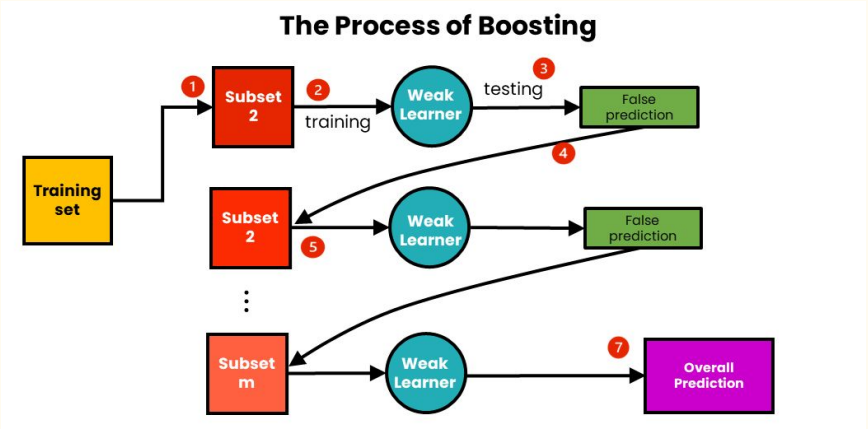
Boosting Algorithms

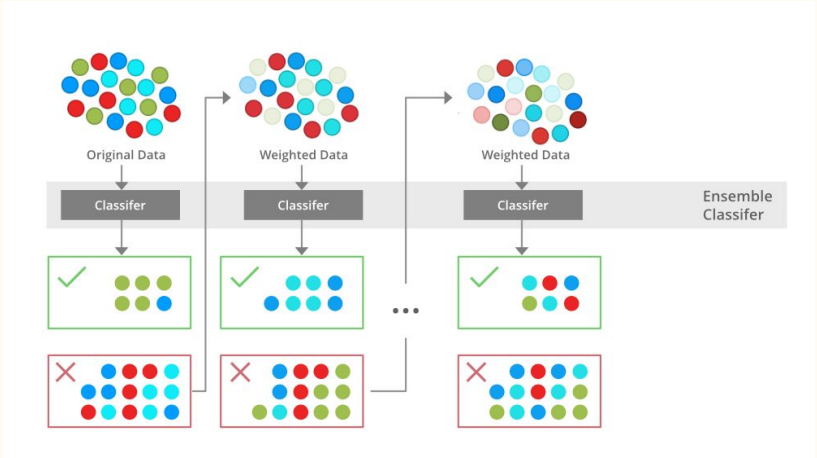
By Will and Adam

What are Boosting Algorithms?

- Boosting improves machine learning models' predictive accuracy and performance by converting multiple weak learners into a single strong learning model.
    - Weak Learners: Weak learners have low prediction accuracy, similar to random guessing. They are prone to overfitting
        - If you train an algorithm to identify dogs by 4 legs, it might fail to classify a legless dog
    - Strong Learners: higher prediction accuracy, often close to perfect. We achieve this in Boosting by combining multiple weak learners
        - Now identifies dogs by combining weak learners of fur, number of legs, ear shape, etc. This makes the learners less prone to misclassification based on new data.
- 2 Main Types of Boosting Algorithms:
    - Ada Boosting
    - Gradient Boosting
        - xgboosting

A visual Aid to Boosting





The Full Ada Boosting Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ **where** $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- **Train weak learner using distribution** D_t.
- **Get weak hypothesis** $h_t : X \to \{-1, +1\}$ **with error**

$$\epsilon_t = \Pr_{i \sim D_t} \left[h_t(x_i) \neq y_i \right].$$

- **Choose** $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.
- **Update:**

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

A deep dive into Ada Boosting

- The algorithm takes as input a training set $(x_1, y_1) \ldots (x_m y_m)$ where each x_i belongs to some domain or instance space X , and each label y_i is in some label set Y
- Weight of Distribution on training example i on round t is denoted $D_t(i)$
  - Initially all weights are equal but on each round the weight of incorrectly classified examples is increased so weak learner is forced to learn.
- Each weak learner's job is to find a weak hypothesis $h_t: X \rightarrow \{-1, +1\}$ appropriate for a distribution D_t
- The error ϵ_t is the probability of h_t misclassifying an example in distribution D_t
  - Note that $\epsilon_t < 0.5$ as if not then the learner is worse than random guessing, which should never happen

More on Ada Boosting

- Once weak Hypothesis is chosen, Algorithm chooses a α parameter. Notice that as the error gets bigger α gets smaller. Thus more weight is given to the better classifiers.

$$\text{Choose } \alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right).$$

Updating the Distribution

- A new distribution $D_{t+1}(i)$ is formed
- Z_t is a normalization factor to ensure that $D_{t+1}(i)$ is a probability distribution

$$
\begin{aligned}
D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\
&= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}
\end{aligned}
$$

- Notice that this means more weight is added to the incorrect predictions, forcing the training to focus more on the incorrectly classifications.

Putting it all Together

- Calculates the weighted sum of all the predictions for each label x
- If the sum is positive, then $H(x) = 1$
- If the sum is negative, then $H(x) = -1$
- Provides a single strong classifier from a bunch of weak ones

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

What if more then two classes?

- If there are more than two classes then we train each classifier in a 1vs all approach
- For k classes and any instance i, $y_{ik} = 1$ if a member of the k class and $y_{ik} = -1$ otherwise

Why Ada Boosting Rocks

- Can write error rating as $\epsilon_t = \sum_{i:h(x_i)\neq y_i}^{T} D(i)$. However as this is a combo of weak learners we can rewrite this as $\epsilon_t = \frac{1}{2} - \gamma_t$ as our error is slightly better than random guessing.
- Freund and Schaipaire prove that the Training error of final Hypothesis $H(x)$ is at most

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_t \gamma_t^2\right).$$

- Thus if each $H(x)$is slightly better than random, then the training error drops exponentially fast

What is a Gradient Boosting

- Gradient Boosting is a generalization of ADA boosting.
- Gradient Boosting was made to handle a variety of loss functions
- AdaBoost adjusts the weights of instances, and Gradient Boosting focuses on minimizing a loss function by adding new models that fit the residuals of the previous ones using gradient descent.

The Full Gradient Boosting Algorithm

Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. for $m = 1$ *to* M:

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ $for\ i = 1,\ldots, n$

2-2. Train regression tree with features x against r and create terminal node reasions R_{jm} for $j = 1,\ldots, J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ $for\ j = 1,\ldots, J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

Deep Dive into Gradient Boosting

- Algorithm Begins with an Initial Model F_0

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

- *argmin* means we are searching for the value γ that minimizes $\Sigma L(y_i, \gamma)$.
- Take the derivative F_0 and solve for γ,
- Turns out that γ = the mean of yi which is important as that is often our first guess for F0

Look Familiar?

- Compute the residuals by taking the negative partial of the loss with respect to F(xi) where F is the last model used

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ $for\ i = 1,\ldots,n$

- This turns out being that $r_{im} = -2(y_i - F_{m-1}(x_i))$
- We can take 2 out because its a constant

Training the Regression Tree

- A regression tree is trained using the features x against the residuals r from the previous step. Each tree thus learns to predict the gradient of the loss function, essentially fitting to the errors of the model ensemble thus far. This aims to put instances with similar residuals in the same region.

2-2. Train regression tree with features x against r and create terminal node reasions R_{jm} for $j = 1,\ldots, J_m$

- R_{jm} is the region (or the set of data points) corresponding to the j-th leaf of the m-th tree.

Computing γ_{im}

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \;\; for\; j = 1,\ldots, J_m$

- We are searching for γ_{jm} that minimizes the loss function on each terminal node j.
- $\frac{d}{d\gamma}\left(\sum_{x_i \in R_{jm}} (y_i - (F_{m-1}(x_i) + \gamma))\text{^}2\right) = -2\left(\sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)\right)$
- Next we solve for γ when the derivative is 0
- $-2\left(\sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)\right) = 0$
- $\sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i)) = n\gamma$
- $n\gamma = \sum_{x_i \in R_{jm}} (r_{im})$
- $\gamma = \frac{1}{n} \sum_{x_i \in R_{jm}} (r_{im})$
- The γ that minimizes the loss function is the average of the residuals of a terminal node in our tree model

Updating the Model

- Finally, we update the model by adding the output of the current tree multiplied by a learning rate v
- Here, $\mathbf{1}(x \in R_{jm})$ is an indicator function that is 1 if x falls in the region R_{jm} and 0 otherwise
- Main purpose of this step is to fit the new weak weak learner to the residuals of the previous model, thus targeting the most significant sources of error.

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

Final Output

- After M iterations, the final model represents the sum of the initial model and the improvements made by the M trees.
- Each tree corrects the residuals left by all previous trees
- Gradient Boosting can be used with a variety of loss functions (e.g., logistic loss for classification, squared error for regression), making it more flexible then ADA boosting in dealing with different kinds of prediction errors and more robust to outliers depending on the choice of the loss function,
- ADA boosting deals with an exponential loss function that is sensitive to outliers

XG Boosting (The Final Form)

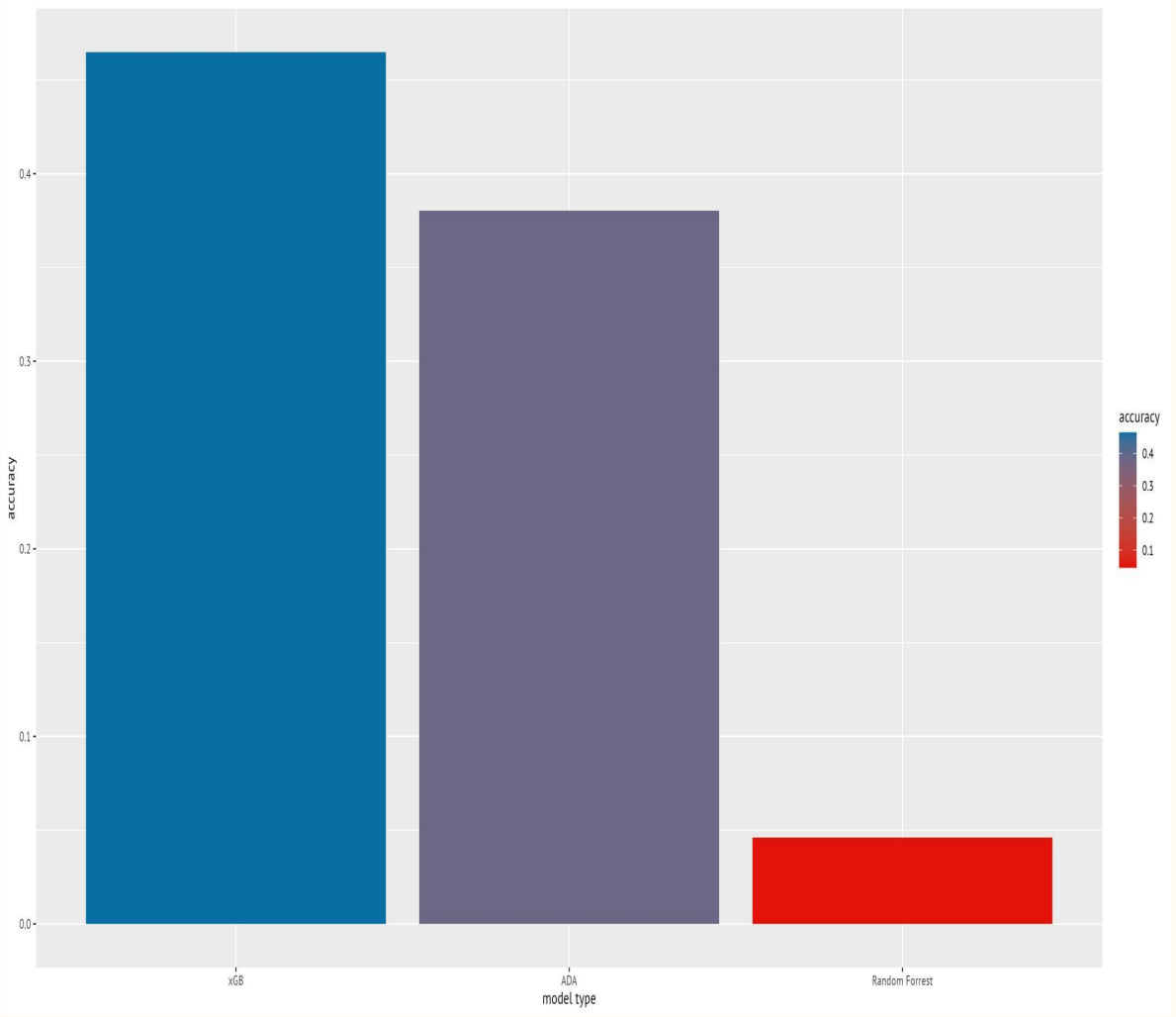
- Much more complicated Algorithm which makes sense as it's the most recent
- Basically a gradient boosting algorithm, but heavily optimized to deal with computational issues
- Includes regularization to reduce overfitting
  - Uses either lasso or ridge regularization
- Parallelization
  - Carries out multiple calculations simultaneously, constantly choosing the best option, increasing both efficiency and accuracy
- Helps handle missing data while regular gradient boosting cannot
  - Very helpful for real world datasets

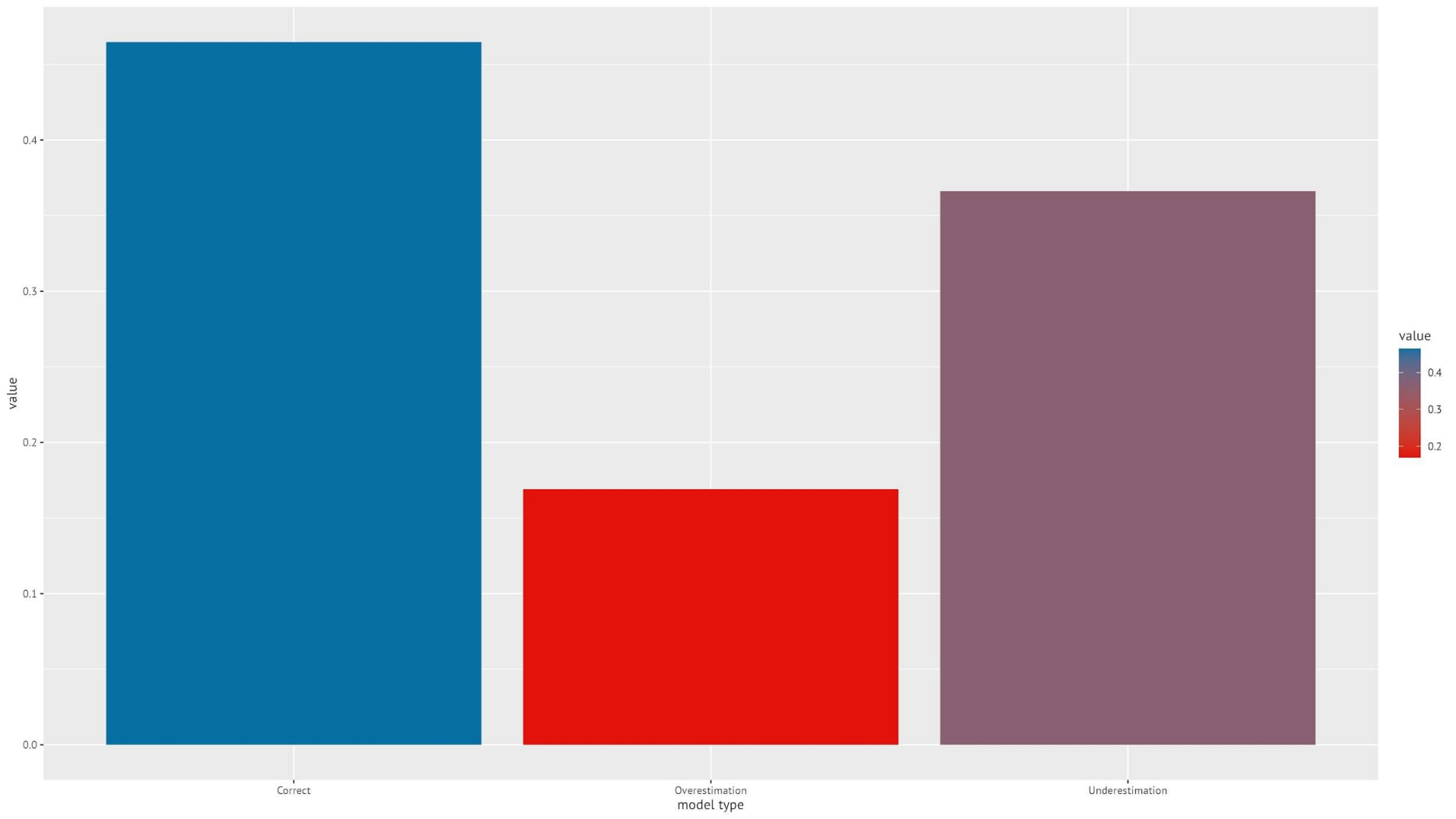
Implementation

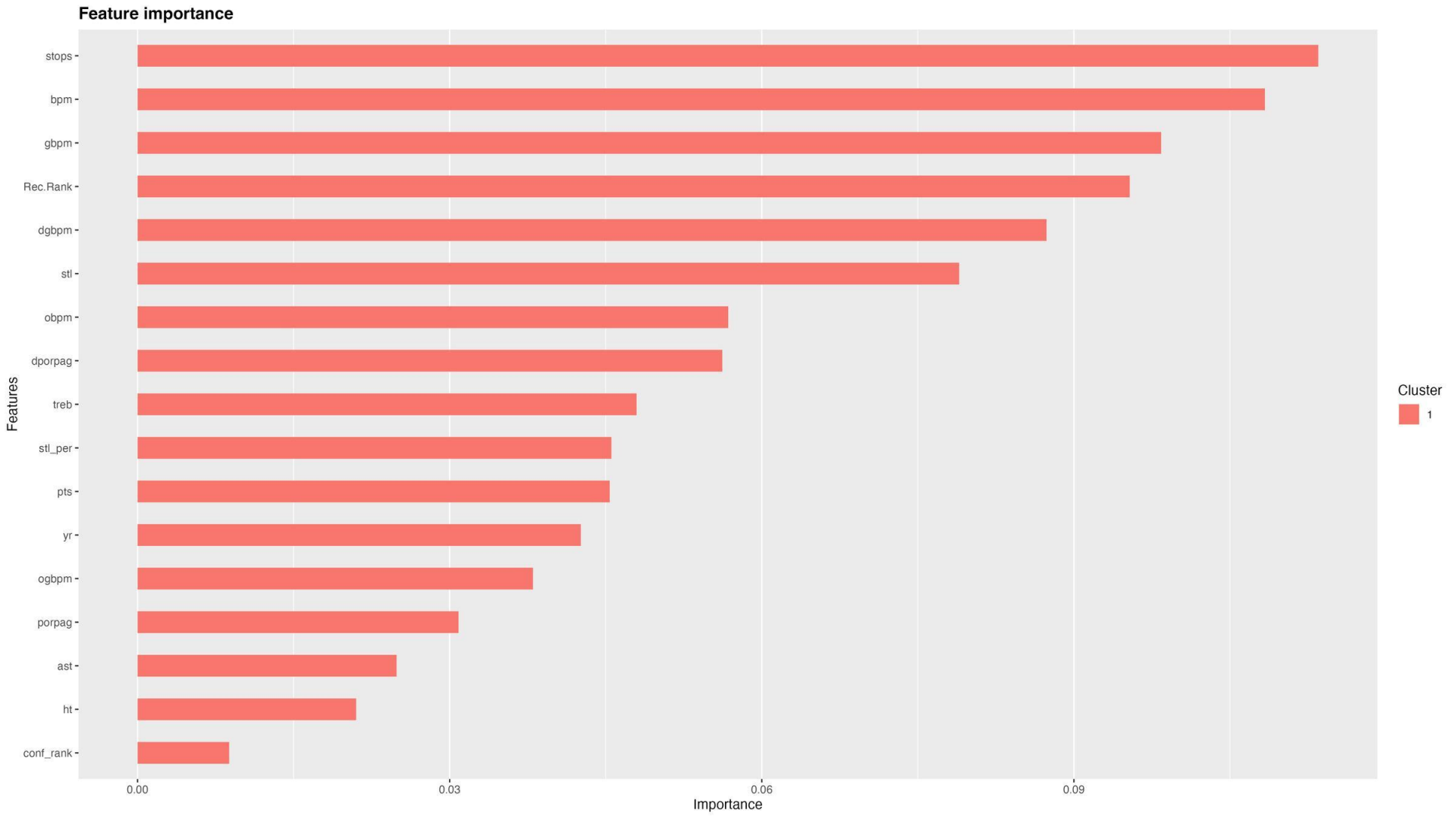
- **Task:** Build a model that uses advanced college statistics to predict a prospect's "peak" NBA future
- Defining "Peak": Using FiveThirtyEight's all-encompassing RAPTOR metric, look at a player's best single season and put it on an all-time percentile
  - Top 10%: All-Star
  - Top 30%: Starter
  - Top 70%: Role Player
  - Bottom 30%: Bench Warmer
- Training set: College stats from 2009-2023, only include the last season of a player's college career, since we're predicting the future of players in the 2024 Draft Class

Models Built

- Single Random Forrest
- ADA Boosting
- xGBoosting




0.4
0.3
0.2
0.1
0.0
value
Correct
Overestimation
Underestimation
model type
value
0.4
0.3
0.2

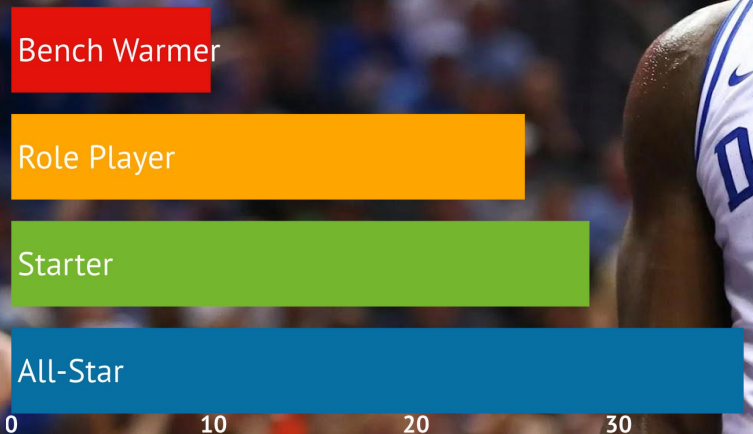

Feature importance
stops
bpm
gbpm
Rec.Rank
dgbpm
stl
obpm
dporpag
treb
stl_per
pts
yr
ogbpm
porpag
ast
ht
conf_rank
Features
0.00
0.03
0.06
0.09
Importance
Cluster
1

Zion WIlliamson

F - Duke University

10.00
Draft Score

All Time Rank: 1

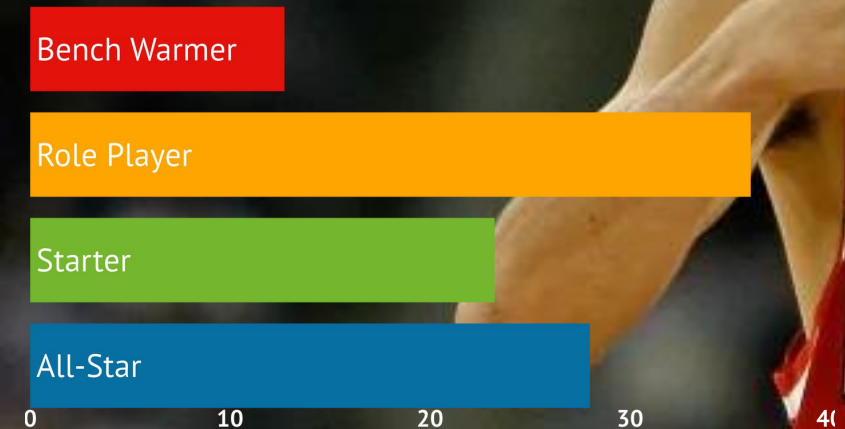


Predicted Peak: All-Star (36.19%)

Stephen Curry

G - Davidson College

9.83
Draft Score
All Time Rank: 14

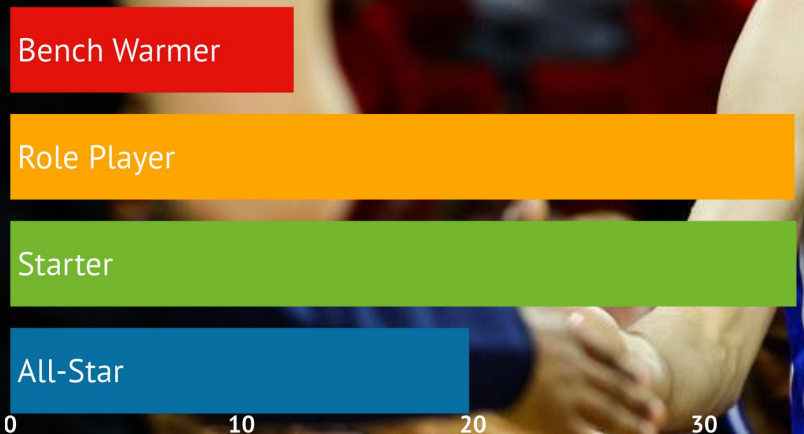


Predicted Peak: Role Player (30.69%)

Kyle Filpowski

C - Duke University

9.73
Draft Score
All-Time Rank: 23
2024 Rank: 1



Predicted Peak: Starter (34.00%)

Zach Edey

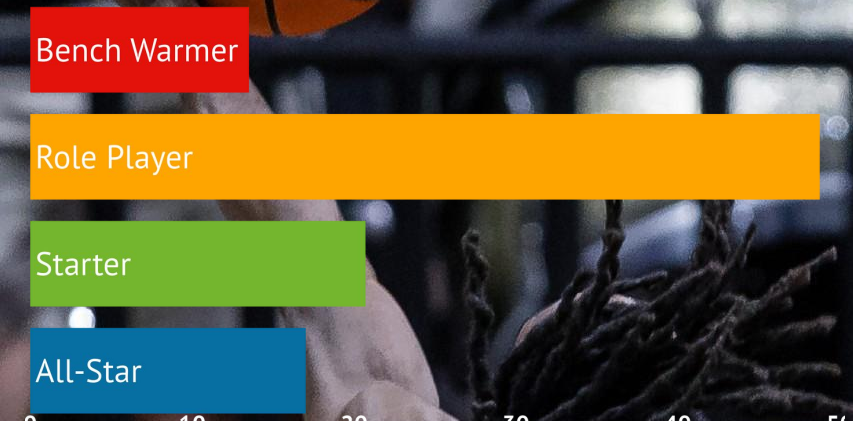
C - Purdue University

8.60

Draft Score

All-Time Rank: 114

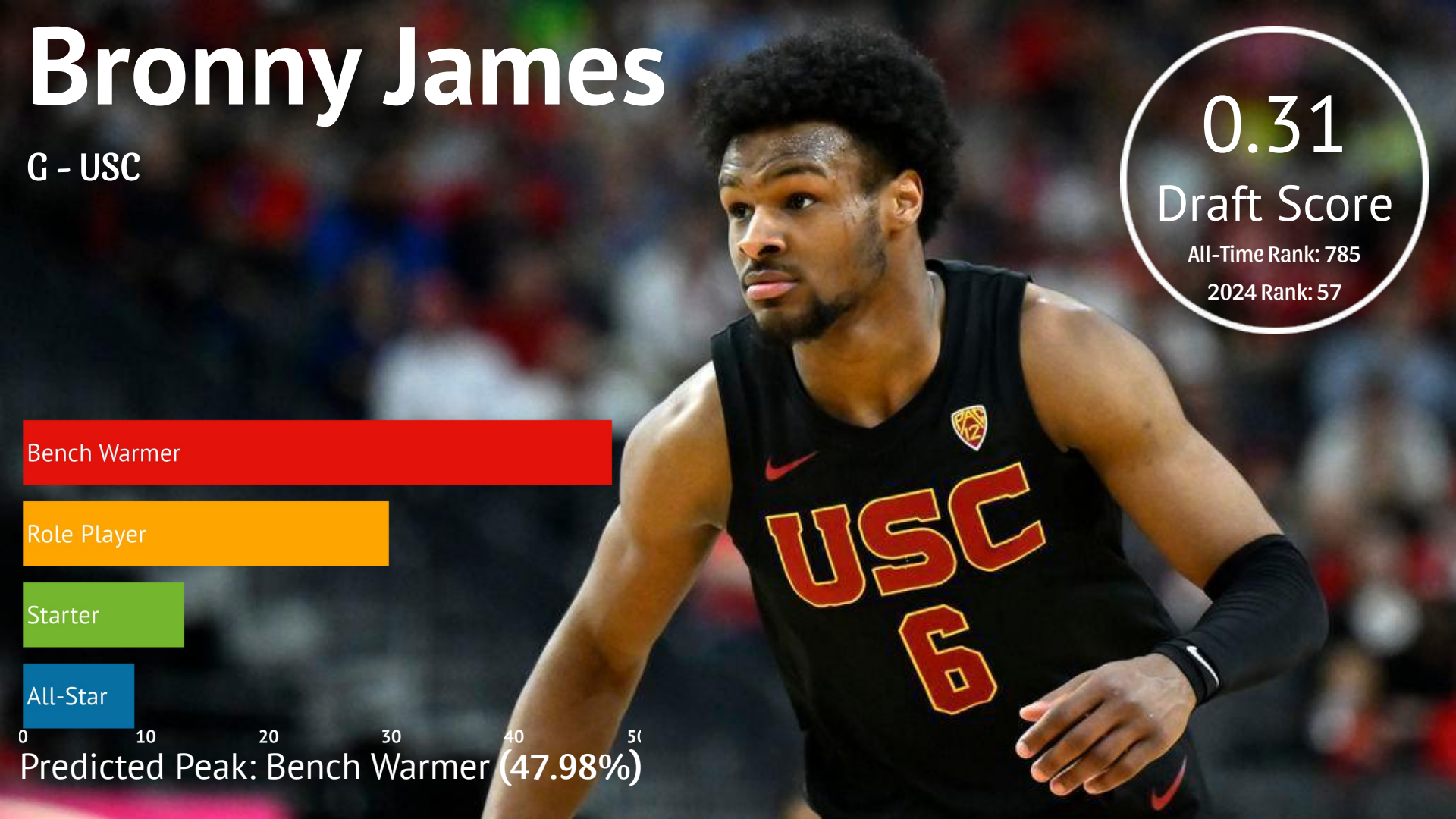
2024 Rank: 8



Predicted Peak: Role Player (48.77%)

Bronny James

G - USC



0.31

Draft Score

All-Time Rank: 785

2024 Rank: 57

Future Work

- The model has coin flip accuracy even after hyper parametrization, which speaks to the variableness of the sport and the problems with trying to predict just based off college stats
- If we had more time, using combine stats or other athleticism or workout metrics would likely give a more accurate model
- If the model had >70% accuracy, we'd be working for the Knicks

Sources:

https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf

Data: barttorvik.com, FiveThirtyEight.com