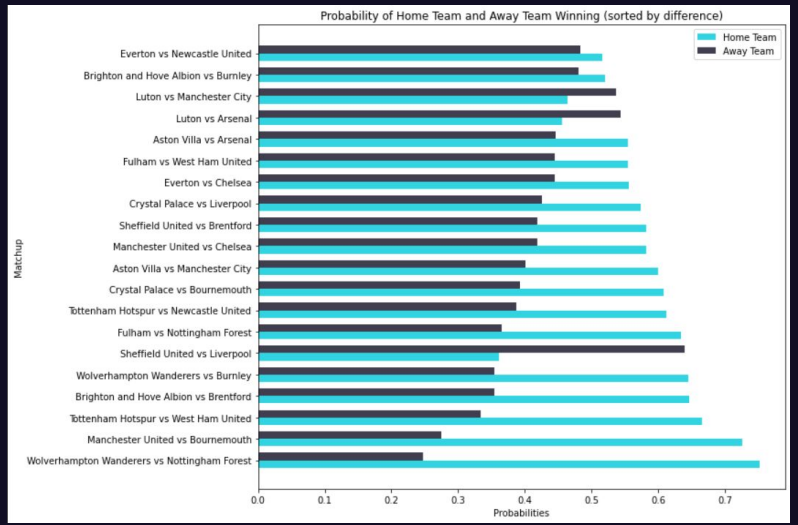
Labels: Home or away win

Model Outputs

Logistic regression returns home vs away win probabilities

Model Accuracy

82% Testing Accuracy, compared to **72% Baseline** Accuracy (pick the favorite)



Neural Network

Layers

- Linear → ReLU → Dropout → Linear → ReLU → Dropout → Linear → Softmax

Problem

- Need probabilities for initial lines
- Softmax?

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Next Steps

- Separate games into ranges depending on how likely a game is skewed (0%-10%, 10%-20%, etc.)
- Identify ranges where the neural network does a better job

MODEL DESCRIPTION

03

Naive Model

Inputs: Money on A, Money on B, Liabilities (payouts) on each ($), desired vig (%)
Outputs: Bookmaker's updated (balanced) odds

```
If (Bets on A + B < Liability if A wins): # we can't afford A to win
    Prob A = Prob A + constant # charge more for bets on A
    Prob B = Prob B - constant # incentivize bets on B

Else if (Bets on A + B < Liability if B wins): # flipped logic
    Prob A = Prob A - constant
    Prob B = Prob B + constant
```

Increment odds by fixed c based on order flow.

Improved Model

- Reflects nature of betting markets using beliefs of bettors
- **Assumption**: every bettor enters the market with a personal belief about the outcome of an event
- Model these beliefs with two distinct distributions to reflect divergent perspectives
- Use a weighted average formula to re-calibrate probabilities

Model v4

Main Improvements

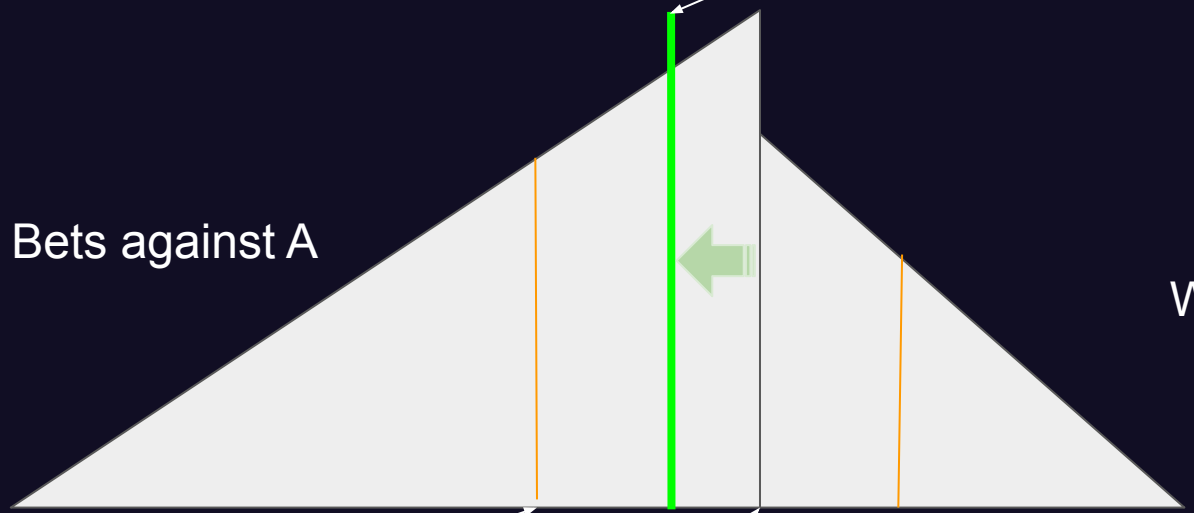
- Estimate distribution of market sentiment based on bet flow
- Expect to converge to true probability A wins → 1-1 conversion to our odds
- *Assumptions: we must assume sentiment follows a fixed distribution*

Given approximated densities of each side of a triangular distribution, estimate its mean.

*Density is **weighted** by **money** on each side.*

Expected Probability Model

$$A_{new} = (E_{not\ a} * W_{not\ a} + E_a * W_a)/(W_a + W_{not\ a})$$



$W_{not\ a}$ = Bets against A

W_a = Bets on A

$E_{not\ a}$ = Avg probability against A
= A * sqrt(2)/2

A = Current Bookmaker's Prob A

E_a = Avg probability on A
= A + (1-A)(1- sqrt(2)/2)

Step-by-step Algorithm

1. Aggregate odds from competing market-makers
2. Pass those odds into trained ML model, receive new probabilistic outputs
3. Bets come in
    a. If we can't afford an outcome, push odds to what we think we *can* afford (plus error margin)
    b. Calculate sample distribution of market sentiment on team A (weighted by total money), estimate p(A wins)
    c. Go back to 3.
4. Observe outcome

Naive Model

Bet Comes in

Update:
- Bets on A
- Bets on B
- Liability if A wins
- Liability if B wins

If (Bets on A + B < Liability if A wins):
    Prob A = Prob A + constant
    Prob B = Prob B - constant
Else if (Bets on A + B < Liability if B wins):
    Prob A = Prob A - constant
    Prob B = Prob B - constant

Update:
- Prob A
- Prob B

BACKTESTING RESULTS

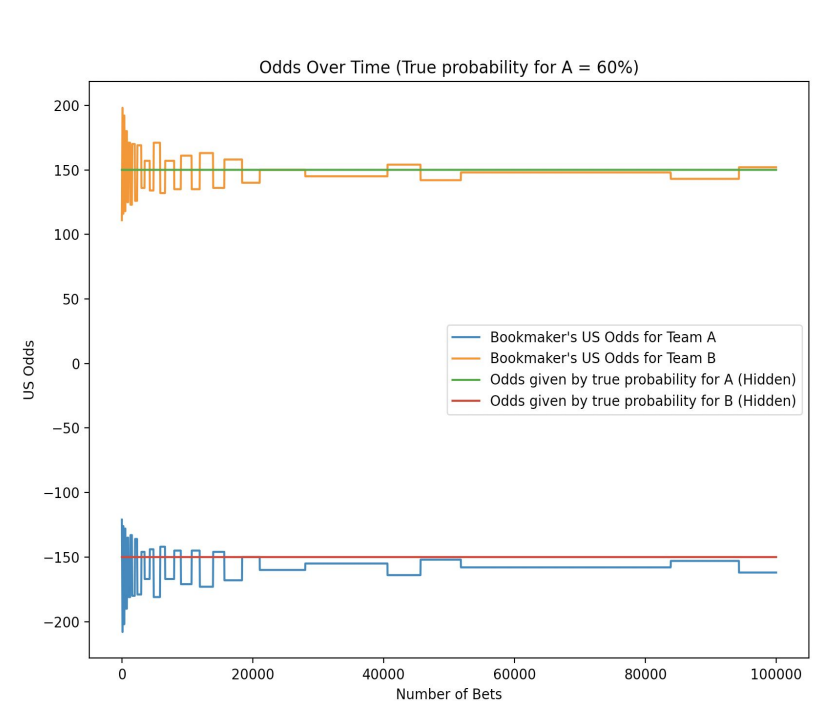
04

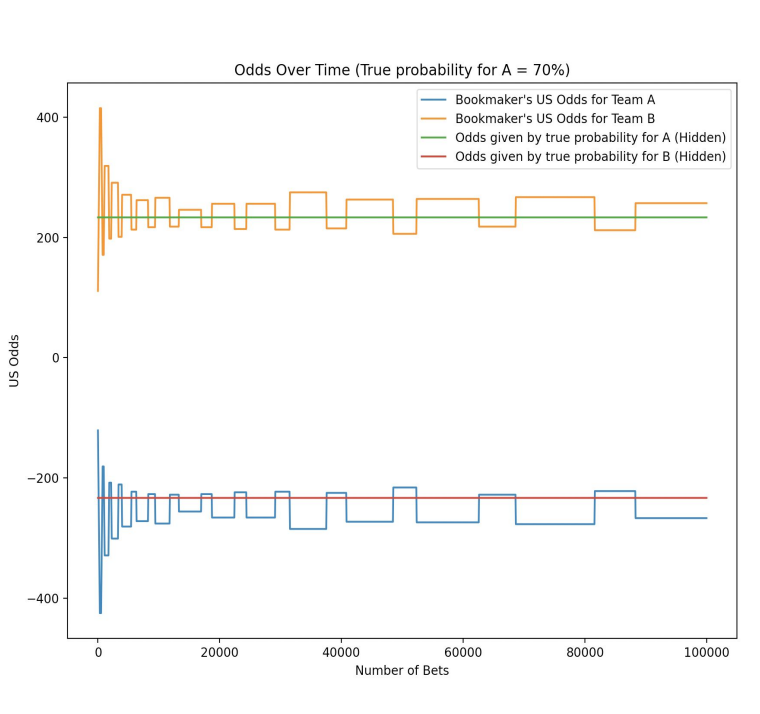
Backtesting Algo

Most information isn't available...

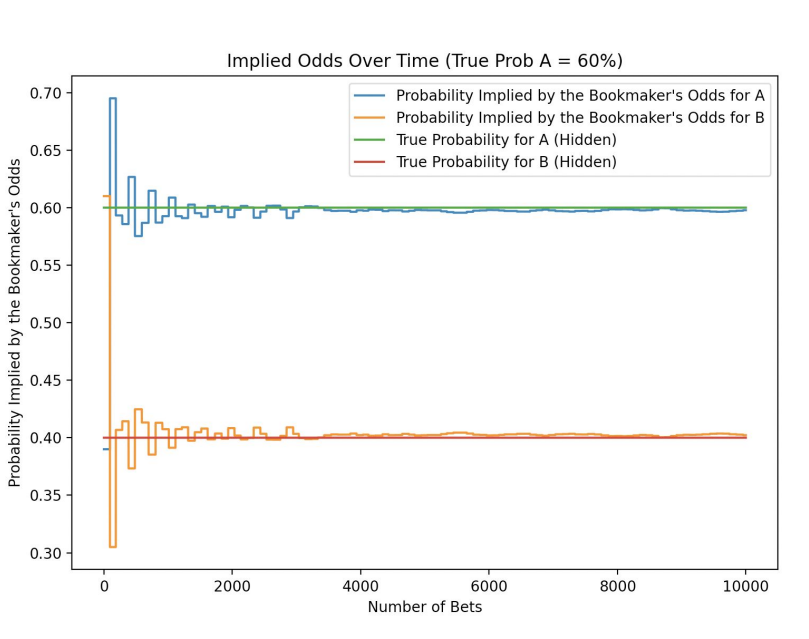
1. Initialize p(A) = 50% (conservative, weak guess)
2. Randomly set true p'(A) (hidden from algo)
3. Bets come in (randomly from hidden distribution)
    a. If we can't afford an outcome, push odds to what we think we *can* afford (plus error margin)
    b. Calculate sample distribution of market sentiment on team A (weighted by total money), update p(A). (estimate)
    c. Go back to 3.
4. Flip a weighted coin p=p'(A) to determine outcome, record PnL

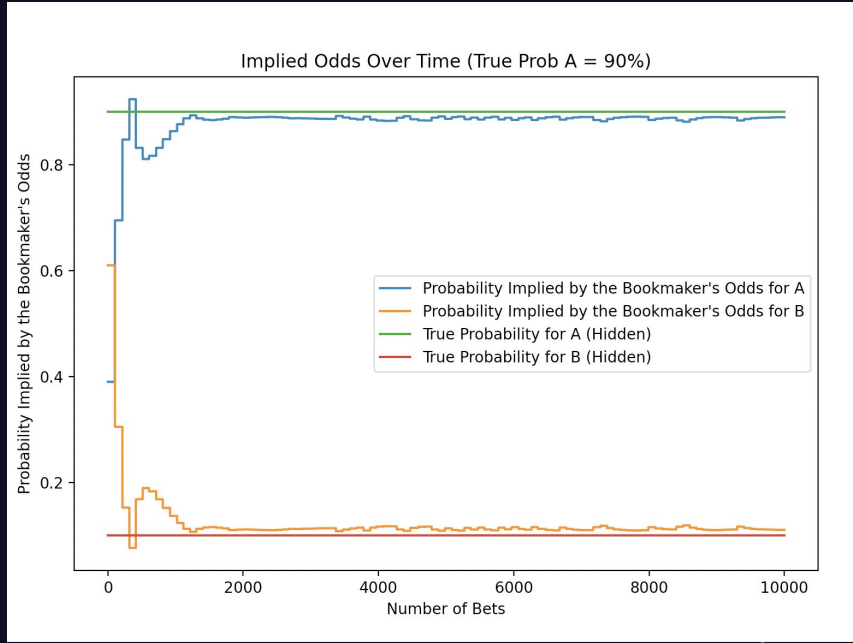
Model 1 Results - Convergence of Odds



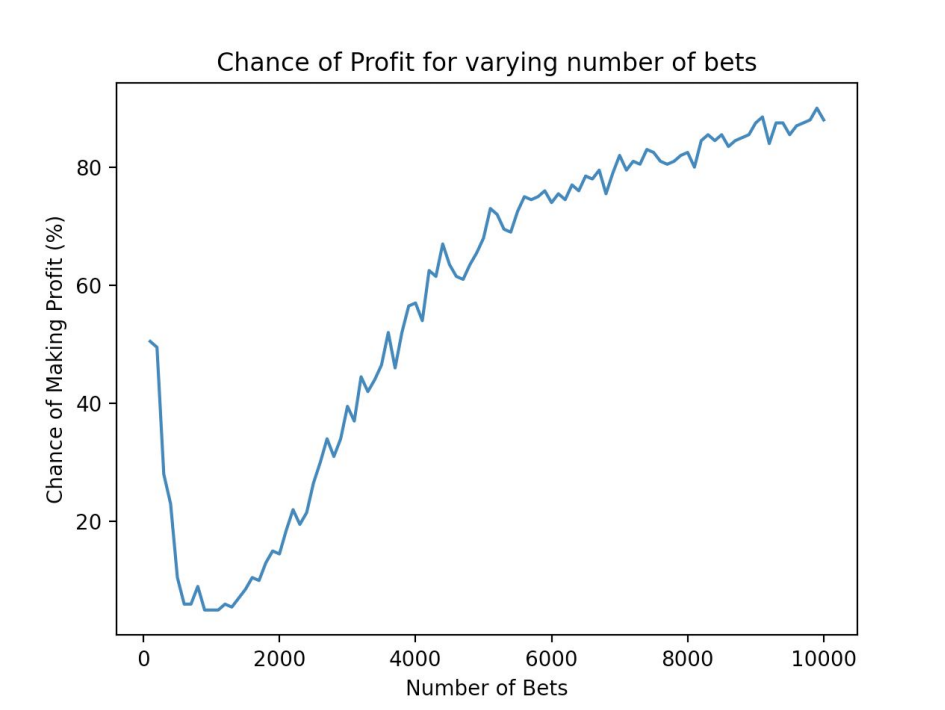


Model 2 Results - Convergence of Odds



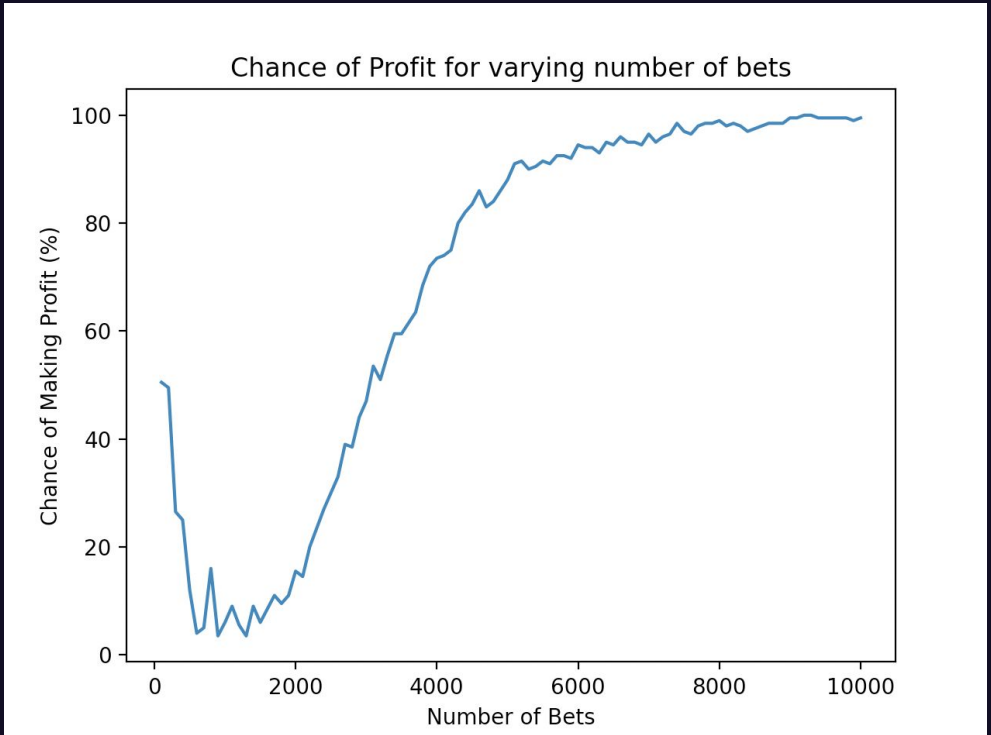


How many bets do we need? Model 1



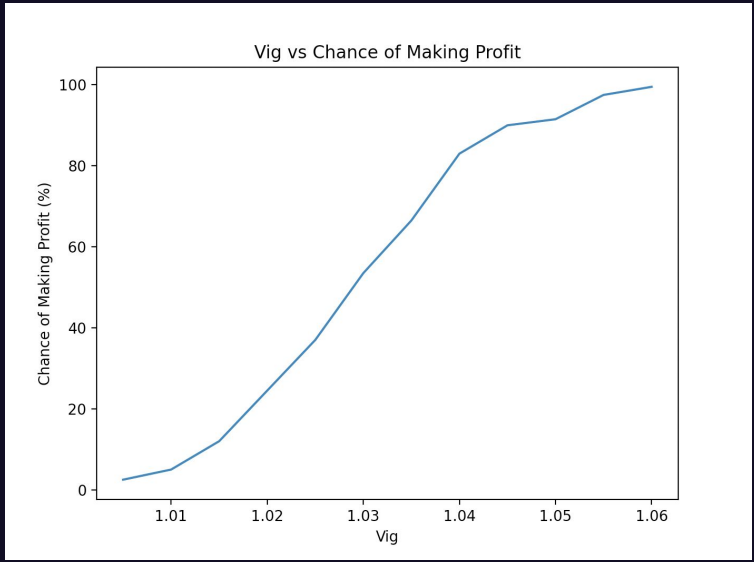
Y = chance of making profit if we stopped the simulation with **X** number of bets.

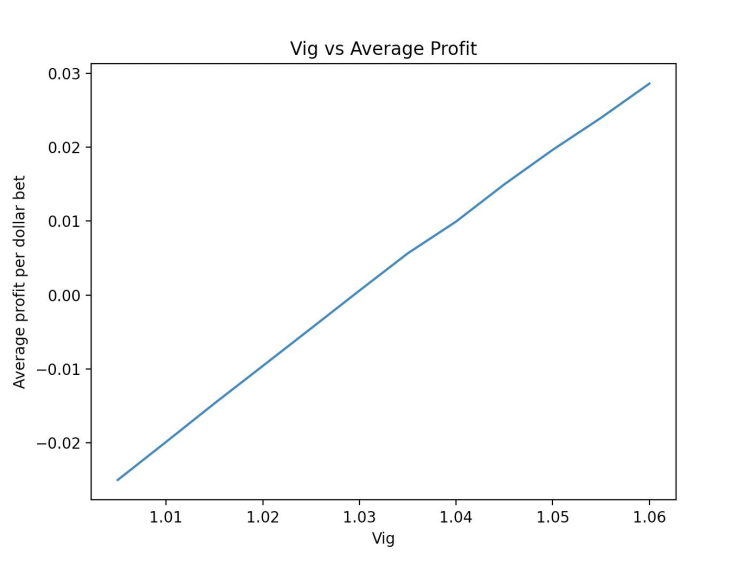
How many bets do we need? Model 2



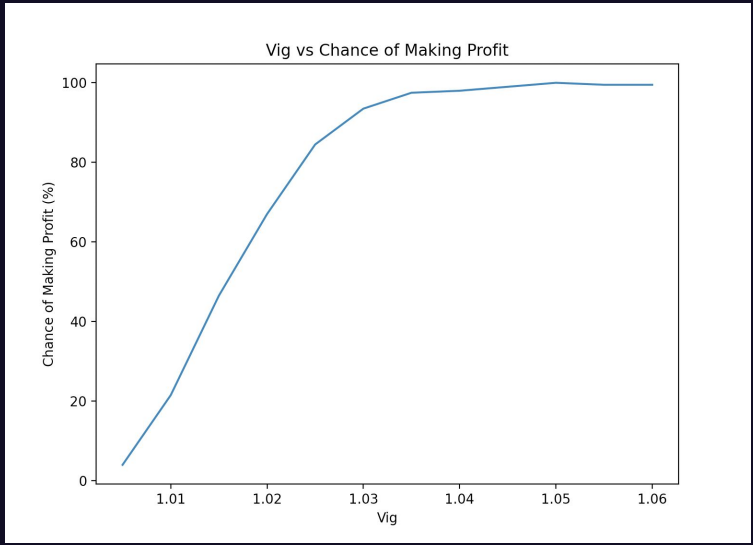
Y = chance of making profit if we stopped the simulation with **X** number of bets.

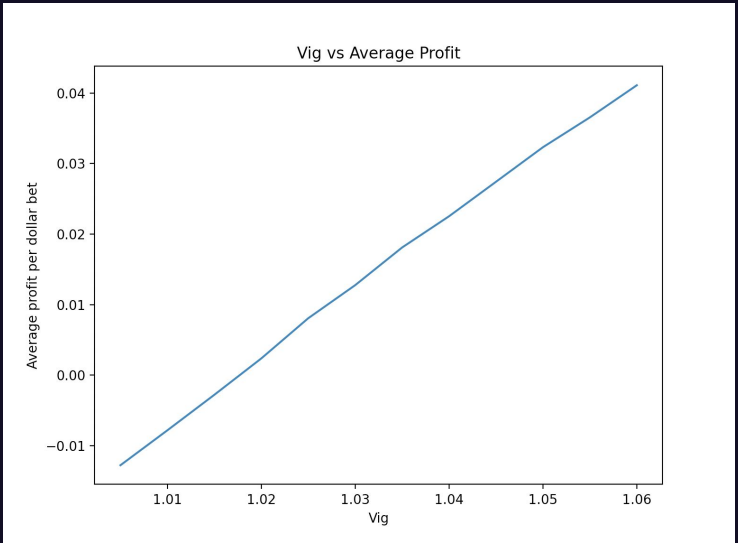
Vig vs Profit: Model 1



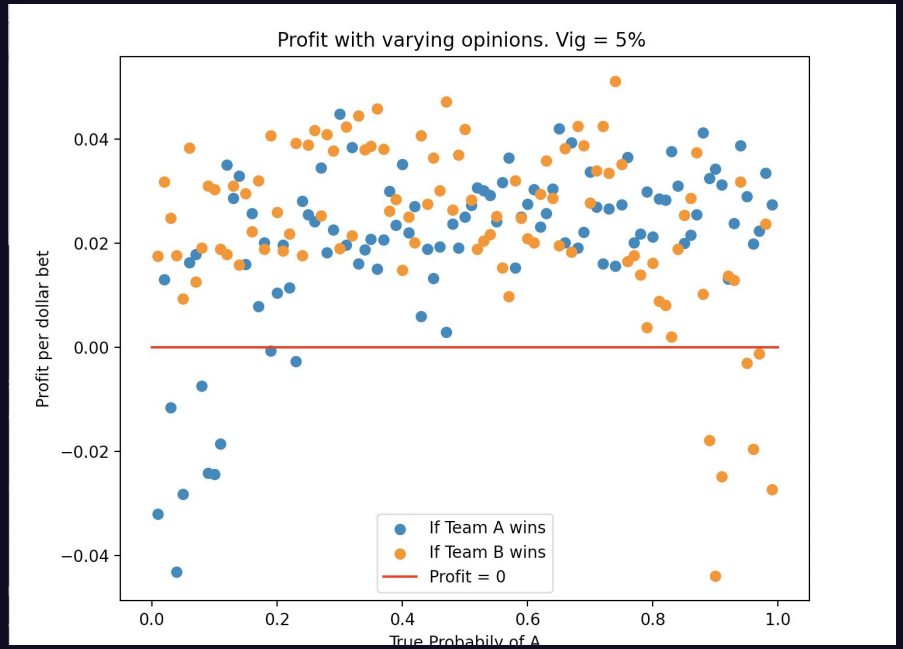


Vig vs Profit: Model 2





Profitability of *Naive Model* (5% VIG)



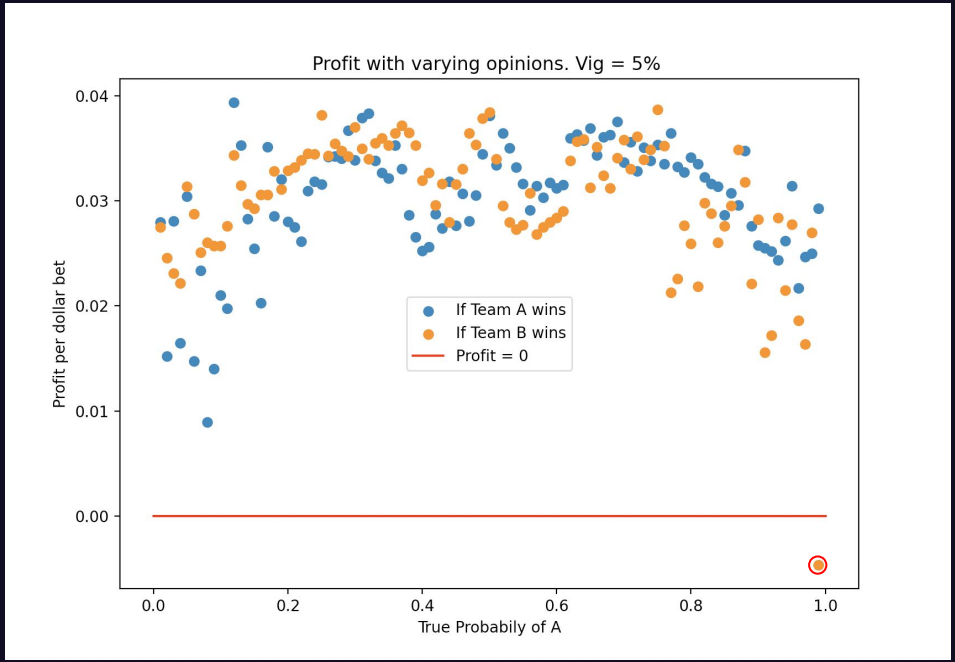
Average Profit = $0.0219 per $1 bet = **2.19%**

Games Losing Money: **8.5%**

Average Loss if we do lose money

= - $0.0178 (per $1) = **-1.78%**

Profitability of *Updated Model* (5% VIG)



Average Profit = $0.0302 per $1 bet = **3.02%**

Games Losing Money: **0.5%**

Average Loss if we do lose money = - $0.00138 (per $1) = **0.138%**

BACKEND + USER INTERFACE

05

Backend Design

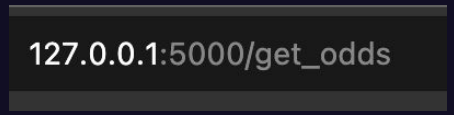
Utilized **Flask** for backend design

- Interfaces easily with our odds API
- Provides easily definable endpoints (functions) on local server important to rebalancing, generating initial betting lines, and accessing data in the frontend

AWS Databases

- Used AWS CDK to define database schema in backend
- Allows us to write new matchups, keep track of accounts with the application, and keep track of historical user bets on previous matchups

```python
@app.route('/get_odds')
def get_odds():
    data = database.get_all_games()
    return data
```



{"Count":25,"Items":[{"away_team":"Manchester City","away_team_odds":"-345","game_id":"67c3f64f5a61b1b7c10e7120c36af5b0","home_team":"Tottenham Hotspur","home_team_odds":"900","num_bets_since_rebalance":"0","val_bets_away":"0","val_bets_away_since_rebalance":"0","val_bets_home":"0","val_bets_home_since_rebalance":"0"},{"away_team":"Manchester United","away_team_odds":"-165","game_id":"3b0d03c60d2694bb23d5e0d5abf5464b","home_team":"Bournemouth","home_team_odds":"450","num_bets_since_rebalance":"0","val_bets_away":"0","val_bets_away

User Interface

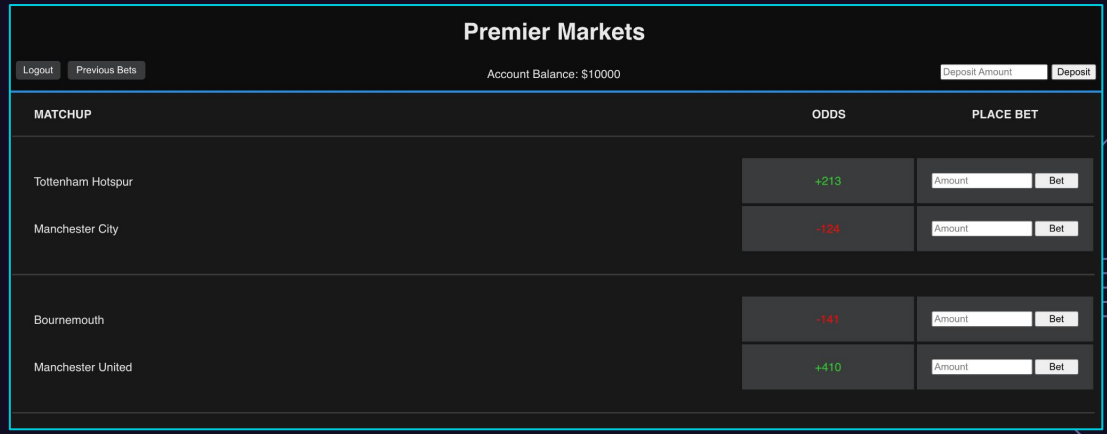
Used **React**

Connected to Flask server to get necessary data for display

Login Page: calls the user's information from DB (AWS)

Home page: displays upcoming odds, allows user to bet on them

Previous bets: shows previous bets placed by user logged in



DEMO

06

NEXT STEPS

07

Scaling

Application

- Expand to more sports
- Incorporate other significant variables

API New and Old Concurrency

- Run new games from the API through the model to get initial lines without overwriting updating odds
- Handle user inputs and updated odds with hundreds of users

Scaling

AWS

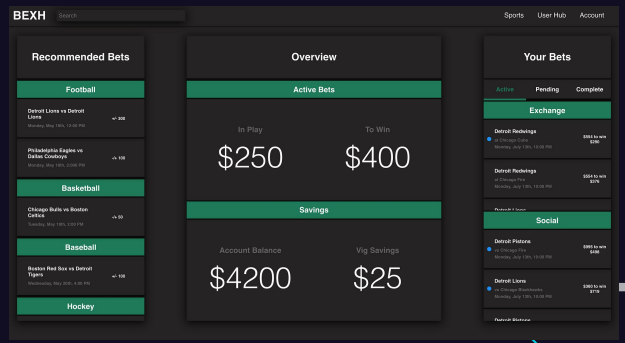
- Receive funding for larger database access (currently using personal account)

Improve ML Model

- Find alternative API and web scrape for more past match data
- Achieve better model accuracy for initial odds (optimize)
- Find other sources of data to gather public consensus (Twitter?)

Better UI/UX

- More functionality
- Increase user interactivity



Appendix

08

Bet Comes in

Update:
- Bets on A
- Bets on B

*Resets Every 100 bets

When we have 100 bets...

$$\text{Prob } A_{new} = (E_{\text{not a}} * W_{\text{not a}} + E_a * W_a) / (W_a + W_{\text{not a}})$$

Update:
- Prob A
- Prob B = 1 - Prob A