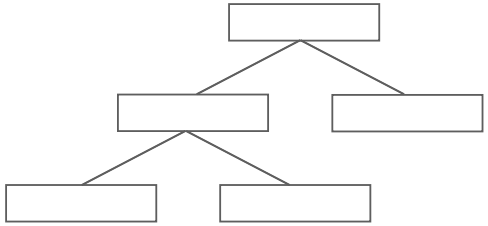
Based Methods

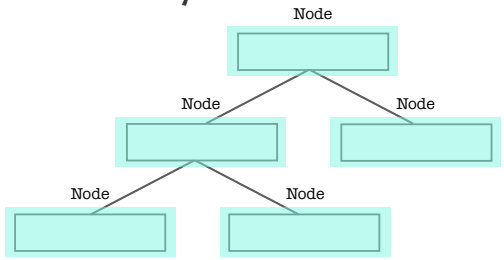
Lecture 9

Termeh Shafie

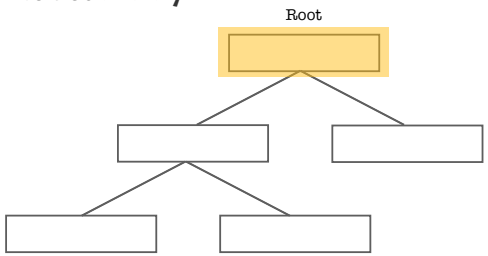
The Vocabulary



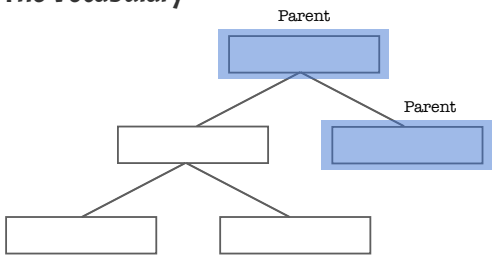
The Vocabulary



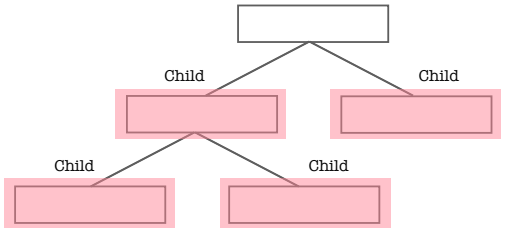
The Vocabulary



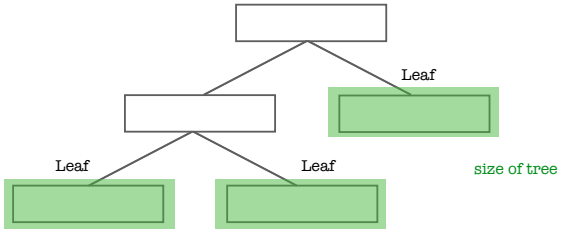
The Vocabulary



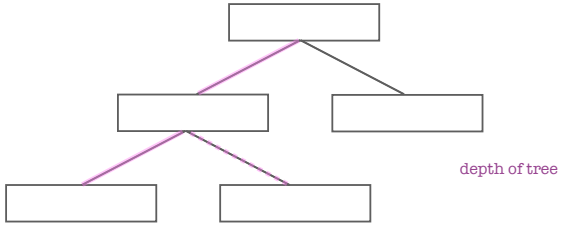
The Vocabulary



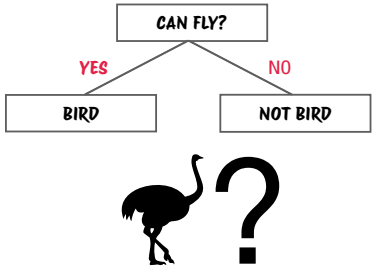
The Vocabulary



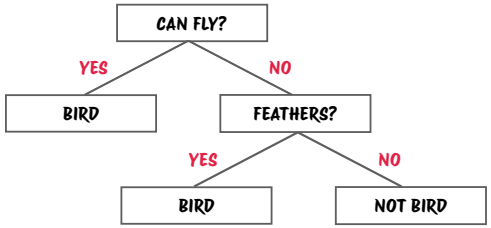
The Vocabulary



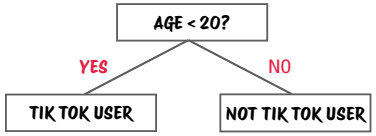
Data Type: Categorical



Data Type: Categorical



Data Type: Continuous

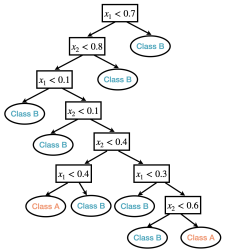


Classification and Regression Trees (CART)

Tree-Based Classification

nonparametric algorithms **partitioning the feature space** into a number of smaller (non-overlapping) regions with **similar response** values using a set of **splitting rules**

example: classification tree based on two predictors



Classification and Regression Trees

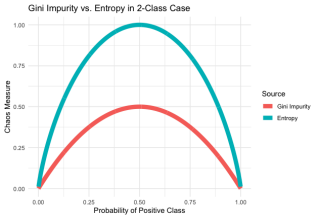
1. Start with an empty decision tree (undivided feature space)
2. Choose the 'optimal' predictor on which to split,
3. choose the 'optimal' threshold value for splitting by applying a **splitting criterion**
4. Recurse on on each new node until stopping condition is met

Splitting Criteria

Gini Index: $GI = 1 - \sum_{i=1}^{n} p_i^2$

Entropy: $H = -\sum_{i}^{n} p_i \log(p_i)$

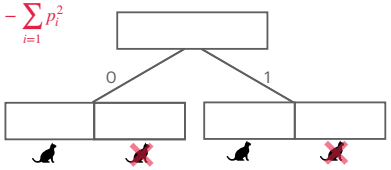
Goal: split where GI or *H* is minimzed



Example

cats	house	ho	children	income
1	0	1	1	34
0	1	0	1	58.3
1	1	1	0	71.5
0	0	0	1	74.9
0	0	0	1	75.3
1	0	0	1	75.6
0	0	0	1	81
1	1	1	0	82.3
1	1	1	0	85.6
1	1	1	1	95.4

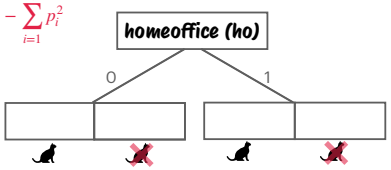
$GI = 1 - \sum_{i=1}^{n} p_i^2$



Example

cats	house	ho	children	income
1	0	1	1	34
0	1	0	1	58.3
1	1	1	0	71.5
0	0	0	1	74.9
0	0	0	1	75.3
1	0	0	1	75.6
0	0	0	1	81
1	1	1	0	82.3
1	1	1	0	85.6
1	1	1	1	95.4

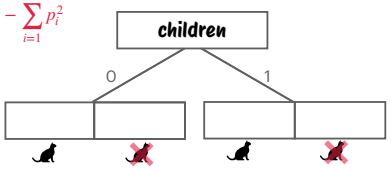
$$GI = 1 - \sum_{i=1}^{n} p_i^2$$



Example

cats	house	ho	children	income
1	0	1	1	34
0	1	0	1	58.3
1	1	1	0	71.5
0	0	0	1	74.9
0	0	0	1	75.3
1	0	0	1	75.6
0	0	0	1	81
1	1	1	0	82.3
1	1	1	0	85.6
1	1	1	1	95.4

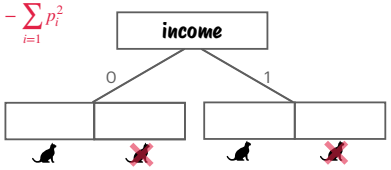
$$GI = 1 - \sum_{i=1}^{n} p_i^2$$



Example

cats	house	ho	children	income
1	0	1	1	34
0	1	0	1	58.3
1	1	1	0	71.5
0	0	0	1	74.9
0	0	0	1	75.3
1	0	0	1	75.6
0	0	0	1	81
1	1	1	0	82.3
1	1	1	0	85.6
1	1	1	1	95.4

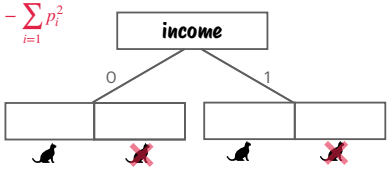
$$GI = 1 - \sum_{i=1}^{n} p_i^2$$



Example

$$GI = 1 - \sum_{i=1}^{n} p_i^2$$

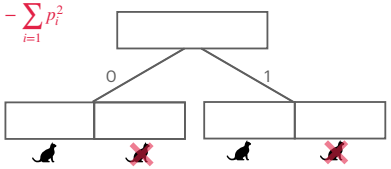
cats	house	ho	children	income	
1	0	1	1	34	0.44
0	1	0	1	58.3	0.48
1	1	1	0	71.5	0.48
0	0	0	1	74.9	0.47
0	0	0	1	75.3	0.4
1	0	0	1	75.6	0.45
0	0	0	1	81	0.34
1	1	1	0	82.3	0.4
1	1	1	0	85.6	0.44
1	1	1	1	95.4	



Example

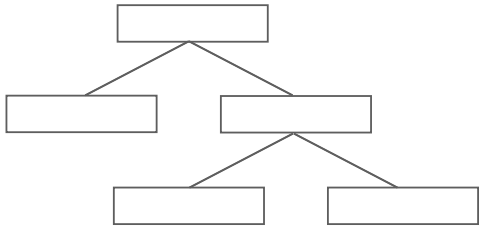
$$GI = 1 - \sum_{i=1}^{n} p_i^2$$

		0.4	0.16	0.343	0.343	
cats	house	ho	children	income		
1	0	1	1	34		
0	1	0	1	58.3		
1	1	1	0	71.5		
0	0	0	1	74.9		
0	0	0	1	75.3		
1	0	0	1	75.6		
0	0	0	1	81		
1	1	1	0	82.3		
1	1	1	0	85.6		
1	1	1	1	95.4		



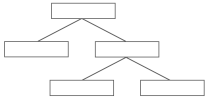
Review: Basic Steps

1. Compute Gini index or Entropy as measure of impurity for each node
2. Choose node with lowest score
3. If the parent node has the lowest score, it is a leaf



Variable Importance Measure: Gini Importance

1. How much does this feature reduce node impurity?



weighted parent node impurity

$$\underbrace{\text{node-impo}_j}_{\text{importance of node j}} = \overbrace{w_j C_j}^{} - \underbrace{\left(w_{\text{left}_j} C_{\text{left}_j} + w_{\text{right}_j} C_{\text{right}_j} \right)}_{\text{weighted child node impurity}}$$

feature importance (fi):

$$\text{fi}_j = \frac{\sum_{j \in S_i} \text{node-impo}_j}{\sum_{k \in S_{all}} \text{nodeimpo}_k}$$ where S_i is set of all nodes that split on feature$_i$

Variable Importance Measure: Permutations

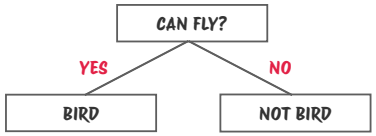
2. How much does reshuffling of a variable reduce model performance?

The larger the discrepancy between baseline model predictions and reshuffled model predictions, the more important is that feature

cats	ho
1	1
0	0
1	1
0	0
0	0
1	0
0	0
1	1
1	1
1	1

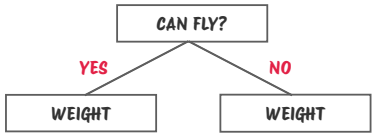
Regression Trees

when what we wish to predict is continuous instead of categorical



Regression Trees

when what we wish to predict is continuous instead of categorical



for every data point that ends up in a leaf, we predict its value is *the mean* of all values that ended up in that node when training

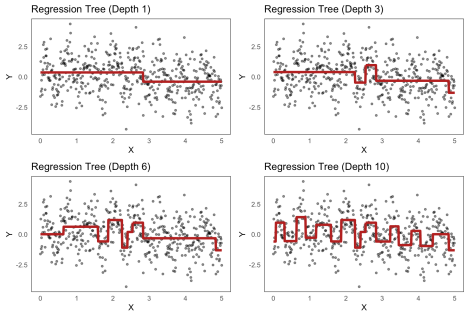
Regression Trees

- For **classification**, purity of the regions is a good indicator the performance of the model
- For **regression**, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by e.g. the MSE

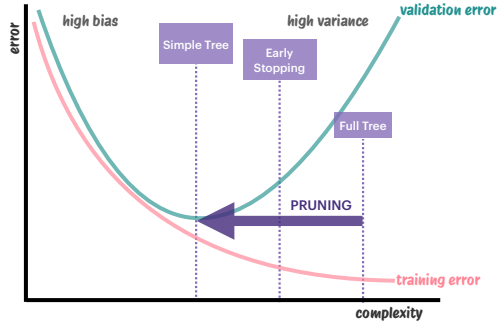
1. start with an empty decision tree
2. choose a predictor on which to split and choose a threshold for splitting such that the weighted average MSE of the new region is as small as possible
3. Recurse on each node until **stopping condition** is met
   - maximum depth
   - minimum number of points in region

instead of purity gain, we instead compute accuracy gain

Regression Trees



Motivation for Pruning



Cost Complexity Pruning

- we can obtain a simpler tree by 'pruning' a complex one
- we select from an array of smaller subtrees of the full model that optimizes a balance of performance and efficiency

$$C(T) = \text{Error}(T) + \alpha |T|$$

where T is a decision subtree

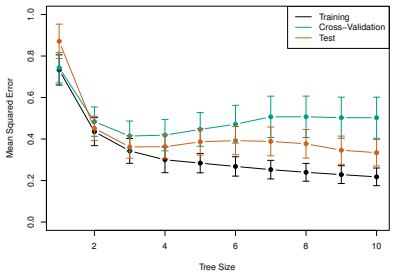
$|T|$ is the number of leaves in the tree

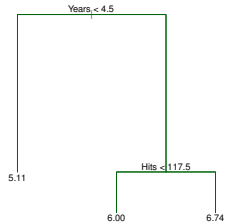
α penalizes model complexity

1. Fix α
2. Find best tree for a given α and based on complexity C
3. Find the best using CV and error measure

Regression Trees: Pruning

Example: Hitters (ISLR2)



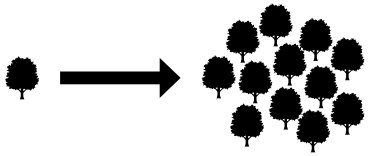


Decision Trees

+ Decision trees models are highly interpretable and fast to train

- In order to capture a complex decision boundary we need a large trees:
  - have high variance and prone to overfitting
  - often underperform compared to other classification/regression methods

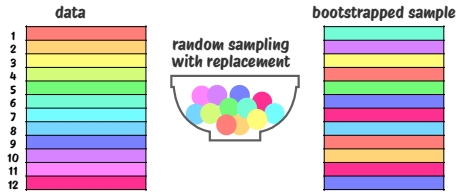
Random Forests



Ensemble method comprising:
- Bagging
- Random Feature Selection

Bagging (Bootstrap Aggregating)

- **Bootstrap:**
  - generate multiple samples of training data via bootstrapping
  - train a full decision tree on each sample of data
- **Aggregate:**
  - given an input, we output the averaged outputs of all the models for that input
- Works with the rows of the data



Random Feature Selection

- Just like bagging but
  - works with **the columns** of the data
  - without replacement
- Every tree will have slightly different predictors

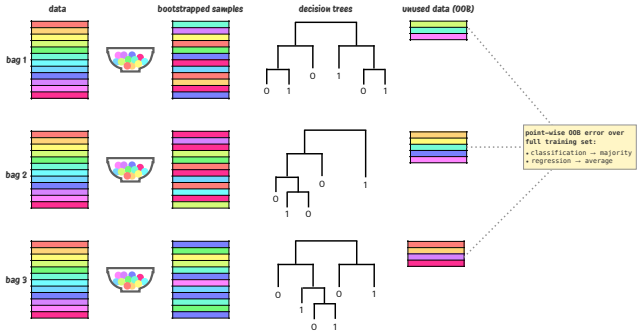
Random Forest

- **Benefits**
  - \+ By using full trees each, each model is able to approximate complex function and decision boundaries
  - \+ By averaging the predictions of all models reduces the variance in the final prediction (given sufficiently large number of trees)
- **Drawback**
  - \- the averaged model is no longer easily interpretable - we can no longer trace the 'logic' of an output through a series of decisions based on predictor values

Tuning the Random Forest

- Random forest models have multiple hyper-parameters to tune
  - the number of predictors to randomly select at each split
  - the total number of trees in the ensemble
  - the minimum leaf node size
- Generally tuned through cross validation ($\Longrightarrow$ data and problem dependent)
- Use **out-of-bag errors** to evaluate model's predictive accuracy
  - cease training once the out-of-bag error stabilizes
  - if sample large enough, estimate is approximately LOO-CV error for bagging
- When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly
- Increasing number of trees in ensemble does not increase risk of overfitting
  - but the trees in the ensemble may become more correlated, increase the variance.

Out-Of-Bag Errors (OOB Errors)



Boosting Trees

Boosting Trees

The trees are grown sequentially:

- Each tree is grown using information from previously grown trees
- The boosting approach *learns slowly*, thus avoiding overfitting

correct errors made by all previous trees

Boosting Trees

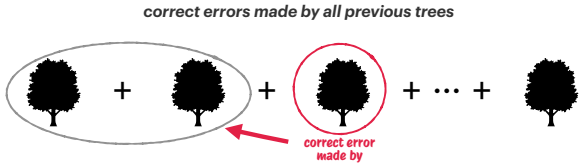
The trees are grown sequentially:

- Each tree is grown using information from previously grown trees
- The boosting approach *learns slowly*, thus avoiding overfitting

Boosting Trees

The trees are grown sequentially:

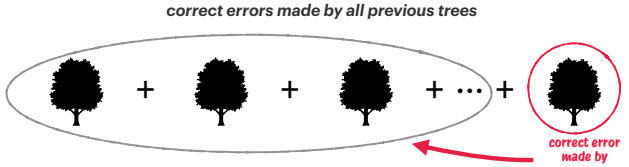
- Each tree is grown using information from previously grown trees
- The boosting approach *learns slowly*, thus avoiding overfitting



Boosting Trees

The trees are grown sequentially:

- Each tree is grown using information from previously grown trees
- The boosting approach *learns slowly*, thus avoiding overfitting



Example

	age	initial guess	residual
obs 1	21	22	-1
obs 2	22	22	0
obs 3	23	22	1
obs 4	22	22	0
obs 5	21	22	-1

actual value = predicted + residual

what if we had a tree that could predict the residuals made by the initial model?

$\Longrightarrow$ ***gradient boosting tree!***

instead of fitting a bunch of independent trees, we incrementally improve on our initial guess

Example

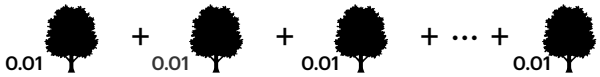
	age	initial guess	residual
obs 1	21	22	-1
obs 2	22	22	0
obs 3	23	22	1
obs 4	22	22	0
obs 5	21	22	-1

actual value = predicted + residual

what if we had a tree that could predict the residuals made by the initial model?

$\Longrightarrow$ ***gradient boosting tree!***

instead of fitting a bunch of independent trees, we incrementally improve on our initial guess

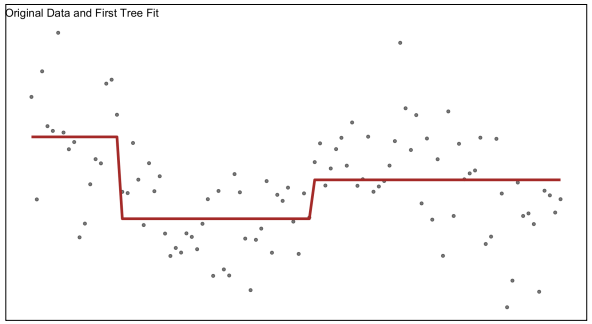


Gradient Boosting Trees

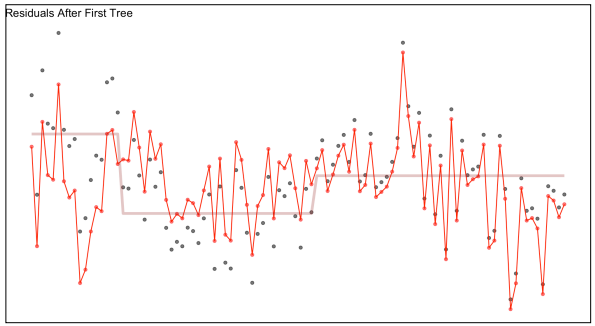
Intuitively:

- Gradient boosting is a method for iteratively building a complex model T by adding simple models.
- Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble:
  - each simple model $T^{(i)}$ we add to our ensemble model T, models the errors of T
  - Thus, with each addition of $T^{(i)}$, the residual is reduced

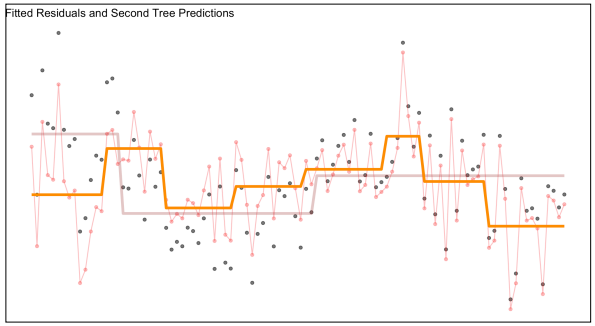
Gradient Boosting Trees



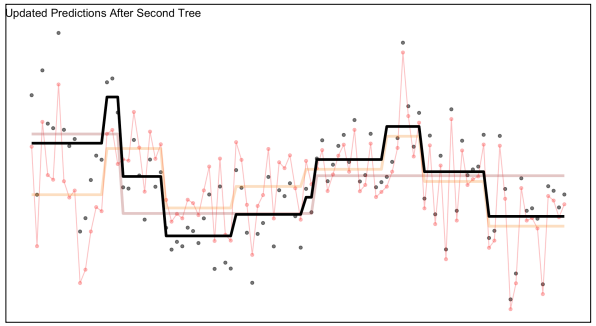
Gradient Boosting Trees



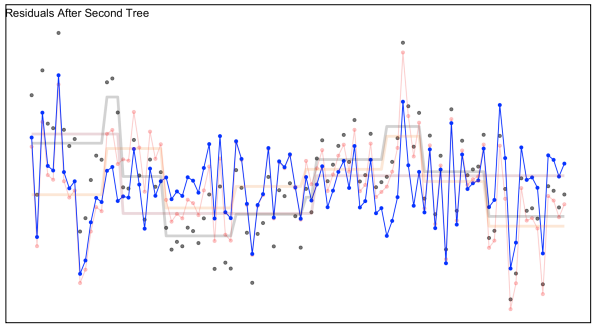
Gradient Boosting Trees



Gradient Boosting Trees



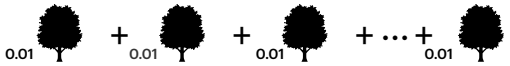
Gradient Boosting Trees



Gradient Boosting Trees: The Algorithm

1. Fit a simple model $T^{(0)}$ on the training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
   Set $T \leftarrow T^{(0)}$ and compute residuals $\{r_1, \ldots, r_N\}$ fort T
2. Fit a simple model $T^{(1)}$ to the current **residuals**, i.e. train using $\{(x_1, r_1), \ldots, (x_N, r_N)\}$
3. Set $T \leftarrow T + \lambda T^{(1)}$ where λ is the learning rate (usually 0.01 or 0.001)
4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^{(i)}(x_n),\ n = 1, \ldots, N$
5. Repeat steps 2-4 until stopping condition is met

Gradient Boosting Trees: The Math



$$z_0 + 0.01 z_1 + 0.01 z_2 + \cdots + 0.01 z_n$$

baseline prediction

how do we choose these errors to correct?

future trees predict error for a regression tree given defined loss function

let F_i be our predictions $F_i = \sum_{t=0}^{i} z_t$

$$F_1 = z_0 + z_1$$
$$F_2 = z_0 + z_1 + z_3$$
$$\vdots$$

$$F_i = F_{i-1} + z_i \qquad z_i = -\frac{\partial Loss(y, F_i)}{\partial F_i}$$

Gradient Boosting Trees: The Math

$$z_i = -\frac{\partial Loss(y, F_i)}{\partial F_i}$$

Negative Gradient of Loss w.r.t. Ensemble Prediction

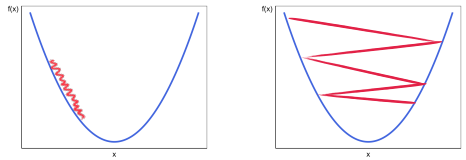
- The Negative Gradient tell us what adjustments we should make to our prediction F_i in order to decrease our loss
- Example:

$$Loss(y, \hat{y}) = (y - \hat{y})^2 \implies -\frac{\partial Loss(y, \hat{y})}{\partial \hat{y}} \implies 2(y - \hat{y})$$

  - With squared loss, error is the negative gradient, but the negative gradient will work in other situations!

Choosing a Learning Rate: Convexity

- Under ideal conditions, gradient descent iteratively approximates and converges to the optimum
- For a constant learning rate λ
  - if λ is too small, it takes too many iterations to reach the optimum
  - if λ is too large, algorithm may 'bounce' around the optimum and never get close



  - Better to treat learning rate as a variable, that is let the value depend on gradient
  - around optimum λ is small, and far from optimum λ is larger

This Week's Practical