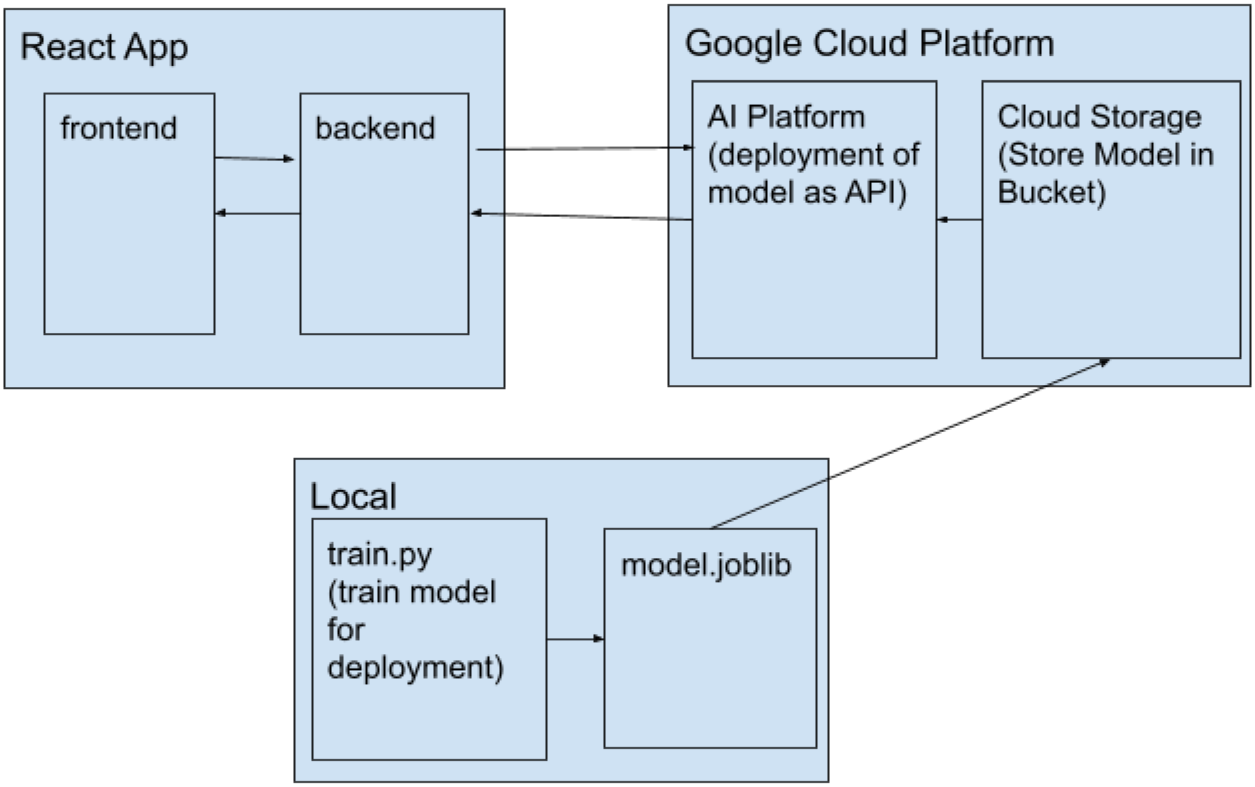
ML Deployment Architecture

Diagram


React App
frontend
backend
Google Cloud Platform
AI Platform (deployment of model as API)
Cloud Storage (Store Model in Bucket)
Local
train.py (train model for deployment)
model.joblib

Deployment Architecture Design Decisions

My deployment architecture consists of a React Web application and a Google Cloud Platform API. To start I will train the model locally and save it as a .joblib file. Then I will load this model into a Google Cloud Storage bucket. I can then deploy this model to an API on the Google AI Platform. Then with the right credentials I can access this API with a python script or curl request. This API request takes in 71 features for each team requested and returns a run prediction for each team. I choose to deploy my Machine Learning model this way because I found it to be the easiest to actually deploy and access. At first I tried to deploy my model on paper space because I had credits there but there was not much documentation on how to deploy the model I was working with. So, I then decided to switch platforms and I found that Google Cloud Platform had many tutorials and documentation on how to deploy models on their platform. To update the model all I have to do is add the new model to a new bucket on the Google Cloud Storage and then deploy a new version of the model on the AI Platform. This theoretically could be done everyday to keep the model the most up to date, but this would be excessive for how little traffic the model/api gets. For now updating the model at most once or twice a season would be sufficient until the website brings in more traffic. The Google Cloud Platform console also lets you monitor the health of the model deployment and its active status. After running the deployment for a few days it seems like my monthly estimate to run the API is about 150$. The only downside to the Google Cloud Platform deployment is that the API needs credentials to use so it's harder for outside users to make requests to it. The second half of my deployment architecture is a React App with a react frontend and a Node.js and python backend. The frontend lets users select an mlb match for the day and then it displays each team's predicted runs. The backend makes a request to the Google Cloud Platform API giving it each team's records coming into that game. The API returns the two predicted runs back to the backend and it is processed and passed back to the frontend. This application will be deployed for free on heroku and will be able to be accessed by anyone who has the link.

Pre-Deployment Checklist

1. Define the Problem
    a. What's the Problem
        i. The problem is trying to predict how many runs an mlb team will score in their next game
        ii. assumption we know the team record coming in
        iii. Similar to trying to predict stock prices kind of
    b. Why does the problem need to be solved
        i. It can help odds setters or improve sport gambling abilities
        ii. Could be used by other ML pipelines to fill in missing data
    c. How could the problem be solved manually
        i. Right now people just look at teams records and stats and make a guess
        ii. Som may use more advanced statistics and other metrics to make these predictions
2. Prepare Data
    a. Data Description
        i. There are stats for many different game results on baseball reference like who won, home away, ERA, batting average, and much much more
        ii. It would be great to have all the dat just more organized and not as hard to get
    b. Data Preprocessing
        i. I used the pybaseball python library to request data then I wrote a script to clean and format the data into a CSV
        ii. I clean and analyzed the data using pythons matplotlib and seaborn libraries
    c. Data Transformation
        i. I transformed the data by adding one hot encoding for the team and opponent features
    d. Data Summarization

i. The EDA.py script plots many of the features against the runs and plots the on a heat map as well

3. Spot Check Algorithms
   - a. Create Test Harness
     - i. I use scikit learn train test split function
     - ii. I evaluate my model using mean squared error and percent correct within certain ranges
   - b. Evaluate Candidate Algorithms
     - i. I tried linear regression, random forest and xgboost algorithms
     - ii. This is in train-and-test-various-models.ipynb
4. Improve Results
   - a. Algorithm Tuning
     - i. I did multiple grid searches over the best two models parameters to find which were best
     - ii. This is in train-and-test-various-models.ipynb
   - b. Ensemble Methods
     - i. One of the models being tested was xgboost which is an ensemble model
   - c. Model Selection
     - i. I selected the best model with the best parameters
     - ii. This is in production-ml-code.ipynb
5. Finalize Project
   - a. Present results
     - i. https://github.com/sethvanb/SpringBoard
   - b. Operational Results
     - i. https://mlb-run-predictor.herokuapp.com