Predictive Modeling - XGBoost

DS Development Presentations

Peter Nicholas S. Onglao | MNL

Contents

- Overview of XGBoost
- Some math behind XGBoost – Gradient Boosting
- XGBoost in action – Diabetic Retinopathy classification

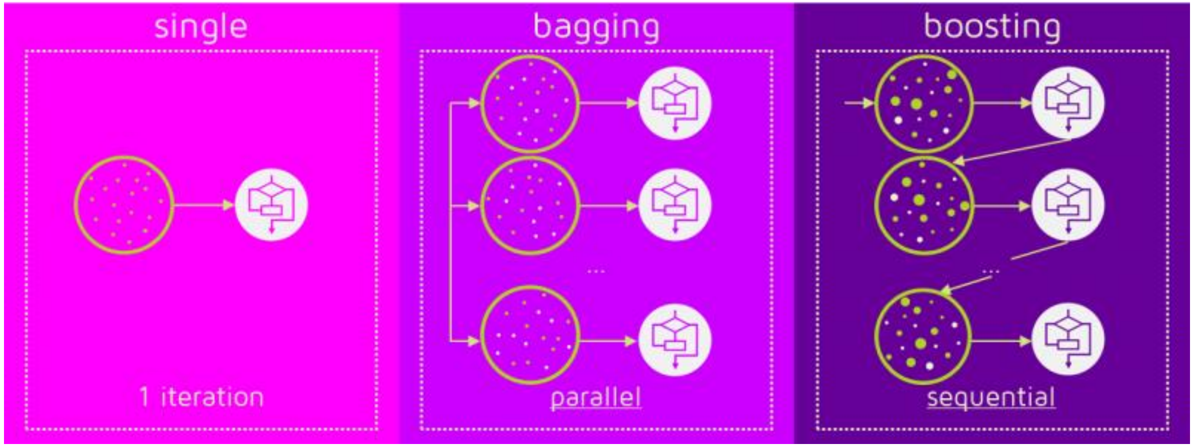
Disclaimer: changed my topic

1. Overview of XGBoost

What is it? What are its features?

What is Boosting?

- Boosting = sequential building of a strong learner from weak learners



- Commonly used weak learner = decision trees

What is XGBoost?

- Created in 2014 by Tianqi Chen
- e**X**treme **G**radient **B**oosting
  - *"The name xgboost, though, actually refers to the engineering goal to* ***push the limit of computations resources for boosted tree algorithms****. Which is the reason why many people use xgboost."*
- **Speed** & **Performance**
- Available in most popular platforms/interfaces (CLI, C++, Python, R, Julia, Scala, Hadoop)

XGBoost Features

- Model Features
  - Gradient Boosting
  - Stochastic Gradient Boosting
    - Sub-sampling at multiple levels
  - Regularized Gradient Boosting
    - L1 and L2 regularization
  - Classification and Regression

XGBoost Features

- System Features
  - Parallelization
    - Parallel node construction
  - Distributed Computing
  - Out-of-Core Computing
    - For large datasets > RAM
  - Cache Optimization

XGBoost Features

- Algorithm Features
  - Sparse Aware
    - Automatic handling of missing values
  - Block Structure
    - Supports parallelization
  - Continued Training
    - Save your current parameters

2. Math Behind XGBoost

Gradient Boosting & the Objective Function

The Objective Function

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

Objective function

Loss function

Reg'zn term

Common Loss Functions

- Regression and Classification use the **same algorithms/formulas**, just **different loss functions**
- Regression – **MSE**

$$L(\theta) = \sum (y_i - \hat{y}_i)^2$$

- Classification – **log loss**

$$L(\theta) = \sum \left[y_i \ln\left(1 + e^{-\hat{y}_i}\right) + (1 - y_i) \ln\left(1 + e^{\hat{y}_i}\right) \right]$$

Decision Tree Ensemble = Adding up the trees

- The prediction for individual *i*

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

K = # trees

f = function in F; a tree

F = set of all possible CARTs (class'fn and regression trees)

Updating the objective function

- The loss function within obj can be expressed as:

$$obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

- Optimize this!
- Find a way to express $\hat{y}_i$
  - Calculating $\hat{y}_i$: iterative process

Objective Function Optimization

- The prediction value at step t ($\hat{y}_i^{(t)}$)

$$\hat{y}_i^{(0)} = 0.5 \text{ (default initial guess)}$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

…

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Objective Function Optimization

- The prediction value at step t ($\hat{y}_i^{(t)}$)

$$\hat{y}_i^{(0)} = 0.5 \text{ (default initial guess)}$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

…

What tree do we want at each step?

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Objective Function Optimization

- The tree that optimizes our objective!

$$obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constants$$

We use the previous $\hat{y}_i$

- Let's rewrite this

Objective Function Optimization

- We take the second-order Taylor expansion of the loss function (for not-so-friendly loss functions)

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2$$

$$l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) \approx l\left(y_i, \hat{y}_i^{(t-1)}\right) +$$

$$\underbrace{\left[\frac{\partial}{\partial \hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)\right]}_{\textbf{gradient}} f_t(x_i) + \frac{1}{2}\underbrace{\left[\frac{\partial^2}{\partial\left(\hat{y}_i^{(t-1)}\right)^2} l\left(y_i, \hat{y}_i^{(t-1)}\right)\right]}_{\textbf{hessian}} f_t(x_i)^2$$

Objective Function Optimization

- We take the second-order Taylor expansion of the loss function (for not-so-friendly loss functions)

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2$$

$$l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) \approx l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2$$

Objective Function Optimization

- The objective is then

$$obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + C$$

$$= \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t) + C$$

Objective Function Optimization

- We can remove the constants

$$obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + C$$

$$= \sum_{i=1}^{n} \left[\cancel{l\left(y_i, \hat{y}_i^{(t-1)}\right)} + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2\right] + \Omega(f_t) + \cancel{C}$$

Objective Function Optimization

- We can remove the constants

$$obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + C$$

$$= \boxed{\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t)}$$

Our optimization goal

Specifics of the regularization term

- Redefine the definition of a tree f(x)

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T, q\colon \mathbb{R}^d \to \{1,2,\dots,T\}$$

w = vector of scores on leaves

q = function assigning data points to leaves

T = number of leaves

- This is a more explicit version of our function f(x), in terms of leaf weights

Specifics of the regularization term

- We then express the regularization term as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Specifics of the regularization term

- We then express the regularization term as

$$\Omega(f) = \boxed{\gamma} T + \frac{1}{2} \boxed{\lambda} \sum_{j=1}^{T} w_j^2$$

Hyperparameters

Objective Function Optimization

- Rewriting our objective function:

$$obj^{(t)} \approx \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Objective Function Optimization

- Rewriting our objective function:

$$obj^{(t)} \approx \boxed{\sum_{i=1}^{n}} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \boxed{\sum_{j=1}^{T}} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Switching sums from by-individual to by-leaf

Objective Function Optimization

- Rewriting our objective function:

$$obj^{(t)} \approx \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}\left[\left(\boxed{\sum_{i\in I_j} g_i}\right) w_j + \frac{1}{2}\left(\boxed{\sum_{i\in I_j} h_i} + \lambda\right) w_j^2\right] + \gamma T$$

Make notation more compact

Objective Function Optimization

- Rewriting our objective function:

$$obj^{(t)} \approx \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}\left[\boxed{G_j} w_j + \frac{1}{2}\left(\boxed{H_j} + \lambda\right) w_j^2\right] + \gamma T$$

Objective Function Optimization

- Rewriting our objective function:

$$obj^{(t)} \approx \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}\left[\boxed{G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2}\right] + \gamma T$$

This is quadratic!

Objective Function Optimization

- The optimum solution for the quadratic eq'n:

$$w_j^* = -\frac{G_j}{H_j + \lambda}\left(Recall: -\frac{b}{a}\right)$$

Objective Function Optimization

- The optimum solution for the quadratic eq'n:

$$w_j^* = -\frac{G_j}{H_j + \lambda}\left(Recall: -\frac{b}{a}\right)$$

- The best objective:

$$obj^* = \sum_{j=1}^{T}\left[G_j\left(-\frac{G_j}{H_j + \lambda}\right) + \frac{1}{2}(H_j + \lambda)\left(-\frac{G_j}{H_j + \lambda}\right)^2\right] + \gamma T$$

Objective Function Optimization

- The optimum solution for the quadratic eq'n:

$$w_j^* = -\frac{G_j}{H_j + \lambda}\left(Recall: -\frac{b}{a}\right)$$

- The best objective:

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Objective Function Optimization

- The optimum solution for the quadratic eq'n:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \left(Recall: -\frac{b}{a} \right)$$

- The best objective:

$$\boxed{obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T}$$

This can measure how good a tree structure is

Building Trees

- We now have a measure for how good a tree is.

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

- Ideal: enumerate all possible trees and pick the best one
- Practical: optimize one level of the tree at a time
  - I.e. find the best split at each level of the tree
  - We now have a way to find the optimum tree!

Building Trees

- We now have a measure for how good a tree is.

$$obj^* = \boxed{-}\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

Removed in implementation **Minimize -> Maximize**

- Ideal: enumerate all possible trees and pick the best one
- Practical: <u>optimize one level of the tree at a time</u>
  - I.e. find the best split at each level of the tree
  - We now have a way to find the optimum tree!

Splitting Nodes – optimizing one level at a time

- We try to split a node into two leaves; the gain is:

$$Gain = \frac{1}{2}\left[\underbrace{\boxed{\frac{G_L^2}{H_L+\lambda}}}_{\text{Left score}} + \underbrace{\boxed{\frac{G_R^2}{H_R+\lambda}}}_{\text{Right score}} - \underbrace{\boxed{\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}}}_{\text{Original/ Unsplit}}\right] - \underbrace{\boxed{\gamma}}_{\text{Pruning}}$$

- If the **gain is negative**, we prune

Splitting Nodes – optimizing one level at a time

- We try to split a node into two leaves; the gain is:

$$Gain = \boxed{\frac{1}{2}}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Removed in implementation to run faster.
eXtreme!

- If the **gain is negative**, we prune
- We **want gain to be high** = large separation

Simple Classification Example

- Leaf weights:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

- Loss function:

$$y_i \ln\left(1 + e^{-\hat{y}_i}\right) + (1 - y_i) \ln\left(1 + e^{\hat{y}_i}\right)$$

$$g = \boxed{-(y_i - \hat{y}_i)}$$ **(-) Residuals**

$$h = \hat{y}_i \times (1 - \hat{y}_i)$$

- We sequentially fit on residuals because of the gradient!

Simple Classification Example

- We try to split a node into two leaves; the gain is:

$$Gain = \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

- Score:

$$\frac{G^2}{H + \lambda} = \frac{(\sum g)^2}{\sum h + \lambda} = \frac{\left(\sum(residuals)\right)^2}{\sum \hat{y}_i \times (1 - \hat{y}_i) + \lambda}$$

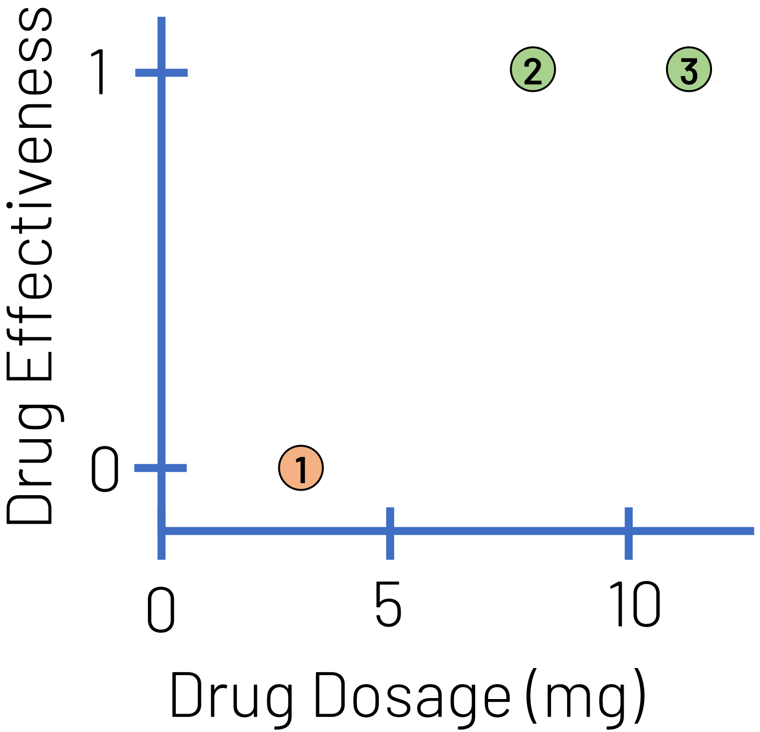
Simple Classification Example

- Data:

Index	Drug Dosage (mg)	Drug is Effective
1	3	0
2	8	1
3	11	1

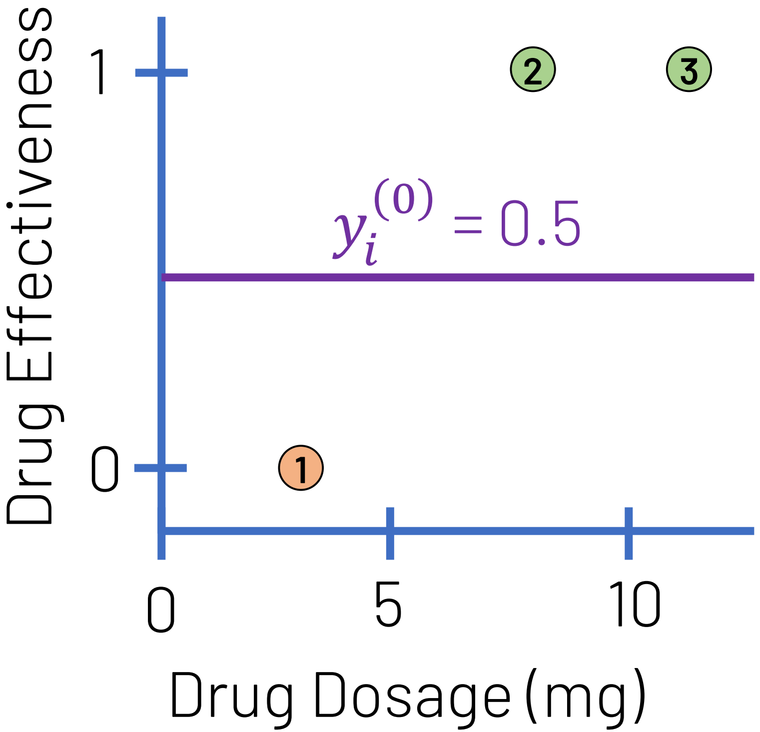
Simple Classification Example

- Data (graphical):



Simple Classification Example

- Base score (default initial guess) is 0.5

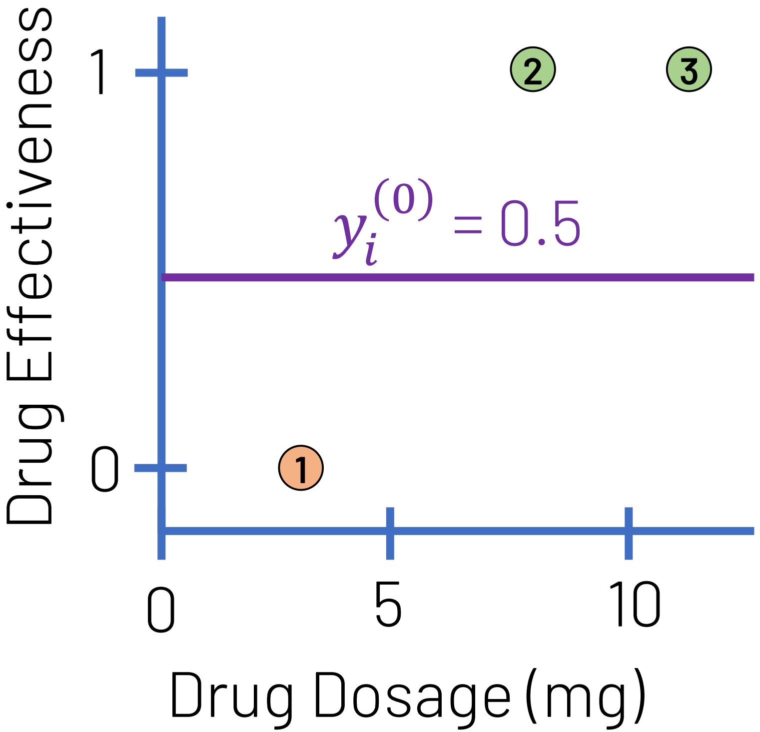


Simple Classification Example

- Build w/ initial guess

Initial guess

$y_i^{(0)} = 0.5$

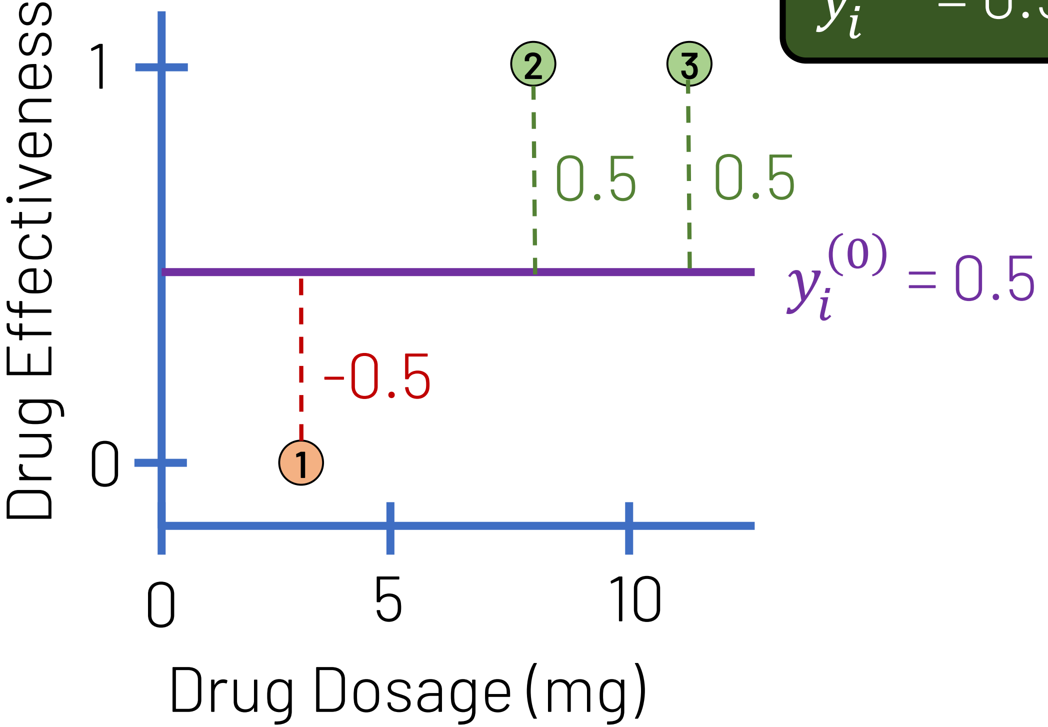


Simple Classification Example

- Let's see the residuals

Initial guess

$$y_i^{(0)} = 0.5$$

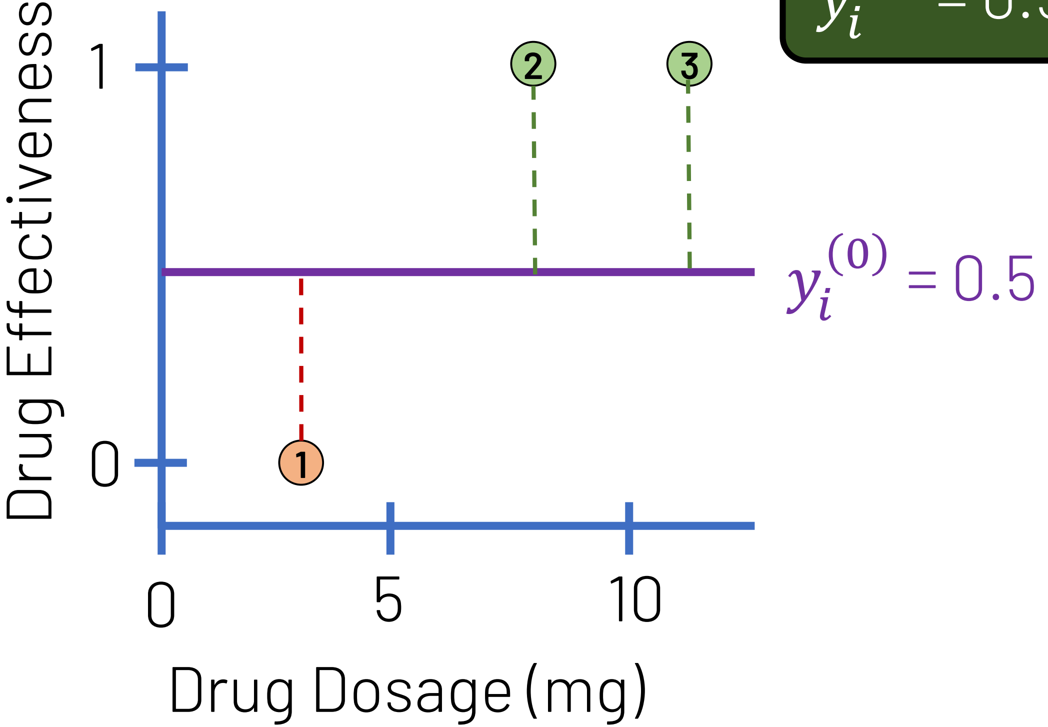


Simple Classification Example

- Let's see the residuals

Initial guess

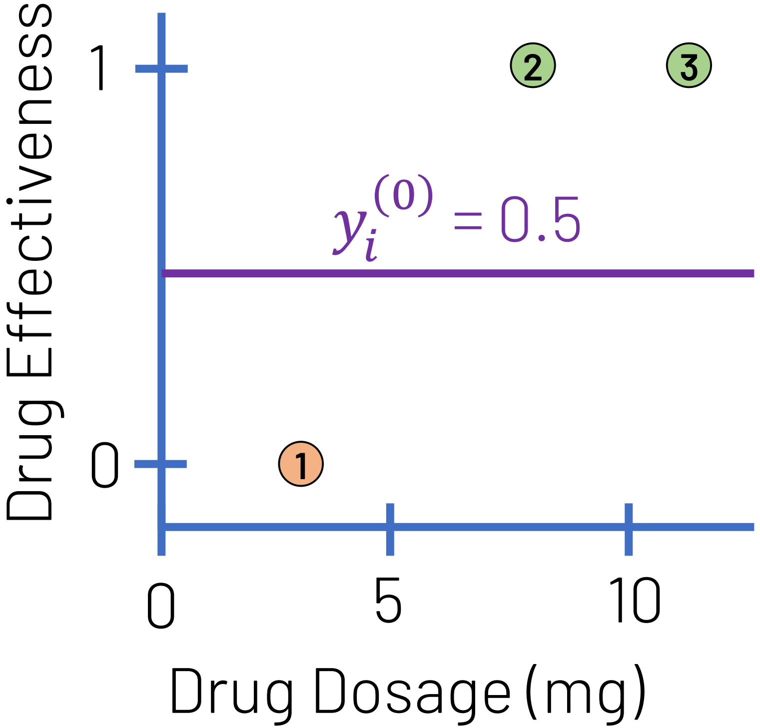
$$y_i^{(0)} = 0.5$$



1	2	3
-0.5	0.5	0.5

Simple Classification Example

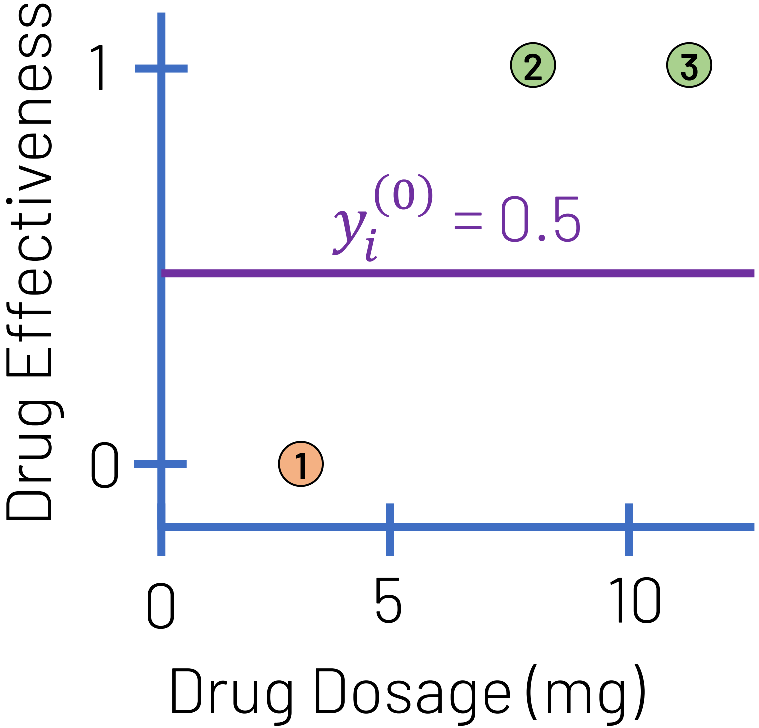
- Let's make a new tree with these residuals



1	2	3
-0.5	0.5	0.5

Simple Classification Example

- Calculate the score of the unsplit node

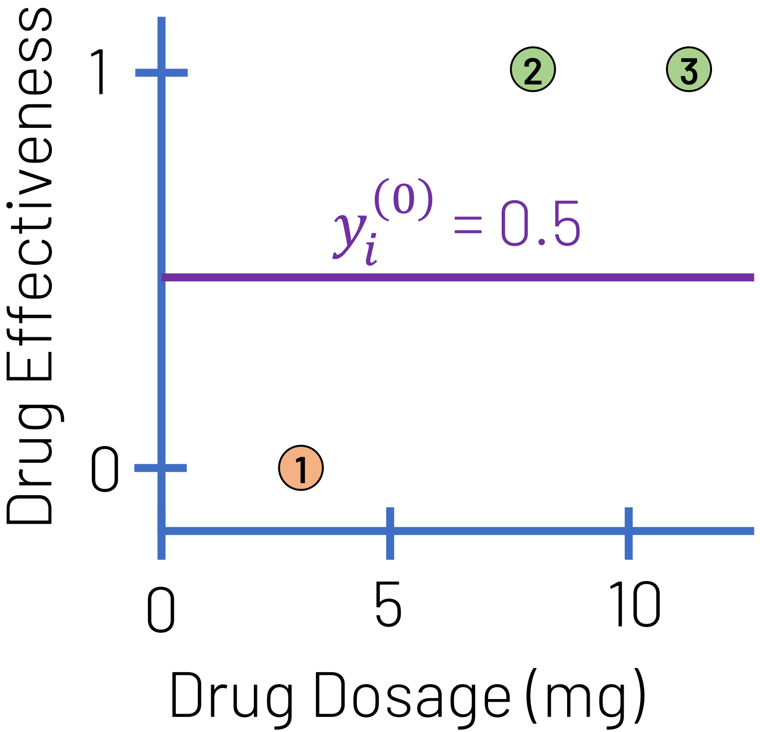


1	2	3
-0.5	0.5	0.5

$$\frac{\left(\sum(residuals)\right)^2}{\sum \hat{y}_i \times (1 - \hat{y}_i) + \boxed{\lambda}}$$ **Ignore**

Simple Classification Example

- Calculate the score of the unsplit node

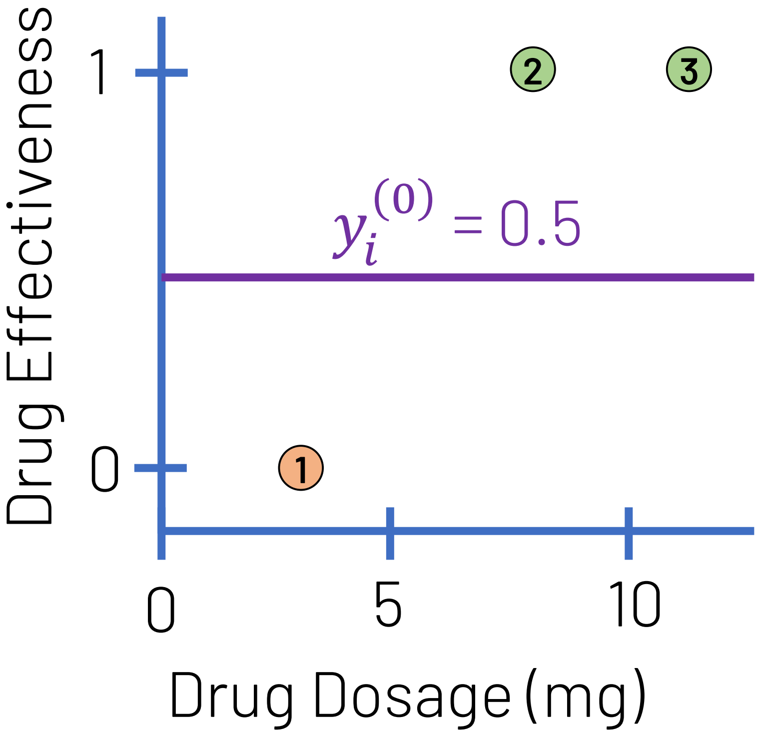


1	2	3
-0.5	0.5	0.5

$$\frac{\left(\sum(residuals)\right)^2}{\sum \hat{y}_i \times (1 - \hat{y}_i)}$$

Simple Classification Example

- Calculate the score of the unsplit node

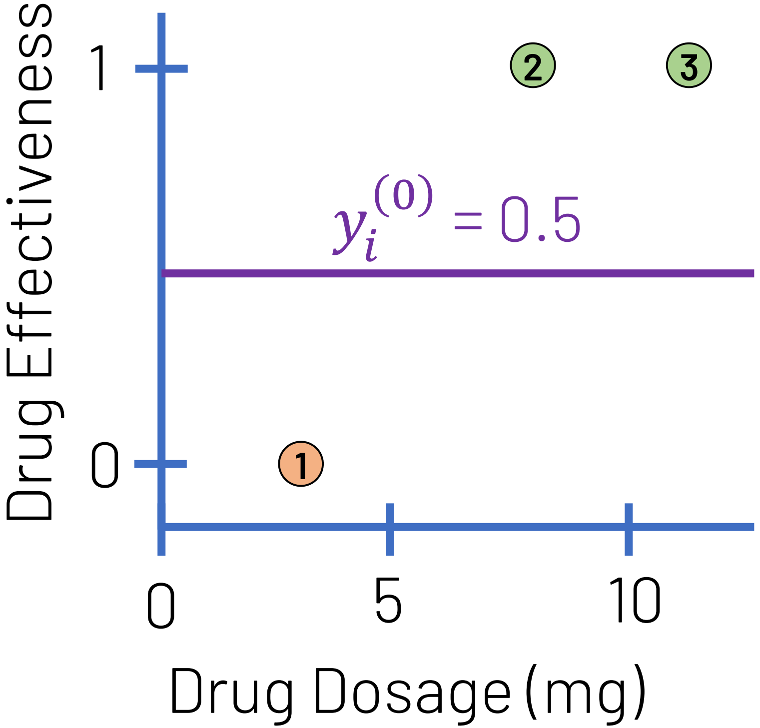


1	2	3
-0.5	0.5	0.5

$$\frac{(-0.5 + 0.5 + 0.5)^2}{(0.5)(1 - 0.5) \times 3}$$

Simple Classification Example

- Calculate the score of the unsplit node

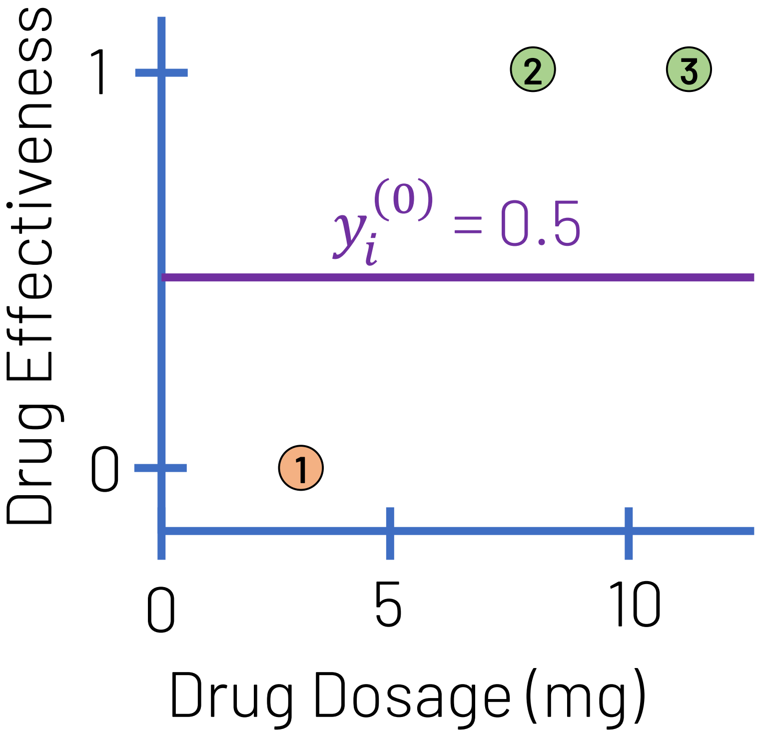


1	2	3
-0.5	0.5	0.5

$$\frac{(0.5)^2}{0.75} = \frac{1}{3}$$

Simple Classification Example

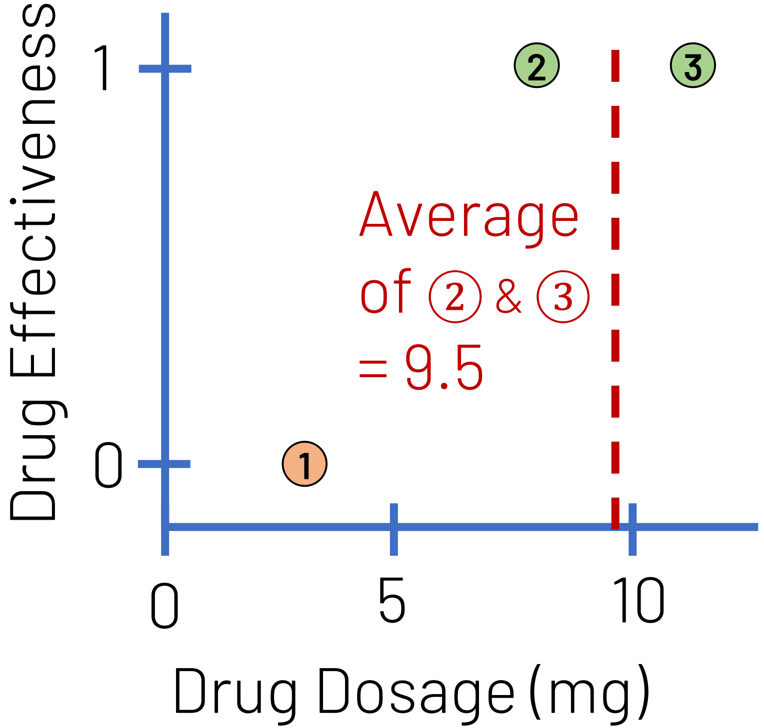
- Calculate the score of the unsplit node



$$\frac{1}{3}\left(-0.5 \quad 0.5 \quad 0.5 \right)$$

Simple Classification Example

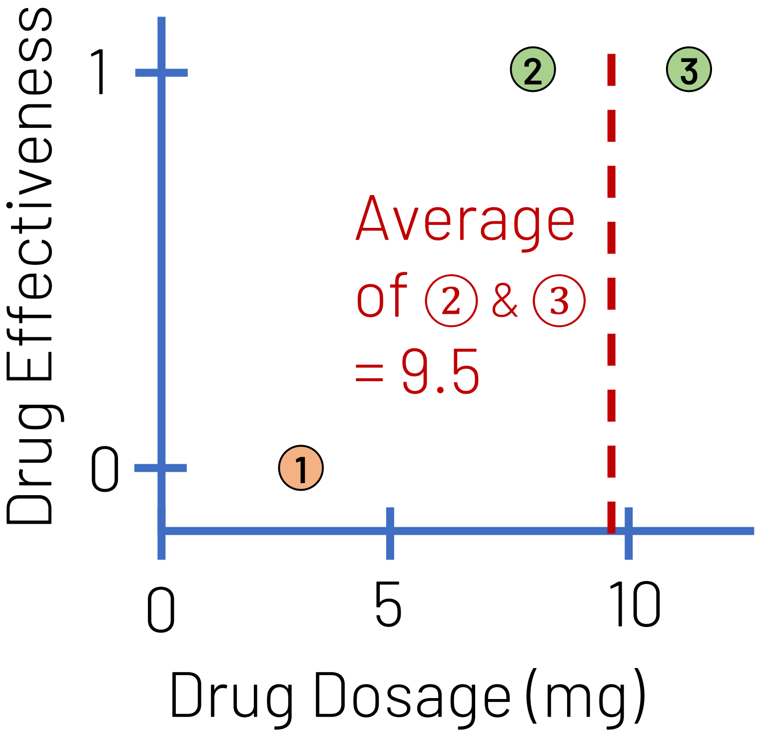
- Use the split between ② & ③

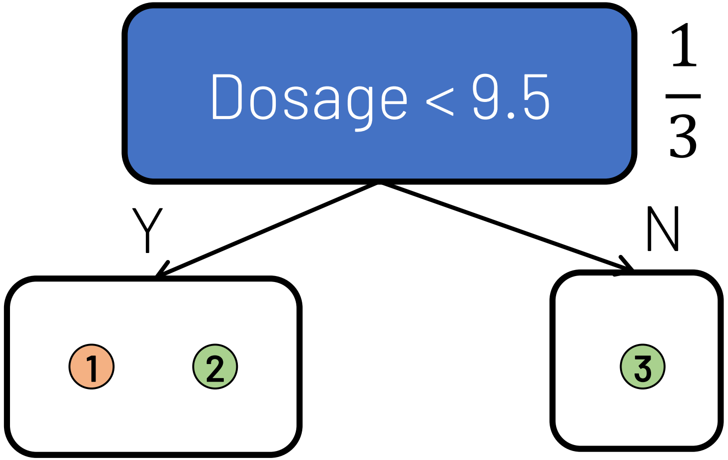


Dosage < 9.5 $\frac{1}{3}$

Simple Classification Example

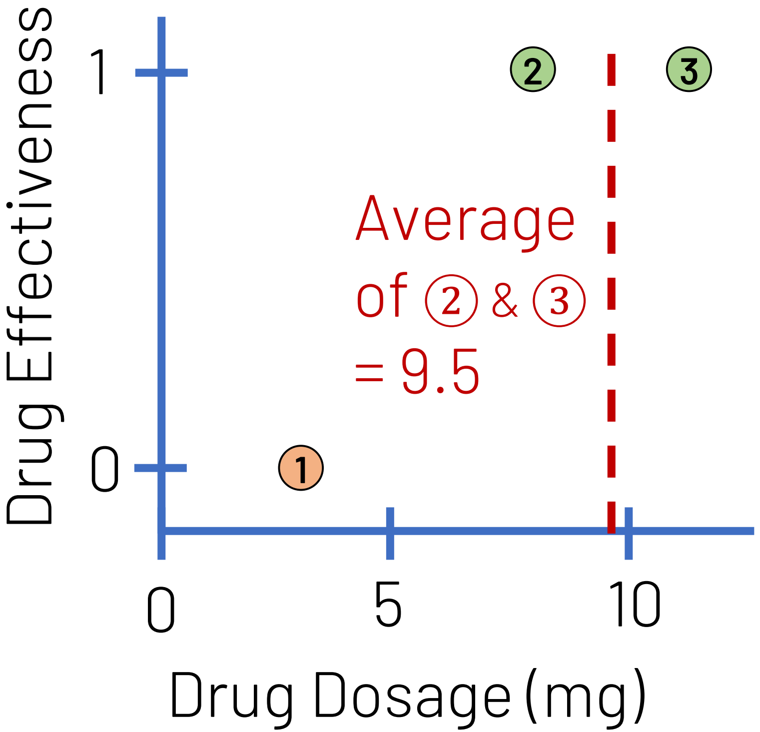
- Use the split between ② & ③

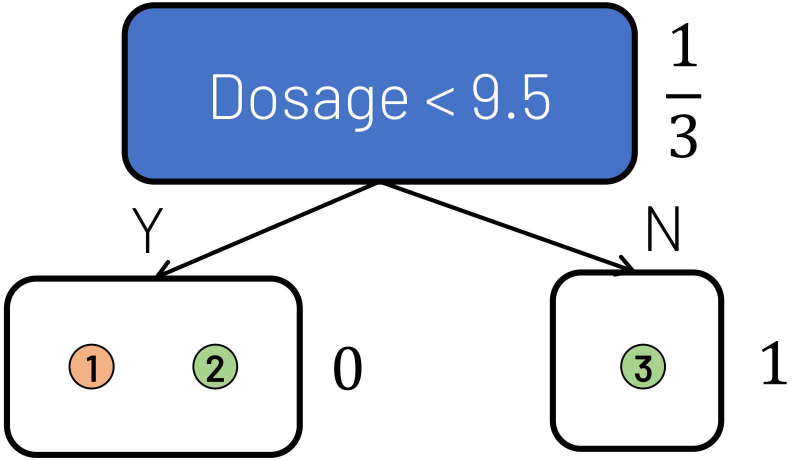




Simple Classification Example

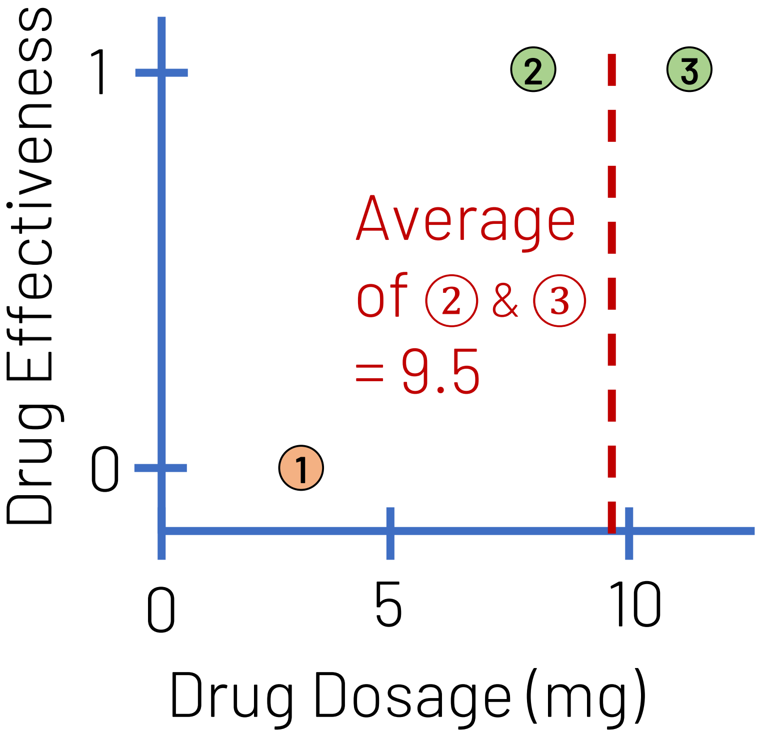
- Calculate the scores of the L & R leaves

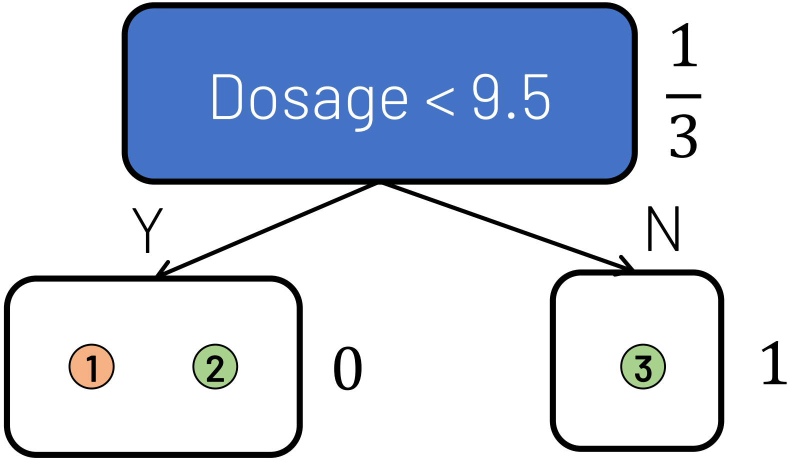




Simple Classification Example

- Calculate the Gain

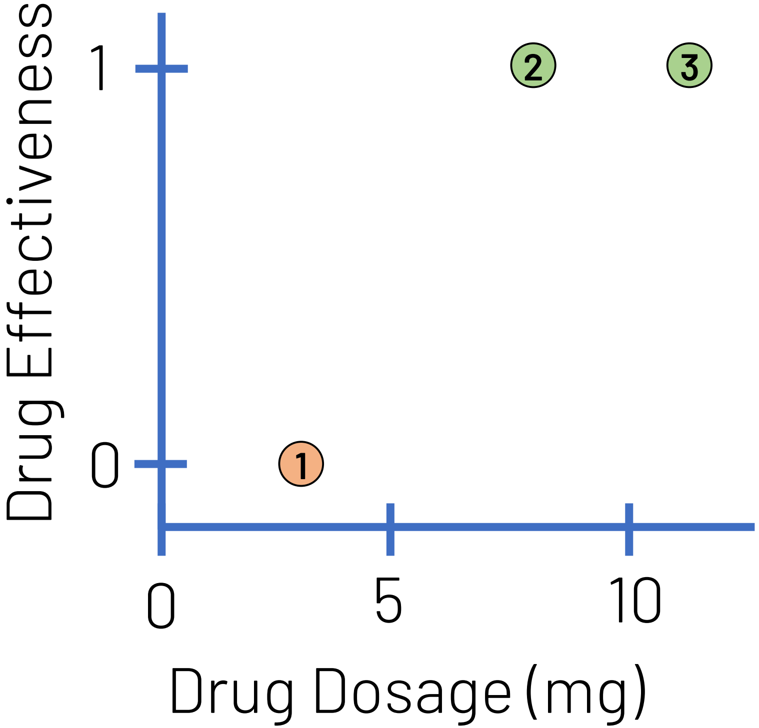




$$Gain = L + R - Orig$$
$$= 0 + 1 - \frac{1}{3} = 0.67$$

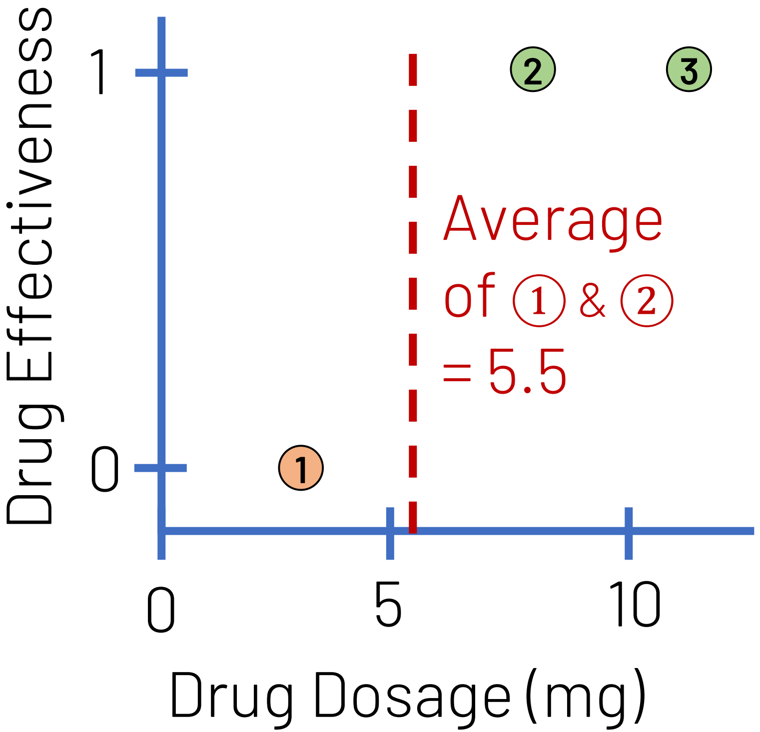
Simple Classification Example

- Is there a better split?



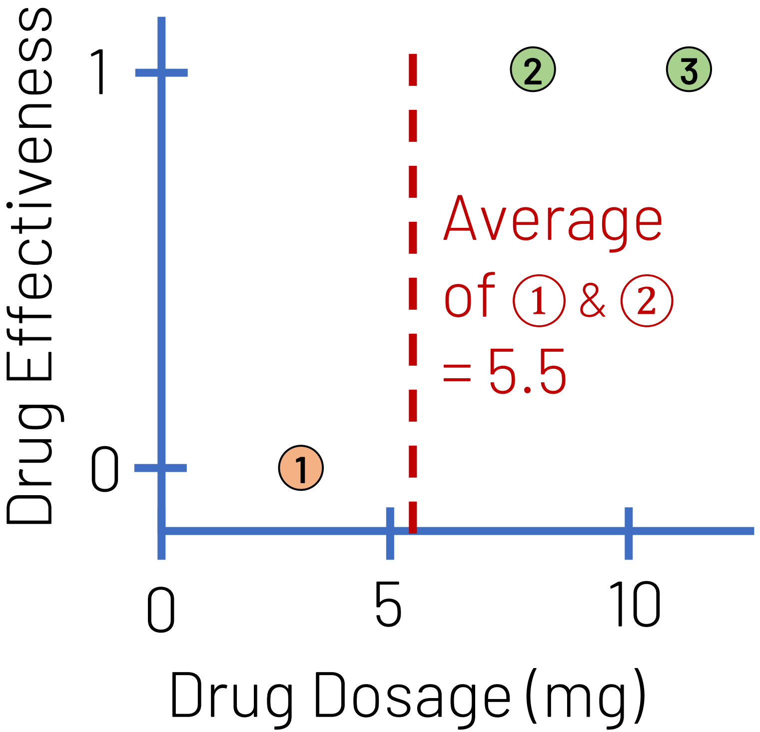
Simple Classification Example

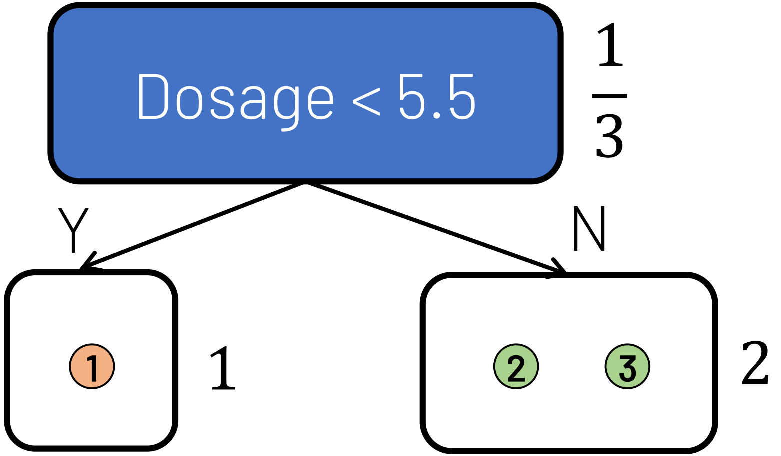
- Is there a better split?



Simple Classification Example

- Is there a better split?

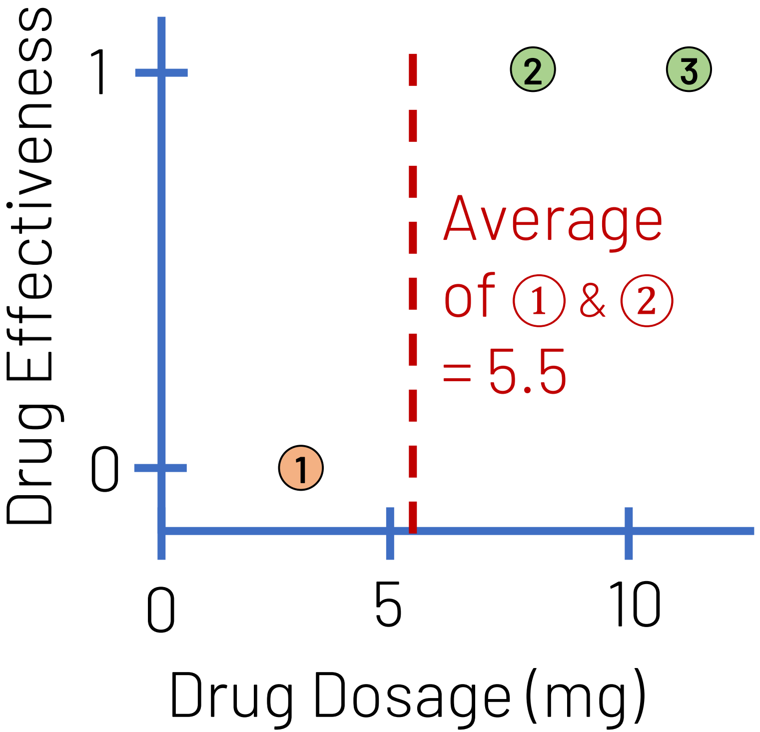


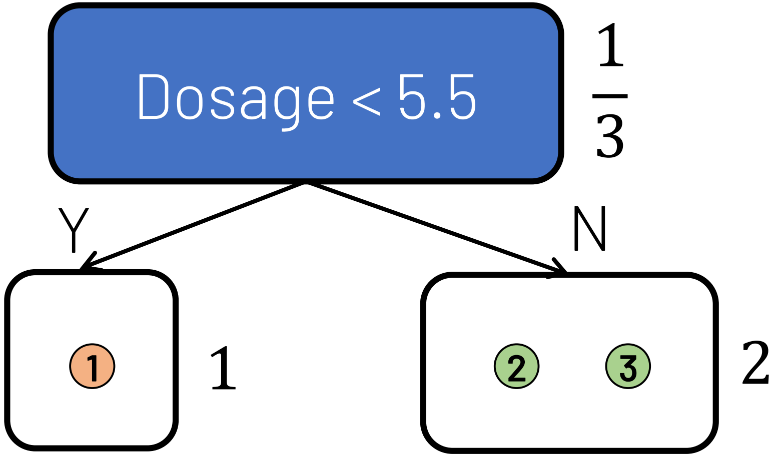


$$Gain = L + R - Orig$$
$$= 1 + 2 - \frac{1}{3} = 2.67$$

Simple Classification Example

- Is there a better split?

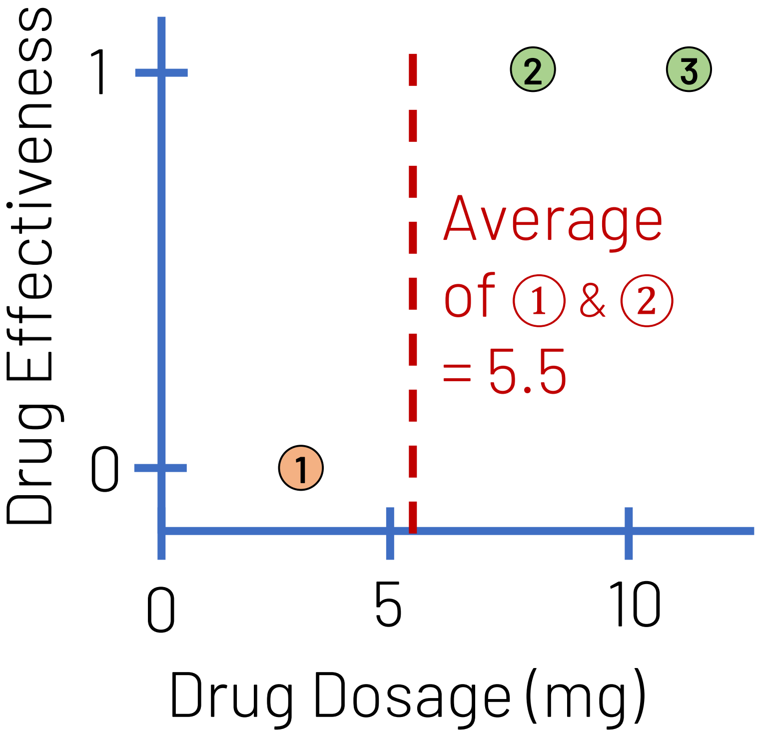


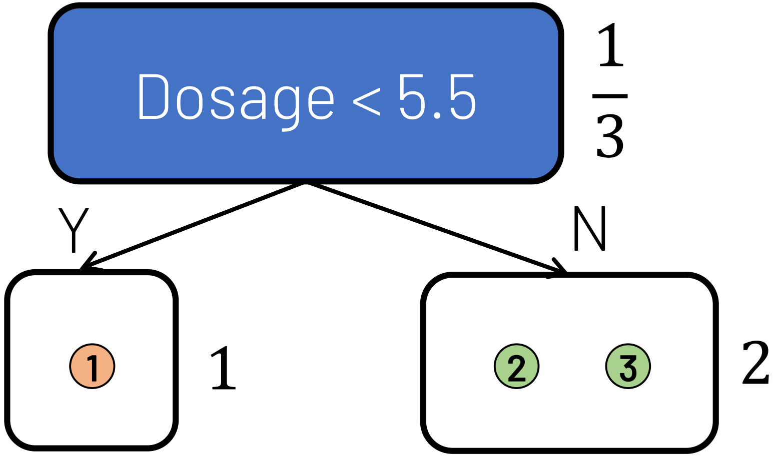


$$Gain = L + R - Orig$$
$$= 1 + 2 - \frac{1}{3} = \boxed{2.67}$$

Simple Classification Example

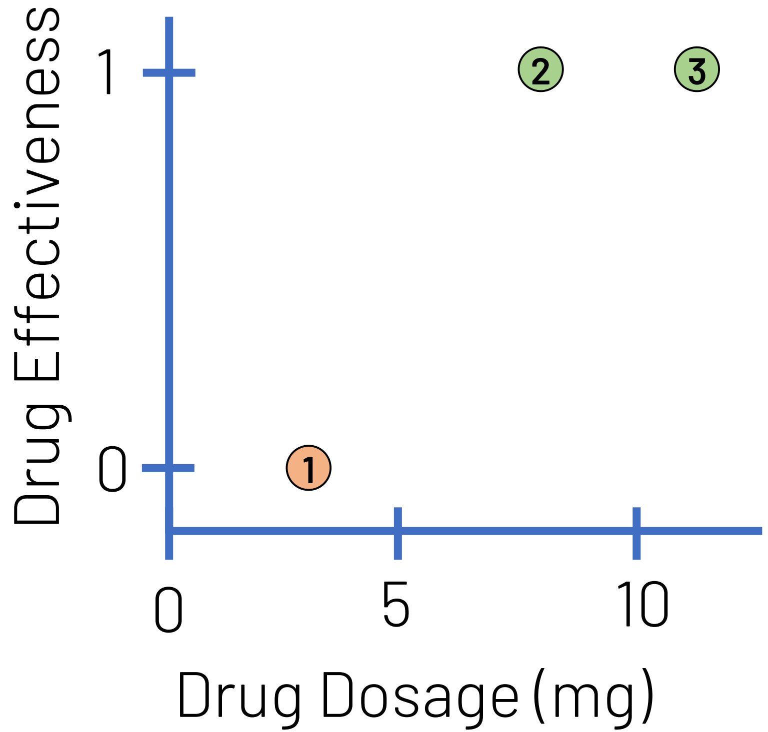
- For now, let's stop splitting





Simple Classification Example

- We now have a tree

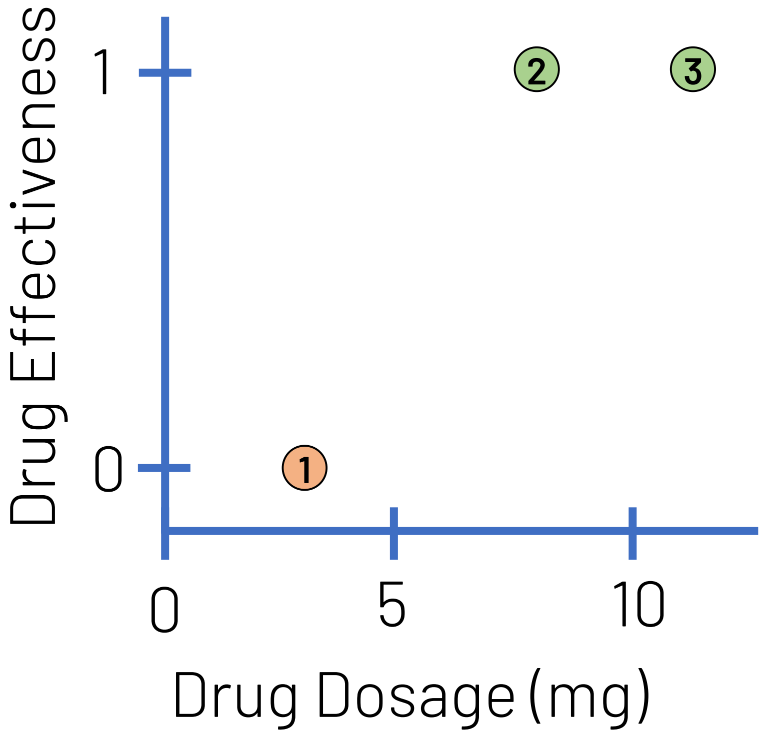


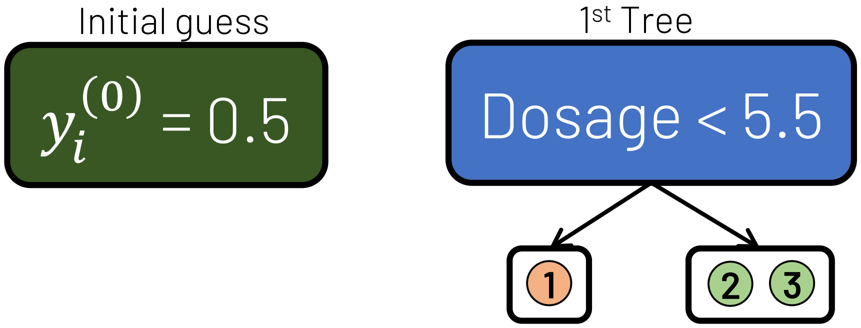
Initial guess

$$y_i^{(0)} = 0.5$$

Simple Classification Example

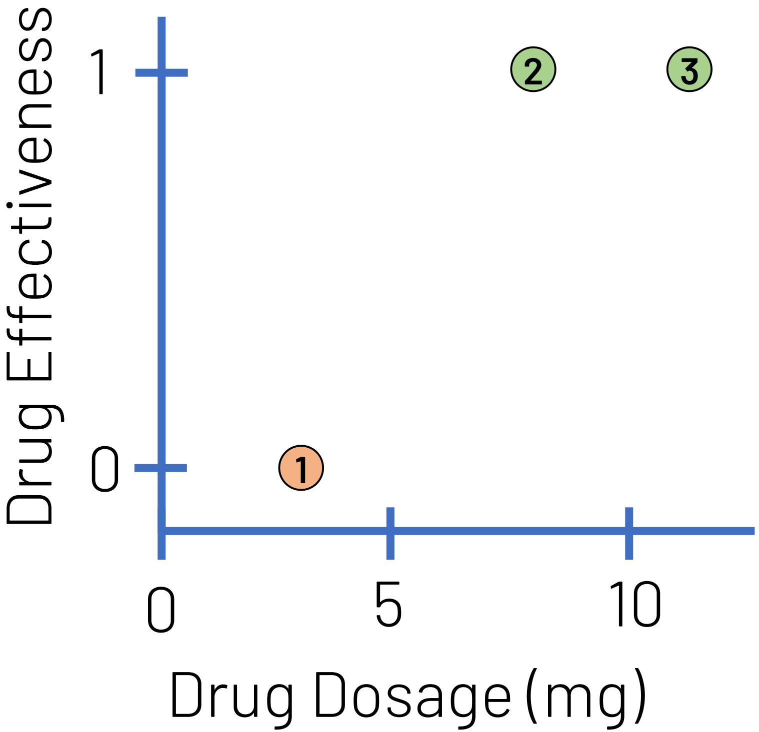
- We now have a tree





Simple Classification Example

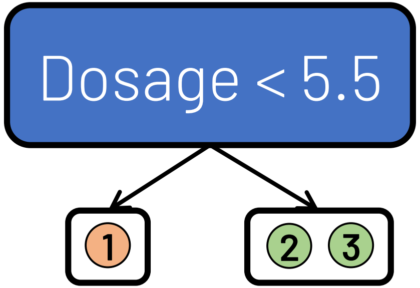
- We calc. leaf weights



Initial guess

$y_i^{(0)} = 0.5$

1st Tree



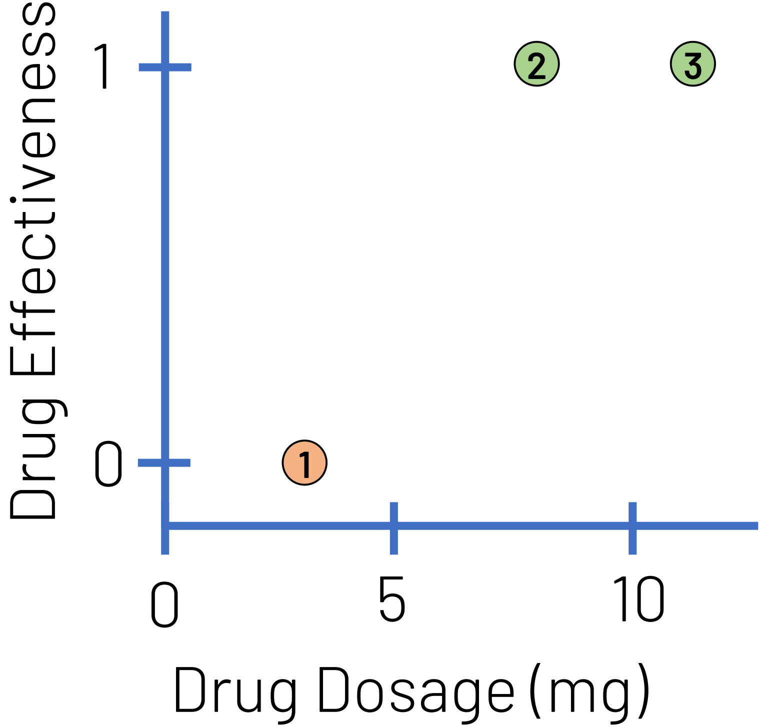
Recall: equation for leaf weights

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$= \frac{\sum(residuals)}{\sum \hat{y}_i \times (1 - \hat{y}_i)}$$

Simple Classification Example

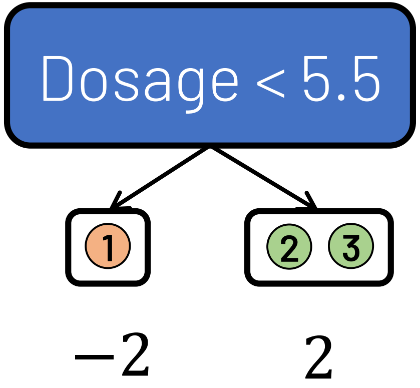
- We calc. leaf weights



Initial guess

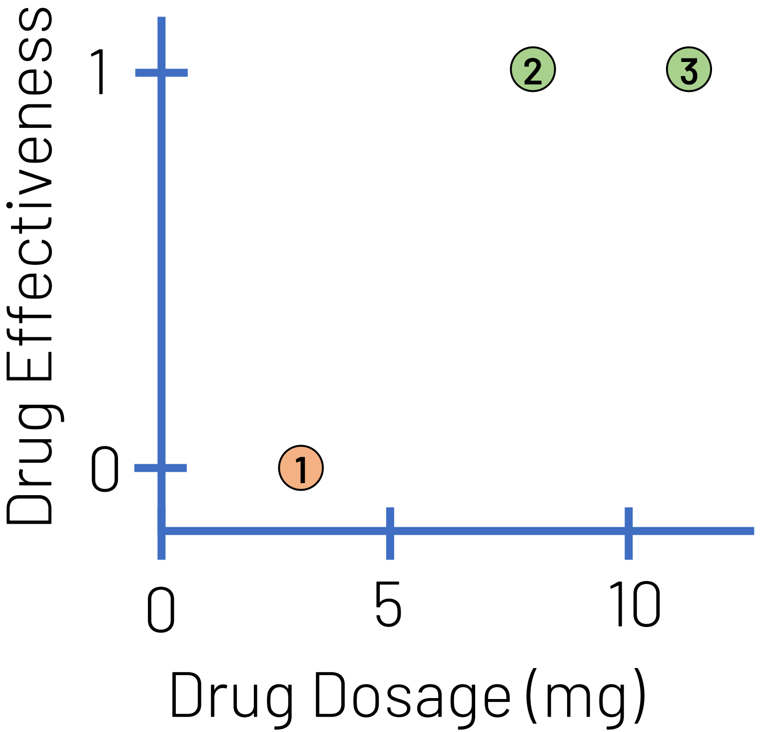
$y_i^{(0)} = 0.5$

1st Tree



Simple Classification Example

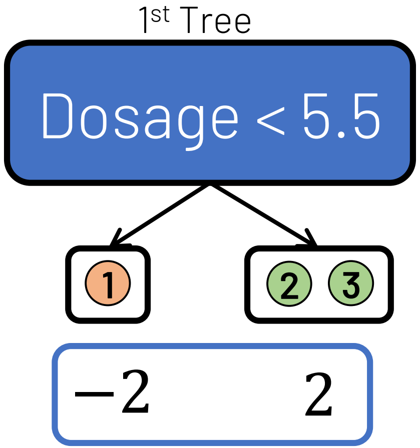
- We make new preds.



Initial guess

$$y_i^{(0)} = 0.5$$

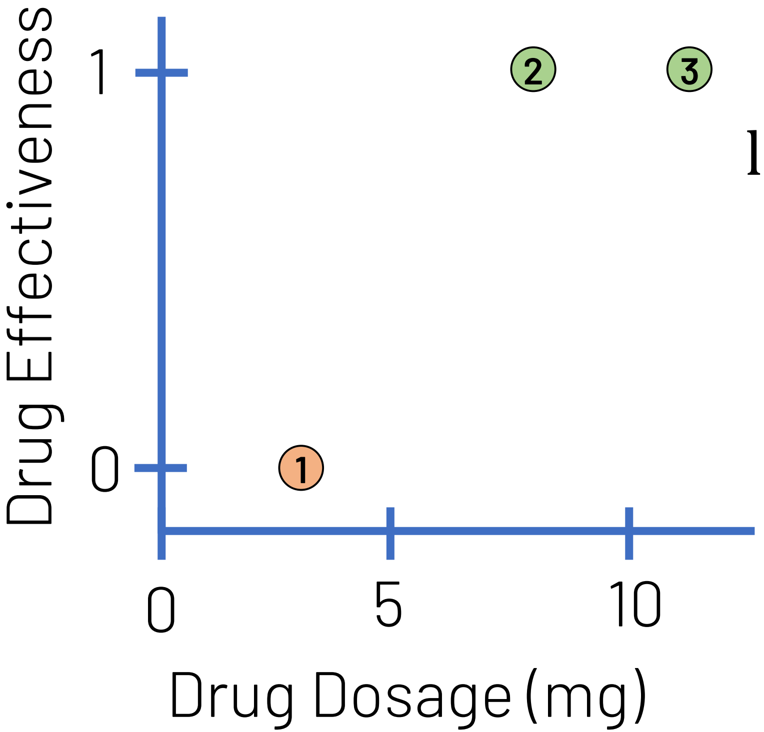
This is in probability



These are in log(odds)

Simple Classification Example

- We make new preds.

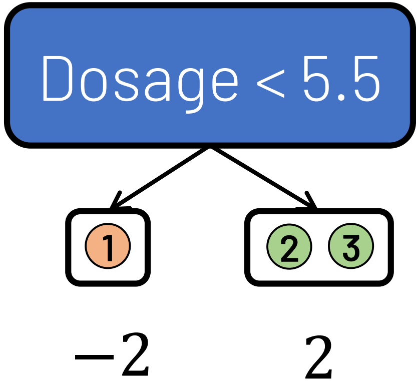


Initial guess

$$y_i^{(0)} = 0.5$$

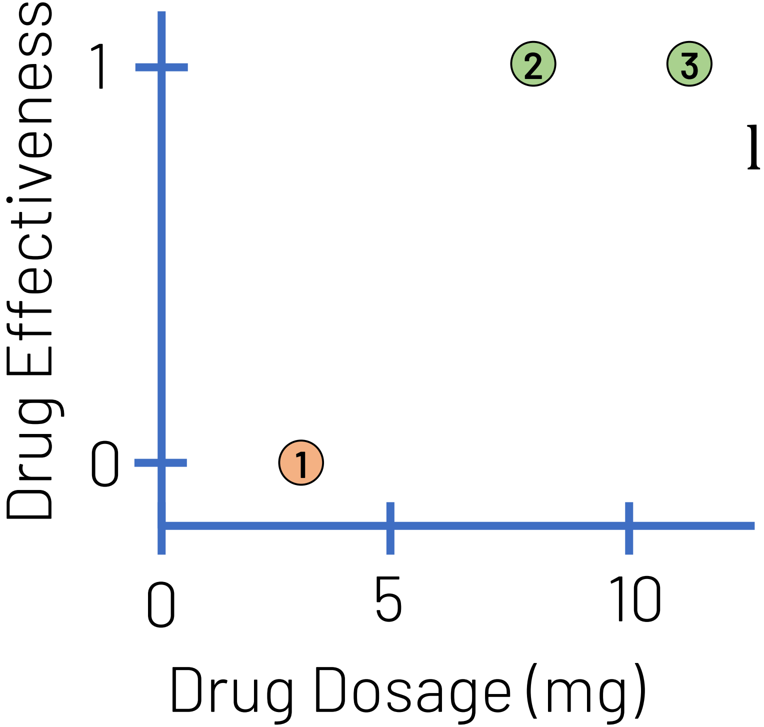
$$\log(odds) = 0$$

1st Tree



Simple Classification Example

- We make new preds.

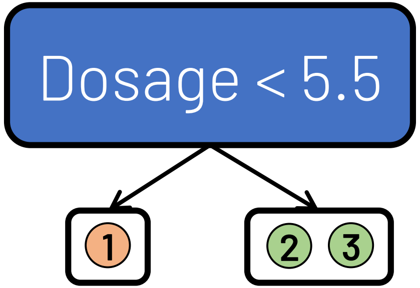


Initial guess

$$y_i^{(0)} = 0.5$$

$$\log(odds) = 0$$

1st Tree



$$-2 \qquad 2$$

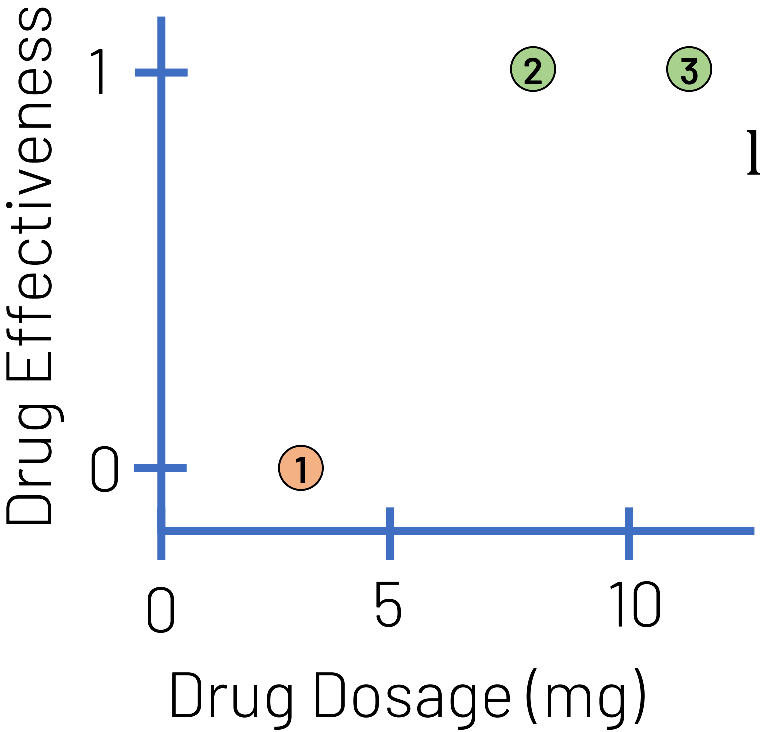
$$\log(odds)\ \ of\ \hat{y}_1^{(1)} = 0 + \eta(-2)$$

$$\log(odds)\ \ of\ \hat{y}_2^{(1)} = 0 + \eta(2)$$

$$\log(odds)\ \ of\ \hat{y}_3^{(1)} = 0 + \eta(2)$$

Simple Classification Example

- We make new preds.

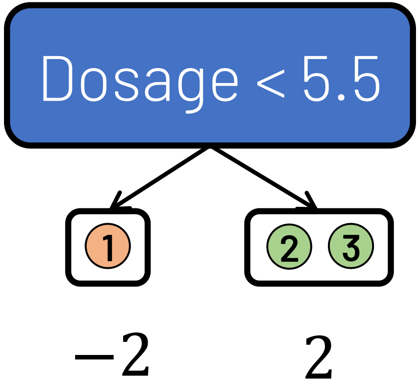


Initial guess

$$y_i^{(0)} = 0.5$$

$$\log(odds) = 0$$

1st Tree



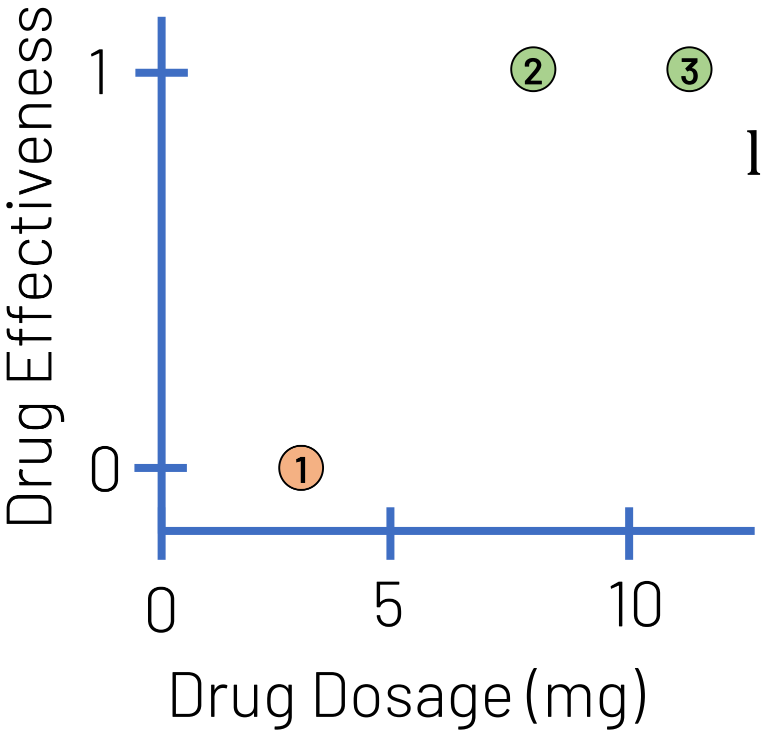
$$\log(odds)\ \ of\ \hat{y}_1^{(1)} = 0 + 0.3(-2)$$

$$\log(odds)\ \ of\ \hat{y}_2^{(1)} = 0 + 0.3(2)$$

$$\log(odds)\ \ of\ \hat{y}_3^{(1)} = 0 + 0.3(2)$$

Simple Classification Example

- We make new preds.

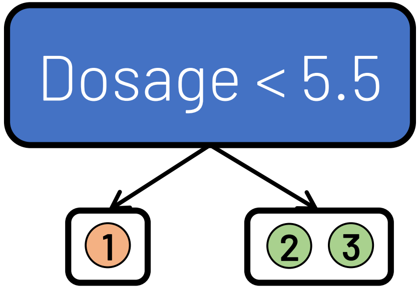


Initial guess

$$y_i^{(0)} = 0.5$$

$$\log(odds) = 0$$

1st Tree



$$-2 \qquad 2$$

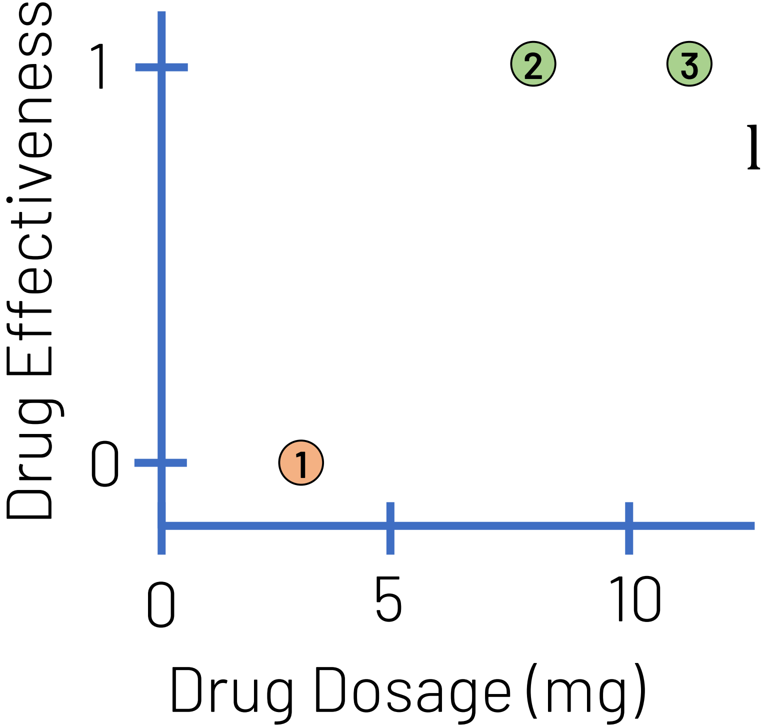
$$\log(odds)\ \ of\ \hat{y}_1^{(1)} = -0.6$$

$$\log(odds)\ \ of\ \hat{y}_2^{(1)} = 0.6$$

$$\log(odds)\ \ of\ \hat{y}_3^{(1)} = 0.6$$

Simple Classification Example

- We make new preds.

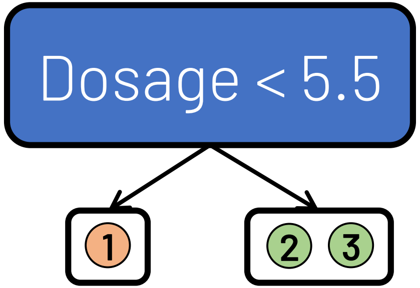


Initial guess

$y_i^{(0)} = 0.5$

$\log(odds) = 0$

1st Tree



-2 $\quad$ 2

Convert back to probability

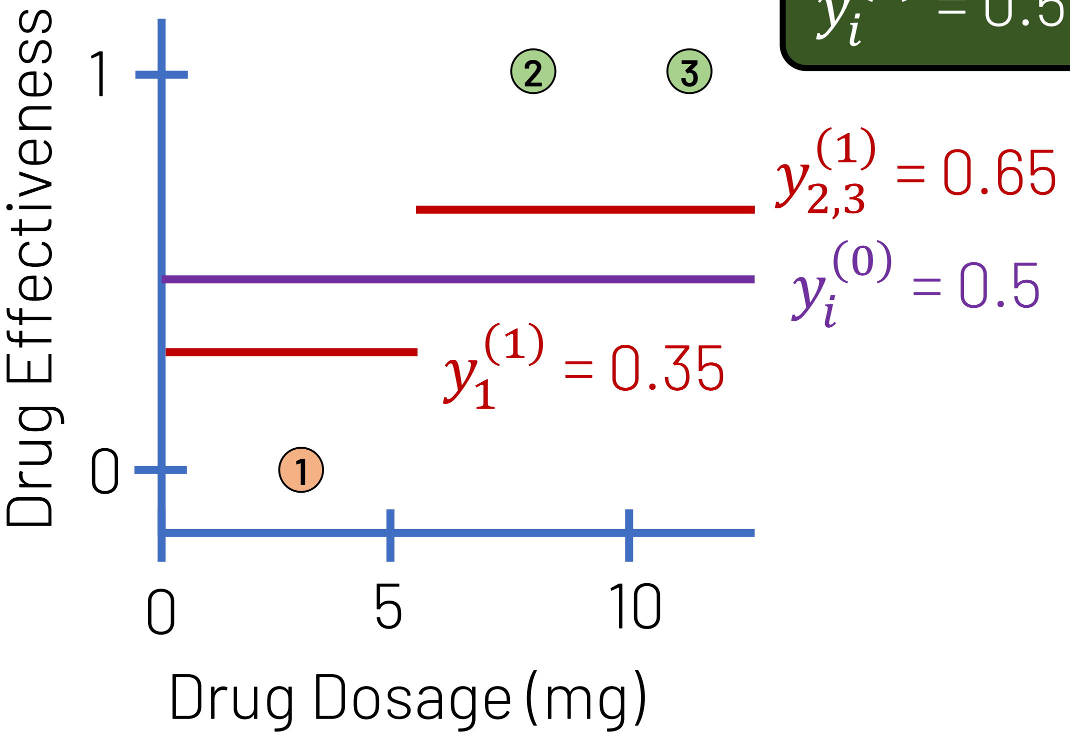
$\hat{y}_1^{(1)} = 0.35$

$\hat{y}_2^{(1)} = 0.65$

$\hat{y}_3^{(1)} = 0.65$

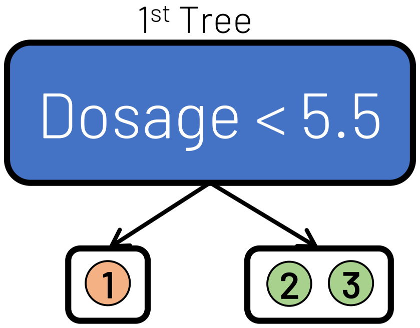
Simple Classification Example

- We make new preds.



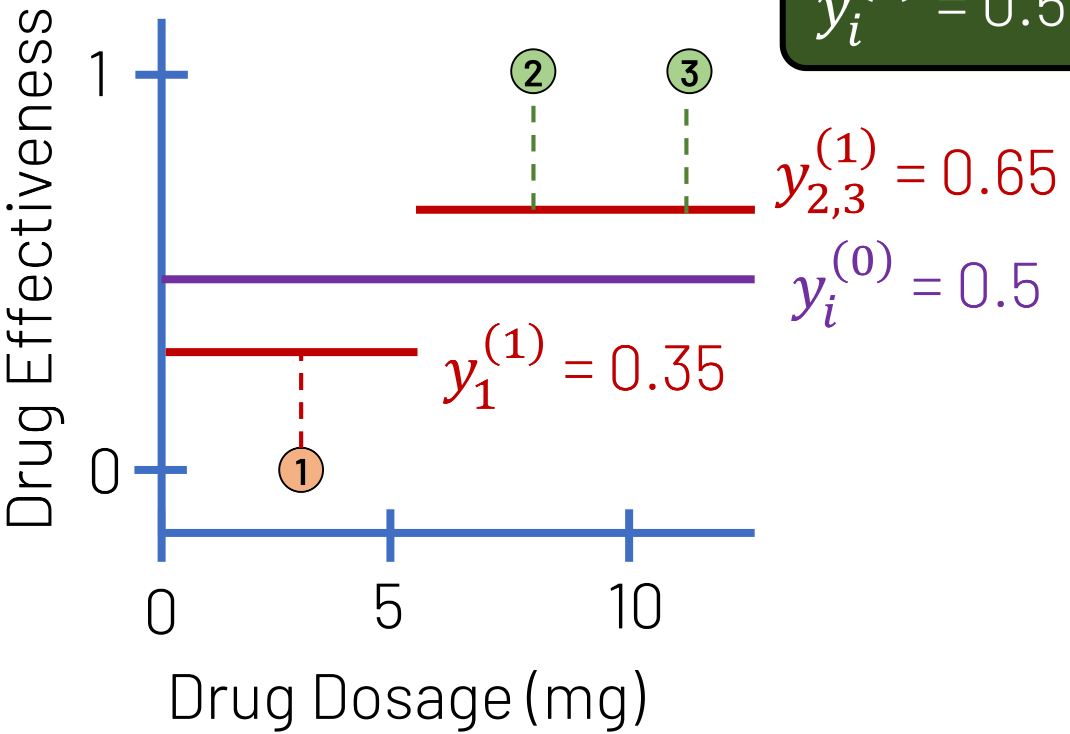
Initial guess

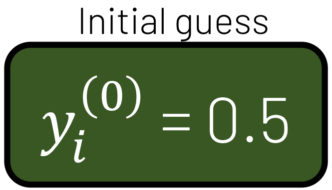
$y_i^{(0)} = 0.5$

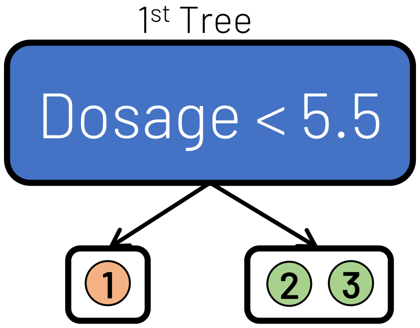


Simple Classification Example

- We have new residuals

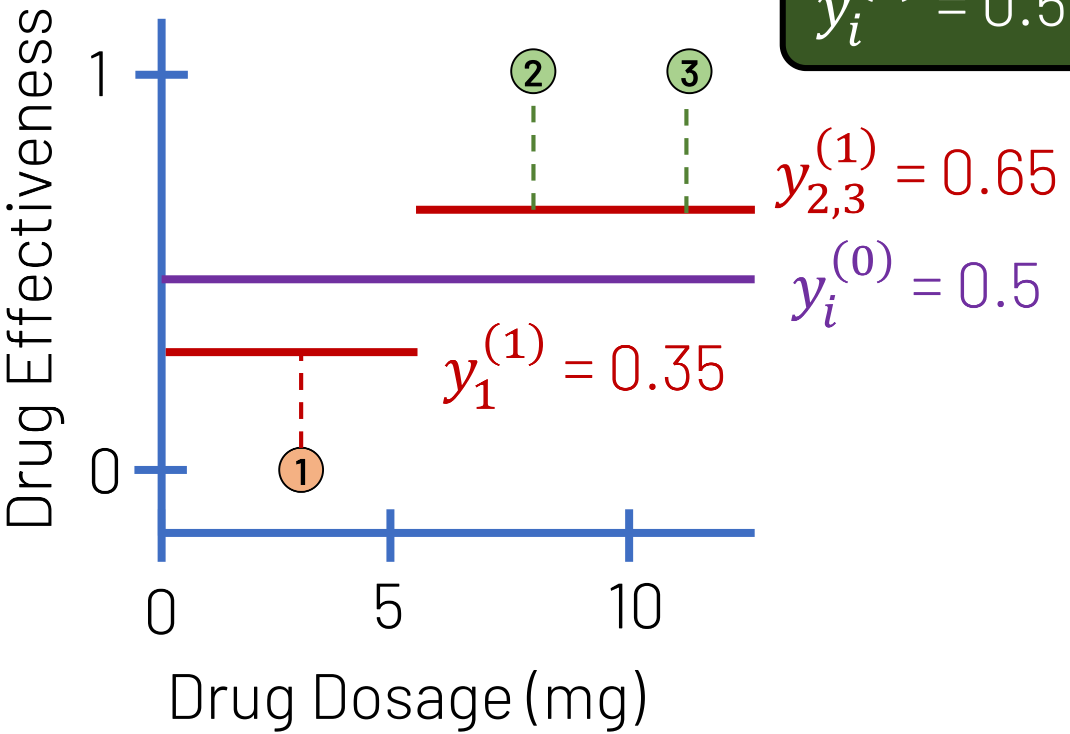


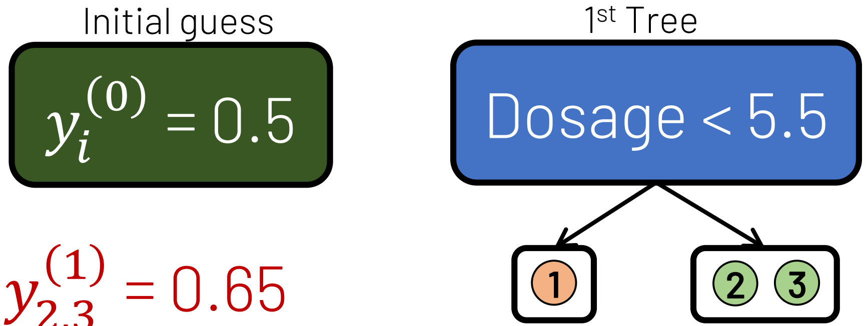




Simple Classification Example

- We have new residuals

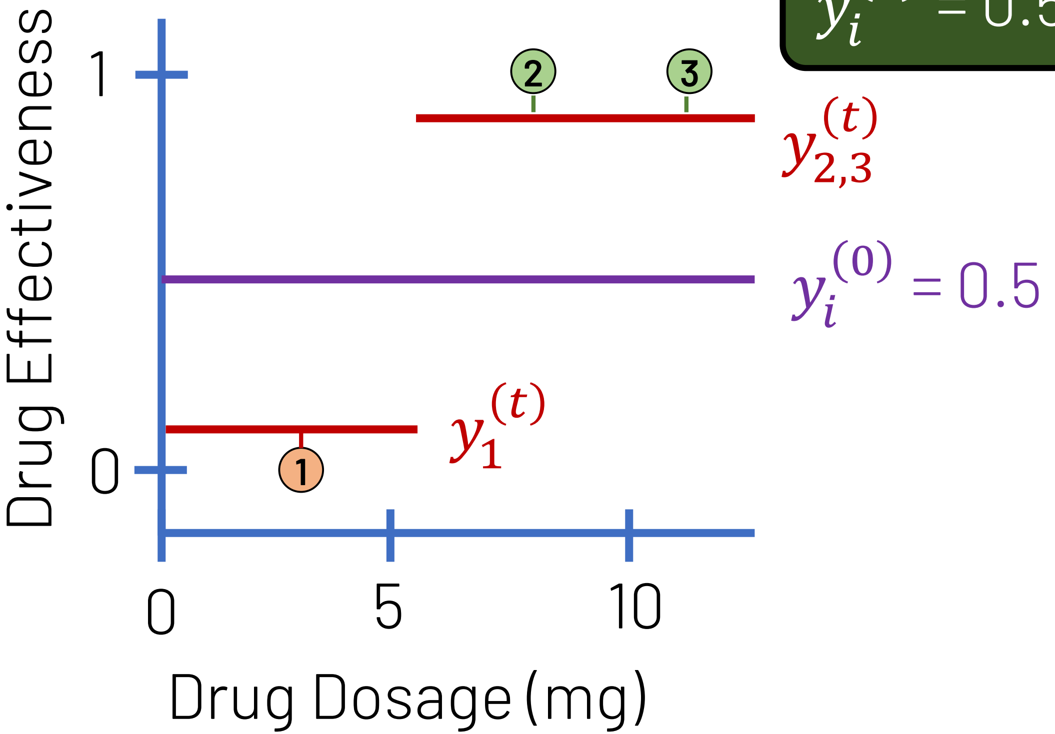


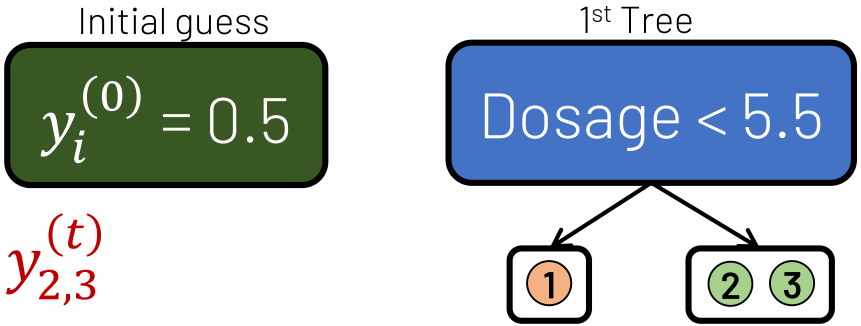


Repeat the process on these new residuals!

Simple Classification Example

- After t trees:





Repeat the process on these new residuals!

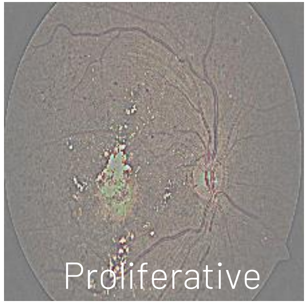
3. XGBoost in Action

Staging of Diabetic Retinopathy

Diabetic Retinopathy Data

- Diabetic Retinopathy – retinal disease caused by diabetes
- 3,662 retinal images labeled as:

0	No DR
1	Mild
2	Moderate
3	Severe
4	Proliferative



- Images have been pre-processed

Methodology

1. Perform PCA
   a. Reduce ~150,000 cols to 2,500 cols (97% var exp.)
2. Train on 80% of the data
3. Test on the remaining 20% and analyze the results

Code Snippets - XGBoost

```
from xgboost import XGBClassifier, DMatrix
import xgboost as xgb
```

Import

Split the data into train and test sets

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

Train-test split

Code Snippets - XGBoost

Perform PCA on the training set

```
In [43]: pca = PCA(n_components = 2500)
```

```
In [44]: %%time
PC_train = pca.fit_transform(X_train)
```

PCA on train

```
Wall time: 1min 47s
```

```
In [45]: pca.explained_variance_ratio_.sum()

Out[45]: 0.9725809679667871
```

Use the same PCA transformations on the test set

```
In [46]: %%time
PC_test = pca.transform(X_test)
```

Apply same transform on test

```
Wall time: 4.78 s
```

XGBoost Learning API

XGB Learning API, Default Parameters

Convert the np arrays into xgb D-Matrices

Special matrix format

In [47]:
```
D_train = DMatrix(PC_train, label = y_train)
```

In [48]:
```
D_test = DMatrix(PC_test, label = y_test)
```

Train with default parameters

Learning API

In [164]:
```
%%time
model_default = xgb.train({'num_class':5,
                           'objective':'multi:softprob',
                           'random_state':2020},
                          D_train,
                          steps)
```

Return probs per class

estimators = 100

```
Wall time: 1min 12s
```

XGBoost SKLearn API

```
In [222]: xgb_clf = XGBClassifier(objective = 'multi:softprob',
                                  num_class = 5,
                                  random_state = 2020)
```

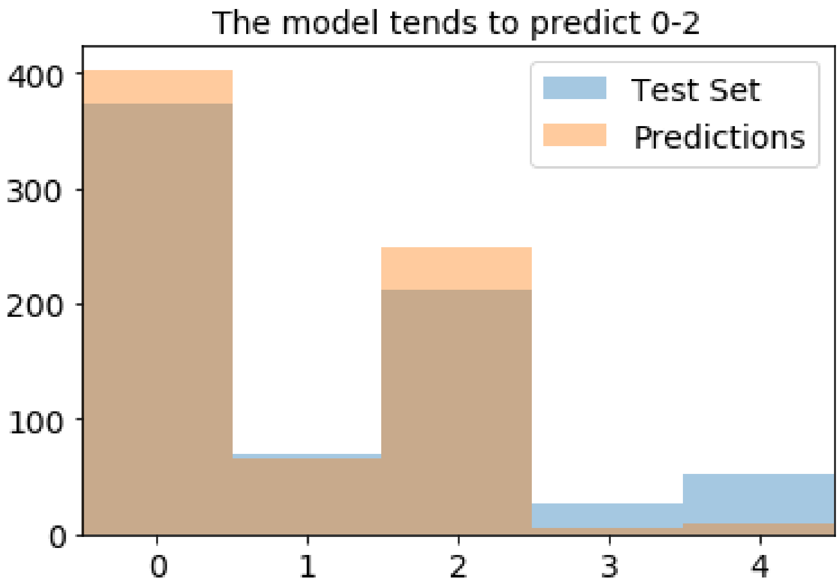
sklearn API

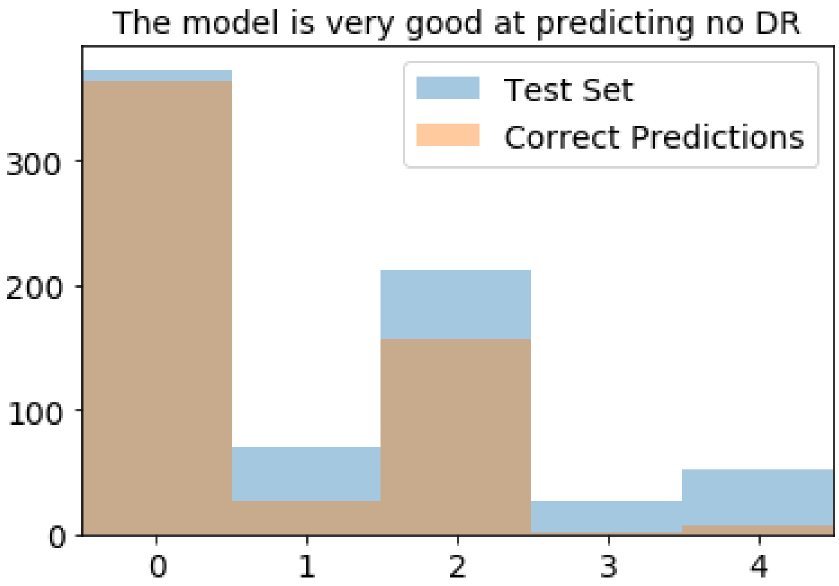
```
In [224]: %%time
          xgb_clf.fit(PC_train,y_train)

Wall time: 1min 13s

Out[224]: XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
                        colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
                        importance_type='gain', interaction_constraints=None,
                        learning_rate=0.300000012, max_delta_step=0, max_depth=6,
                        min_child_weight=1, missing=nan, monotone_constraints=None,
                        n_estimators=100, n_jobs=0, num_class=5, num_parallel_tree=
          1,
                        objective='multi:softprob', random_state=2020, reg_alpha=0,
                        reg_lambda=1, scale_pos_weight=None, subsample=1,
                        tree_method=None, validate_parameters=False, verbosity=Non
          e)
```

Results

Same result w/ either API

```
Precision = 0.5877645479277815
Recall = 0.4579484696597348
Accuracy = 0.757162346521146
```





Using the raw data

Train the model

```
In [144]: %%time
model_orig = xgb.train(param_orig, D_train_orig, steps)
```

```
Wall time: 20min 1s
```

20x slower

Raw Data

```
Precision = 0.631724365975003
Recall = 0.4947558142702054З
Accuracy = 0.7612551159618008
```

PCA

```
Precision = 0.5877645479277815
Recall = 0.457948469659734 8
Accuracy = 0.75716234652114 6
```

Results are slightly better w/ raw data, but it's slower

Next Presentations

Possible follow-up presentations:

- **Hyperparameters** & tuning
  - esp. for multilabel classification (not supported)
- LGBM & CatBoost

Main Sources

- **A series of StatQuest XGBoost Videos**
  - XGBoost Part 1: XGBoost Trees for Regression
  - XGBoost Part 2: XGBoost Trees For Classification
  - XGBoost Part 3: Mathematical Details
- **A Gentle Introduction to XGBoost for Applied Machine Learning**
- **Introduction to Boosted Trees**
- **Original XGBoost Presentation**
- **Original XGBoost Manuscript**
- **XGBoost Docs** (Parameters, Python package)

Thank you!

Questions?