

# Building a Robot Judge: Data Science for Decision-Making

## 5. Classification

## Response Essay RE01 First Draft Due Sunday

(see link from syllabus)

- ▶ Critically read and review an application paper.
- ▶ 300 words is the minimum for a passing grade but 500+ words would be expected for high grade.
- ▶ Please anonymize your submission – put your name in the filename, but not anywhere in the file. Submit as TXT or PDF.
- ▶ First draft is not graded (completion for minimum word count) but will get peer feedback from classmates.

## Why Response Essays?

- ▶ Critical reading is an important skill:
  - ▶ useful for writing/reading reports
  - ▶ understanding the structure/code behind a paper – why have papers and not textbooks?

## What can I write about?

Any “Applications” reading from first page.

**Or another social science article applying tools from one of the first topics (machine learning, causal inference with linear regression, fixed effects models, or classification). Please confirm with me by email by tomorrow.**

## What to think about

- ▶ beyond a minimum summary, no credit for summarizing!

### **Questions to ask in reading responses**

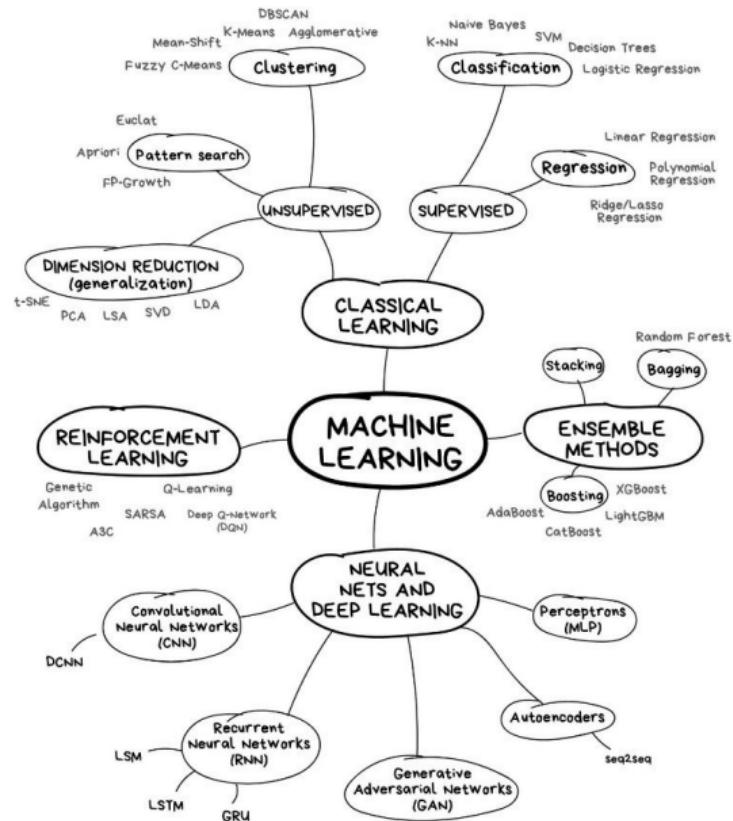
1. What is the research question?
2. What is interesting about the data? Is it the right dataset to answer the research question?
3. Did they provide sufficient visuals and descriptive statistics to provide trust in the data and its usefulness for the stated purpose?
4. What is the goal of the data analysis?
  - a. What are they trying to measure?
  - b. What are they trying to predict or learn?
5. Given the answer to #4, is the right model being used? What other models could they have tried?
6. Did they provide validation that the model is delivering on the stated goals (#4)?
7. How were the model predictions/statistics used in a social-science analysis? What results seemed incomplete or non-robust?
8. Did they answer the research question (#1)? Highlight limitations and open questions.

Another nice guide (on HW Assignments page):

[https://www.icpsr.umich.edu/files/instructors/How\\_to\\_Read\\_a\\_Journal\\_Article.pdf](https://www.icpsr.umich.edu/files/instructors/How_to_Read_a_Journal_Article.pdf)

- ▶ could be useful for peer feedback on classmates' response essays.

## The Machine Learning Landscape



# What we have done so far



# What's Next



# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

Introduction to Neural Nets

Basics

Regularizing neural nets

Applications

## Machine learning metrics

- ▶ For regression, evaluation is relatively straightforward – we have mean squared error, mean absolute error, or R-squared (see Week 3 slides).

## Machine learning metrics

- ▶ For regression, evaluation is relatively straightforward – we have mean squared error, mean absolute error, or R-squared (see Week 3 slides).
- ▶ For classification, things are more complicated.
- ▶ As a (good) baseline, we have:
  - ▶  $\text{accuracy} = (\# \text{ correct test-set predictions}) / (\# \text{ of test-set observations})$

## Machine learning metrics

- ▶ For regression, evaluation is relatively straightforward – we have mean squared error, mean absolute error, or R-squared (see Week 3 slides).
- ▶ For classification, things are more complicated.
- ▶ As a (good) baseline, we have:
  - ▶  $\text{accuracy} = (\# \text{ correct test-set predictions}) / (\# \text{ of test-set observations})$
- ▶ But what if one of the outcomes is rare – e.g., one out of 20?
  - ▶ Then I can guess the modal class and get 95% accuracy.
- ▶ As we will see, there are a range of other useful metrics besides accuracy for evaluating classifier performance.

## Confusion Matrix

A nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
True Class	Negative	# True Negatives	# False Positives
	Positive	# False Negatives	# True Positives

- ▶ Cell values give counts in the test set.

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(y_test,y_pred)
```

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

## Precision and Recall

		Predicted Class	
		Negative	Positive
True Class	Negative	# True Negatives	# False Positives
	Positive	# False Negatives	# True Positives

```
from sklearn.metrics import precision_score, recall_score  
precision_score(y_test,y_pred)  
recall_score(y_test,y_pred)
```

## Precision and Recall

		Predicted Class	
		Negative	Positive
True Class	Negative	# True Negatives	# False Positives
	Positive	# False Negatives	# True Positives

```
from sklearn.metrics import precision_score, recall_score  
precision_score(y_test,y_pred)  
recall_score(y_test,y_pred)
```

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- ▶ Precision decreases with false positives. “When I guess this outcome, I tend to guess correctly.”

## Precision and Recall

		Predicted Class	
		Negative	Positive
True Class	Negative	# True Negatives	# False Positives
	Positive	# False Negatives	# True Positives

```
from sklearn.metrics import precision_score, recall_score  
precision_score(y_test,y_pred)  
recall_score(y_test,y_pred)
```

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- ▶ Precision decreases with false positives. “When I guess this outcome, I tend to guess correctly.”

$$\text{Recall (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- ▶ Recall decreases with false negatives. “When this outcome occurs, I don’t miss it.”

**Note:** scikit-learn will produce precision/recall for  $Y = 1$  by default

## Trading off precision and recall

- ▶ Recall our decision framework:
  - ▶ a decision-maker who has to make a decision  $W$ , that will produce some value or benefit, conditional on the value of  $Y$ :
$$V = u(W, Y)$$
- ▶ One can imagine decision contexts where precision/recall should be valued asymmetrically. This is part of the function  $u(W, Y)$ .

## Trading off precision and recall

- ▶ Recall our decision framework:
  - ▶ a decision-maker who has to make a decision  $W$ , that will produce some value or benefit, conditional on the value of  $Y$ :
$$V = u(W, Y)$$
- ▶ One can imagine decision contexts where precision/recall should be valued asymmetrically. This is part of the function  $u(W, Y)$ .
- ▶ **For these decisions, would you prefer high precision or high recall?**
  1. deciding “send to prison” in court
  2. detecting bombs during flight screening

## Balanced Accuracy and F1 Score

- ▶ If labels are (almost) balanced, then accuracy (share correct predictions) is a decent metric.
  - ▶ If not (say 90% in one category), accuracy will be uninformative/misleading.

## Balanced Accuracy and F1 Score

- ▶ If labels are (almost) balanced, then accuracy (share correct predictions) is a decent metric.
  - ▶ If not (say 90% in one category), accuracy will be uninformative/misleading.
- ▶ A standard metric in this case is **balanced accuracy** = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2} \left( \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right)$$

- ▶ → equal to accuracy when classes are balanced, or performance is the same across classes.

## Balanced Accuracy and F1 Score

- ▶ If labels are (almost) balanced, then accuracy (share correct predictions) is a decent metric.
  - ▶ If not (say 90% in one category), accuracy will be uninformative/misleading.
- ▶ A standard metric in this case is **balanced accuracy** = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2} \left( \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right)$$

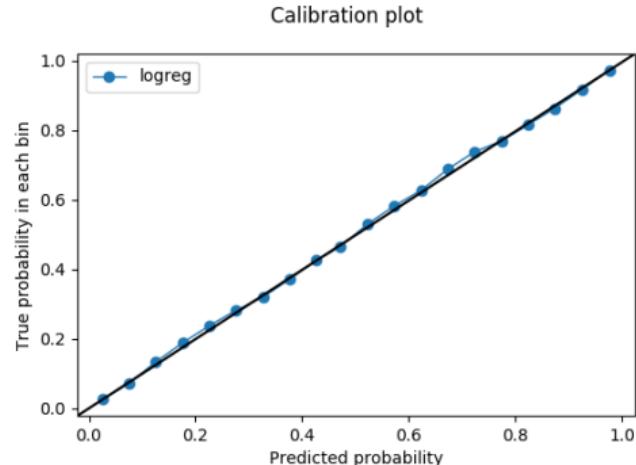
- ▶ → equal to accuracy when classes are balanced, or performance is the same across classes.
- ▶ Another standard metric is  $F_1$  score = the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- ▶ penalizes both false positives and false negatives. still ignores true negatives.

```
from sklearn.metrics import balanced_accuracy_score, f1_score
balanced_accuracy_score(y_test,y_pred)
f1_score(y_test,y_pred) # gives values for both classes
```

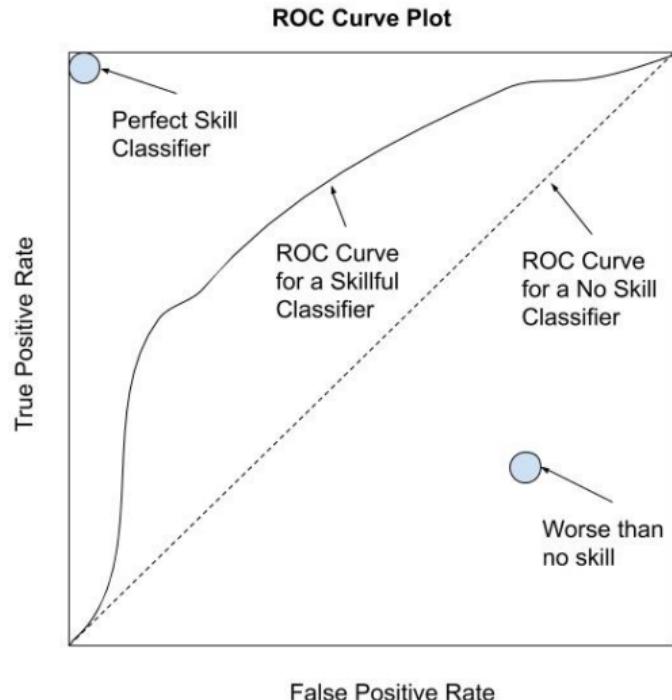
# Evaluating Classification Models: Calibration Curves



- ▶ Plotting the binned fraction in a category (Y axis) against the predicted probability in a category (X axis):
- ▶ Provides evidence of whether the classifier is replicating the conditional distribution of the outcome.

```
from seaborn import regplot  
regplot(y_test, y_pred, x_bins=20)
```

# ROC Curve and AUC



**AUC = area under the (ROC) curve**

- ▶ provides an aggregate measure of performance across all possible classification thresholds.

**Interpreting AUC:**

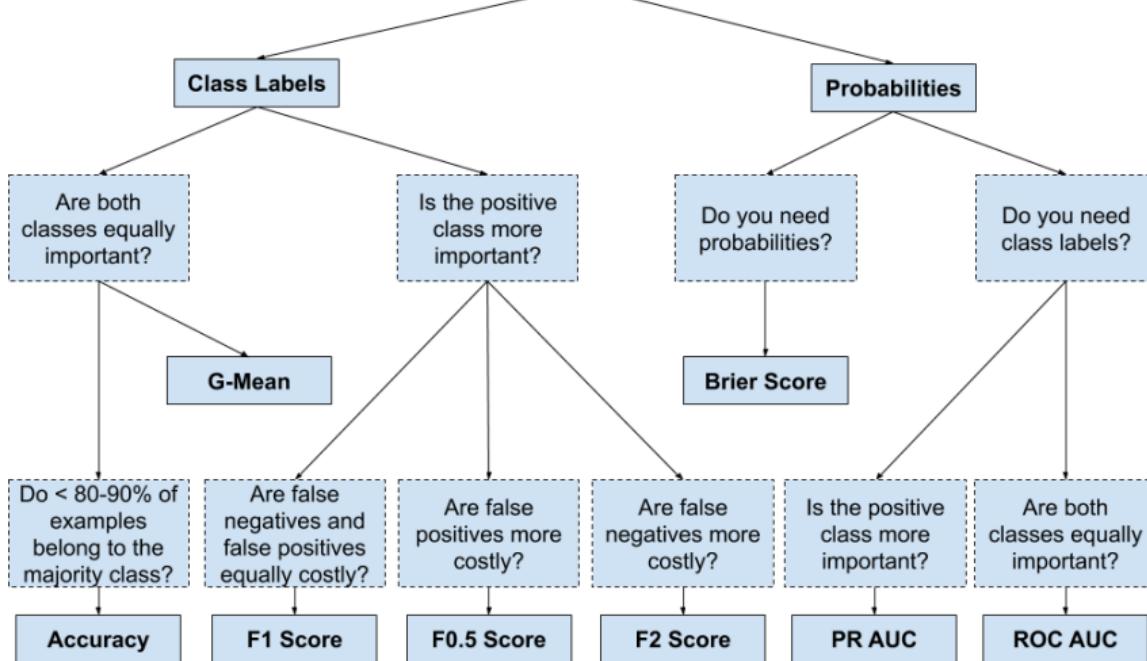
- ▶ = probability that the model (correctly) ranks a random positive example more highly than a random negative example.

```
from sklearn.metrics import roc_auc_score  
roc_auc_score(y_test,y_pred)
```

## Imbalanced Binary Classification

### How to Choose A Performance Metric

What do you  
want to  
predict?



© 2019 MachineLearningMastery.com All Rights Reserved.

See details in “Tour of evaluation metrics for imbalanced classification” (on syllabus).

# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

Introduction to Neural Nets

Basics

Regularizing neural nets

Applications

## Multi-Class Models

Many interesting machine learning problems involve multiple un-ordered categories:

- ▶ categorizing a case by area of law.
- ▶ predicting the political party of a speaker in a multi-party system.
- ▶ predicting authorship from documents
- ▶ image classification / object detection

## Multiple Classes: Setup

- ▶ The outcome is  $y_i \in \{1, \dots, k, \dots, n_y\}$  output classes, which can also be represented as a one-hot vector

$$\mathbf{y}_i = \{\mathbf{1}[y_i = 1], \dots, \mathbf{1}[y_i = n_y]\}$$

## Multiple Classes: Setup

- ▶ The outcome is  $y_i \in \{1, \dots, k, \dots, n_y\}$  output classes, which can also be represented as a one-hot vector

$$\mathbf{y}_i = \{\mathbf{1}[y_i = 1], \dots, \mathbf{1}[y_i = n_y]\}$$

- ▶ We want to learn a vector function

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \theta)$$

outputting a vector of probabilities across outcomes as a function of the inputs:

$$\hat{\mathbf{y}} = \{\hat{y}^1, \dots, \hat{y}^{n_y}\}, \hat{y}^k \in [0, 1] \quad \forall k$$

- ▶ for prediction, select the highest-probability class:

$$\tilde{y} = \arg \max_k \hat{y}_{[k]}$$

## Categorical Cross Entropy

- ▶ The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = - \sum_{k=1}^{n_y} \mathbf{y}_{[k]} \log(\hat{\mathbf{y}}_{[k]}(\mathbf{x}, \theta))$$

- ▶ measures dissimilarity between the true label distribution  $\mathbf{y}$  and the predicted label distribution  $\hat{\mathbf{y}}$ .

## Categorical Cross Entropy

- ▶ The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = - \sum_{k=1}^{n_y} \mathbf{y}_{[k]} \log(\hat{\mathbf{y}}_{[k]}(\mathbf{x}, \theta))$$

- ▶ measures dissimilarity between the true label distribution  $\mathbf{y}$  and the predicted label distribution  $\hat{\mathbf{y}}$ .
- ▶ Since there is just one true class ( $y = 1$  for one class  $k^*$ , and zero for others), simplifies to

$$L(\theta) = - \log(\hat{\mathbf{y}}_{[k^*]}(\mathbf{x}, \theta))$$

- ▶ Rewards putting higher probability on the true class, ignores distribution of probabilities on other classes.
- ▶ function is convex → gradient descent will find the optimum.

## Multinomial Logistic Regression

Multinomial logistic regression computes probabilities for each class  $k$  using the softmax transformation

$$\hat{y}_k(\mathbf{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta'_k \mathbf{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta'_l \mathbf{x}_i)}$$

- ▶ softmax is the multiclass generalization of sigmoid. can then interpret  $\hat{y}$  as probabilities.
- ▶  $n_x$  features and  $n_y$  output classes  $\rightarrow$  there is a  $n_y \times n_x$  parameter matrix  $\Theta$ , where the parameters for each class  $\theta_k$  are stored as rows.
- ▶ the prediction  $y_i \in \{1, \dots, n_y\}$  is determined by the highest-probability category.

## Regularized Multinomial Logistic

- ▶ The L2-penalized logistic regression loss (the default in sklearn) is

$$\mathcal{L}(\theta) = -\frac{1}{n_D} \sum_{i=1}^{n_D} \log \frac{\exp(\theta'_{k^*} \mathbf{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta'_l \mathbf{x}_i)} + \lambda \sum_{j=1}^{n_x} \sum_{k=1}^{n_y} (\theta_{[j,k]})^2$$

- ▶  $\lambda$  = strength of L2 penalty (could also add lasso penalty)
- ▶ as before, predictors should be scaled to the same variance.

```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(X,y)
```

- ▶ if  $y$  is categorical, sklearn automatically uses multinomial logistic

## Multi-Class Confusion Matrix

		Predicted Class		
		Class A	Class B	Class C
True Class	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

- More generally, can have a confusion matrix  $M$  with items  $M_{ij}$  (row  $i$ , column  $j$ ).

## Multi-Class Performance Metrics

Confusion matrix  $M$  with items  $M_{ij}$  (row  $i$ , column  $j$ ).

$$\text{Precision for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Positives for } k} = \frac{M_{kk}}{\sum_l M_{lk}}$$

$$\text{Recall for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Negatives for } k} = \frac{M_{kk}}{\sum_l M_{kl}}$$

$$F_1(k) = 2 \times \frac{\text{precision}(k) \times \text{recall}(k)}{\text{precision}(k) + \text{recall}(k)}$$

- ▶ in sklearn, syntax is the same as binary case.

## Metrics for whole model

$$\text{Balanced Accuracy} = \frac{1}{n_y} \sum_k \text{Recall for } k$$

## Metrics for whole model

$$\text{Balanced Accuracy} = \frac{1}{n_y} \sum_k \text{Recall for } k$$

- ▶ **Macro-averaging:** average of the per-class precision, recall, and F1, e.g.

$$F_1 = \frac{1}{n_y} \sum_{k=1}^{n_y} F_1(k)$$

- ▶ weights all classes equally

## Metrics for whole model

$$\text{Balanced Accuracy} = \frac{1}{n_y} \sum_k \text{Recall for } k$$

- ▶ **Macro-averaging:** average of the per-class precision, recall, and F1, e.g.

$$F_1 = \frac{1}{n_y} \sum_{k=1}^{n_y} F_1(k)$$

- ▶ weights all classes equally
- ▶ Both of these approaches up-weight frequent classes:
  - ▶ **Micro-averaging:** Compute model-level sums for true positives, false positives, and false negatives; compute precision/recall from model sums.
  - ▶ **“Weighted”:** computed like macro-averaging, but classes are weighted by true frequency.

## "Rookie data science mistakes invalidates a dozen medical studies"

When Gilles Vandewiele noticed a large number of studies reporting near-perfect accuracy in predicting whether would-be mothers will undergo premature delivery, his jaw dropped. This was huge.

## “Rookie data science mistakes invalidates a dozen medical studies”

When Gilles Vandewiele noticed a large number of studies reporting near-perfect accuracy in predicting whether would-be mothers will undergo premature delivery, his jaw dropped. This was huge.

Now, it seemed that with the help of artificial intelligence, researchers had managed to solve the puzzle. Swept with excitement, Vandewiele, a Ph.D. candidate in machine learning at Ghent University, recruited his peers and set out to replicate the mind-boggling results. Little did he know they were about to embark on a journey of total scientific annihilation, resulting in the invalidation of almost a dozen peer-reviewed articles.

# "Rookie data science mistakes invalidates a dozen medical studies"

When Gilles Vandewiele noticed a large number of studies reporting near-perfect accuracy in predicting whether would-be mothers will undergo premature delivery, his jaw dropped. This was huge.

Now, it seemed that with the help of artificial intelligence, researchers had managed to solve the puzzle. Swept with excitement, Vandewiele, a Ph.D. candidate in machine learning at Ghent University, recruited his peers and set out to replicate the mind-boggling results. Little did he know they were about to embark on a journey of total scientific annihilation, resulting in the invalidation of almost a dozen peer-reviewed articles.

To their amazement, Vandewiele's team discovered that the authors of the too-good-to-be-true studies were performing oversampling *before* splitting the dataset into two. Because the split was done randomly, this had the devastating side-effect of the same datapoints ending up in both the training and testing set. In effect, the models were being shown the questions they were to be assessed on well before the exam! No wonder their results were suspiciously great.

## Nested Sample Splits

## Nested Sample Splits

To do evaluations in the full data, use nested cross-validation:

- ▶ split data into  $K$  folds, e.g. 5.
- ▶ for each fold  $k \in \{1, 2, \dots, K\}$ :
  - ▶ train and tune model in rest of data  $\neg k$
  - ▶ evaluate metrics (e.g. MSE, balanced accuracy) in  $k$ .
- ▶ Report mean and s.d. of metrics across folds.

# Week 2 Review: Four Types of Confounders



Andy Slavitt    
@ASlavitt

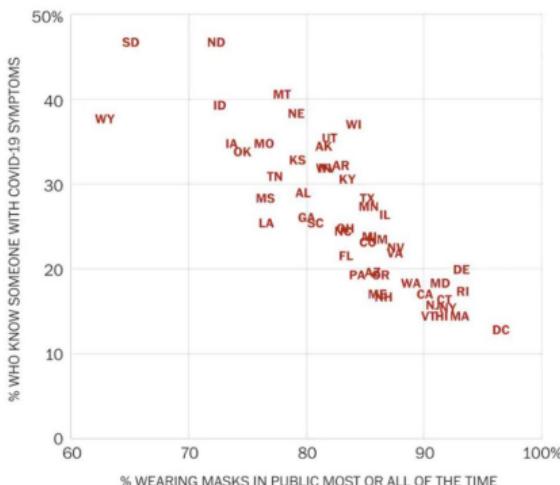
We don't need a high tech miracle as much as we need some people to believe in simple cause and effect.

...

Recall that there are four types of confounders. Based on the first letter your last name, think of an example confounder for this proposed causal relationship.

1. [A-G] positively correlated with outcome, positively correlated with treatment
2. [H-M] positively correlated with outcome, negatively correlated with treatment
3. [N-S] negatively correlated with outcome, positively correlated with treatment
4. [T-Z] negatively correlated with outcome, negatively correlated with treatment

Write it down privately. Which direction is the bias of the estimate (up or down), relative to the true causal effect (if this is the only confounder)? (3 minutes)



Source: Delphi COVIDCast, Carnegie Mellon University

THE WASHINGTON POST

## Week 3 Review: What line # has a bug?

**Write it down (90 seconds)**

```
1 X_train, X_test, y_train, y_test = train_test_split(X,y_true)
2 lasso = Lasso()
3 lasso.fit(X_train, y_train)
4 y_pred = lasso.predict(X)
5 mae = mean_absolute_error(y_true,y_pred)
```

- ▶ Note: mean absolute error ( $\sum |y - \hat{y}|$ ) is an alternative regression metric that is less sensitive to outliers than mean squared error ( $\sum (y - \hat{y})^2$ ).

## Week 4 Review: Confounding factors for a tax cut (3 minutes)

- ▶ Imagine that all the French-speaking cantons of Switzerland enact a tax cut and you estimate a negative effect on local employment using **two-way fixed-effects regression** (TWFE). What are the two types of fixed effects associated with “two-way”? What do they adjust for? What are some potential confounding factors that are not adjusted for by TWFE, that would still bias this estimate?
  - ▶ write down your answer privately.

# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

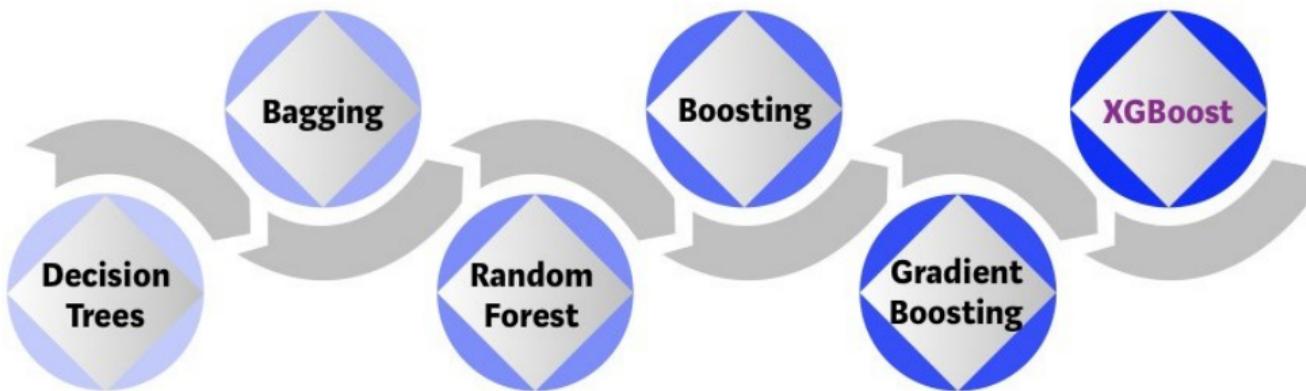
Introduction to Neural Nets

Basics

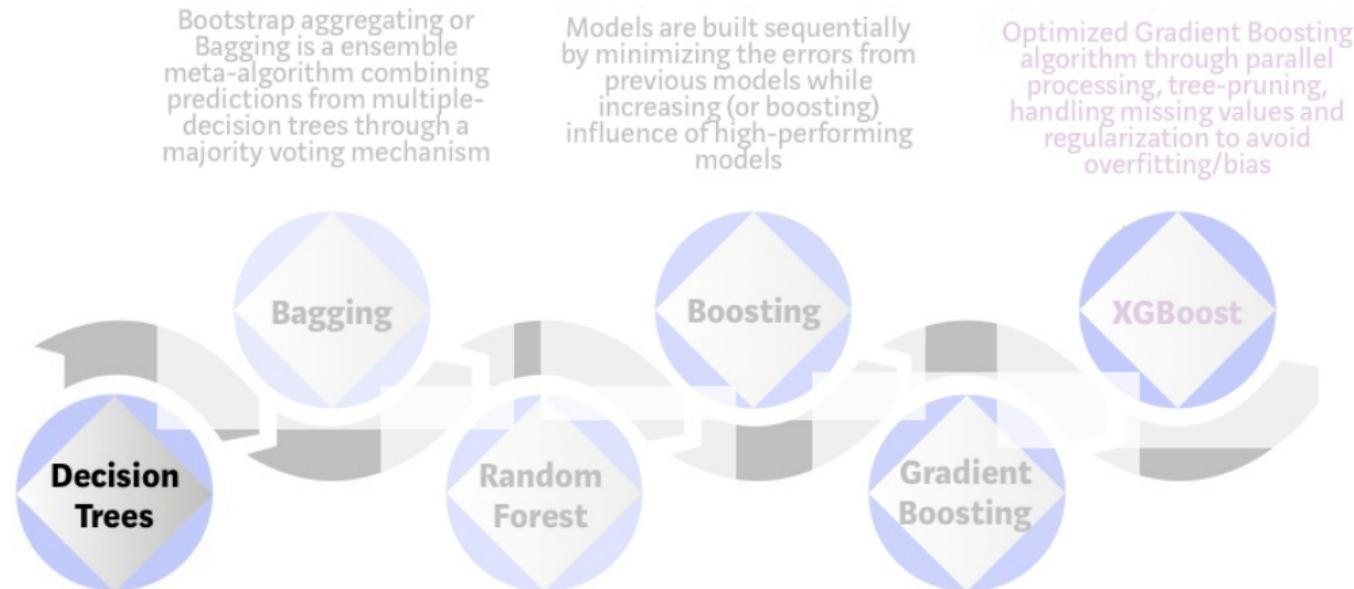
Regularizing neural nets

Applications

# XGBoost: Overview



# XGBoost Ingredients: Decision Trees



A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

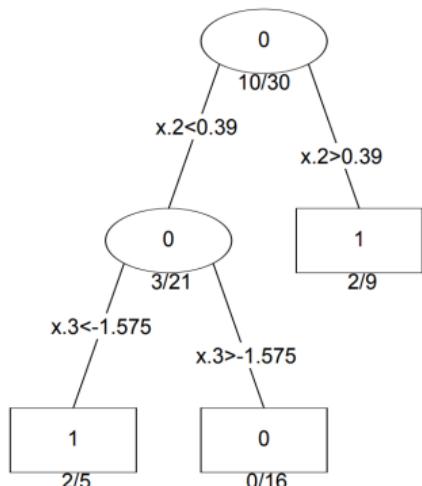
Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

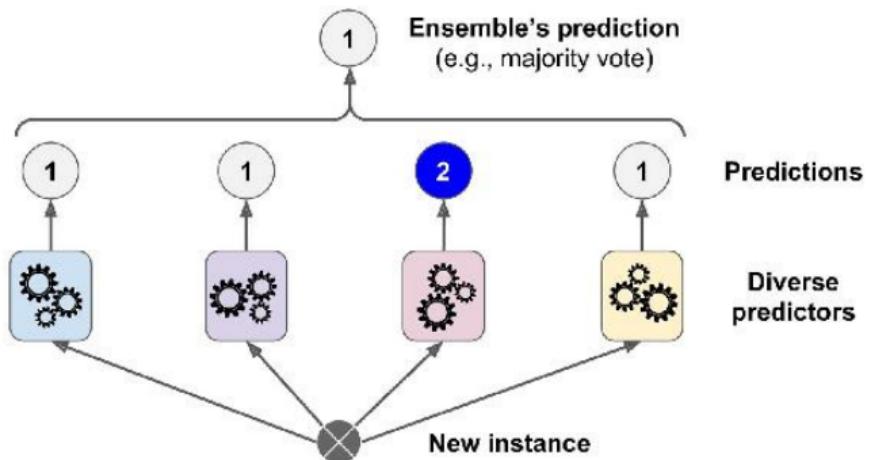
# Decision Trees

## Classification Tree



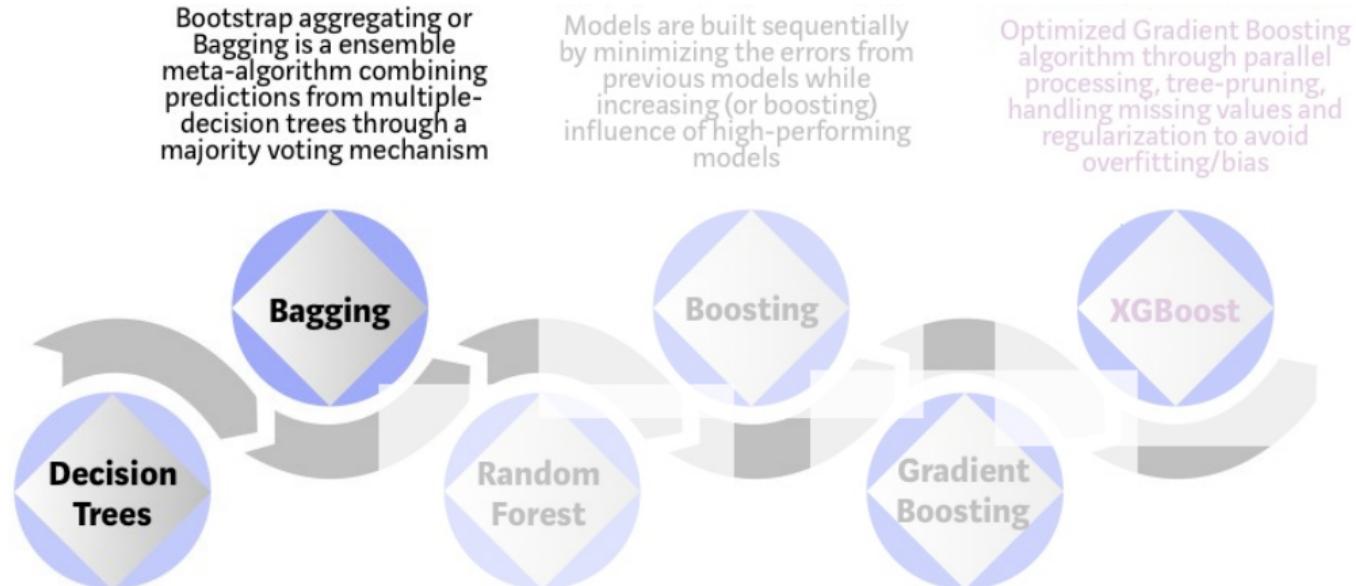
- ▶ Decision trees learn a series of binary splits in the data based on hard thresholds.
  - ▶ if yes, go right; if no, go left.
- ▶ Can have additional splits as you move through the tree.
- ▶ fast and interpretable, but performance is often poor.

## Voting Classifiers



- ▶ voting classifiers (ensembles of different models that vote on the prediction) generally out-perform the best classifier in the ensemble.
  - ▶ more diverse algorithms will make different types of errors, and improve your ensemble's robustness.

# XGBoost Ingredients: Bootstrapping



A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

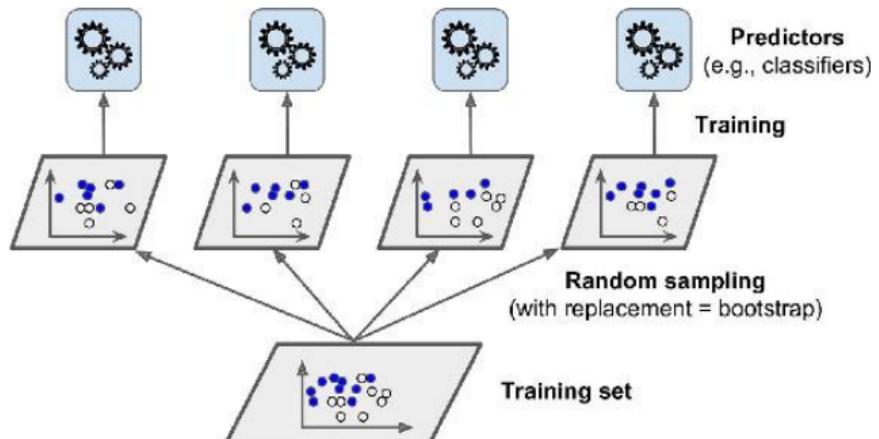
Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

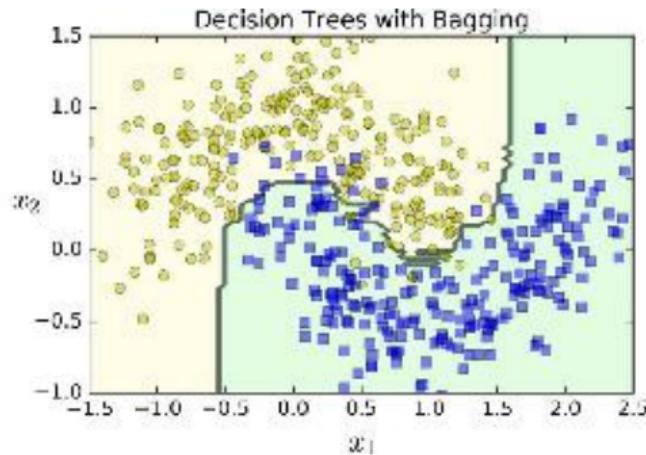
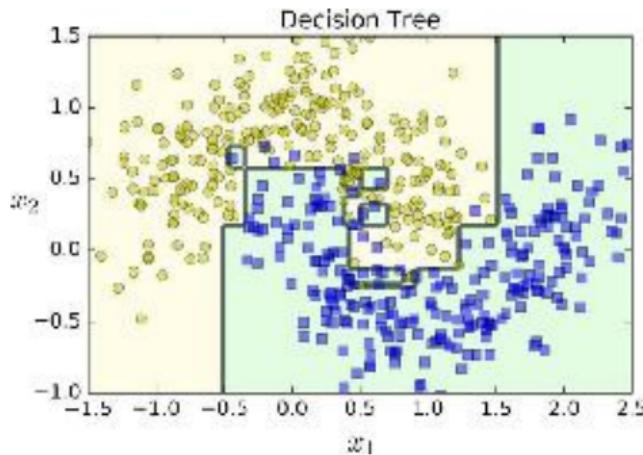
# Bootstrapping

- ▶ Rather than use the same data on different classifiers, one can use different subsets of the data on the same classifier:



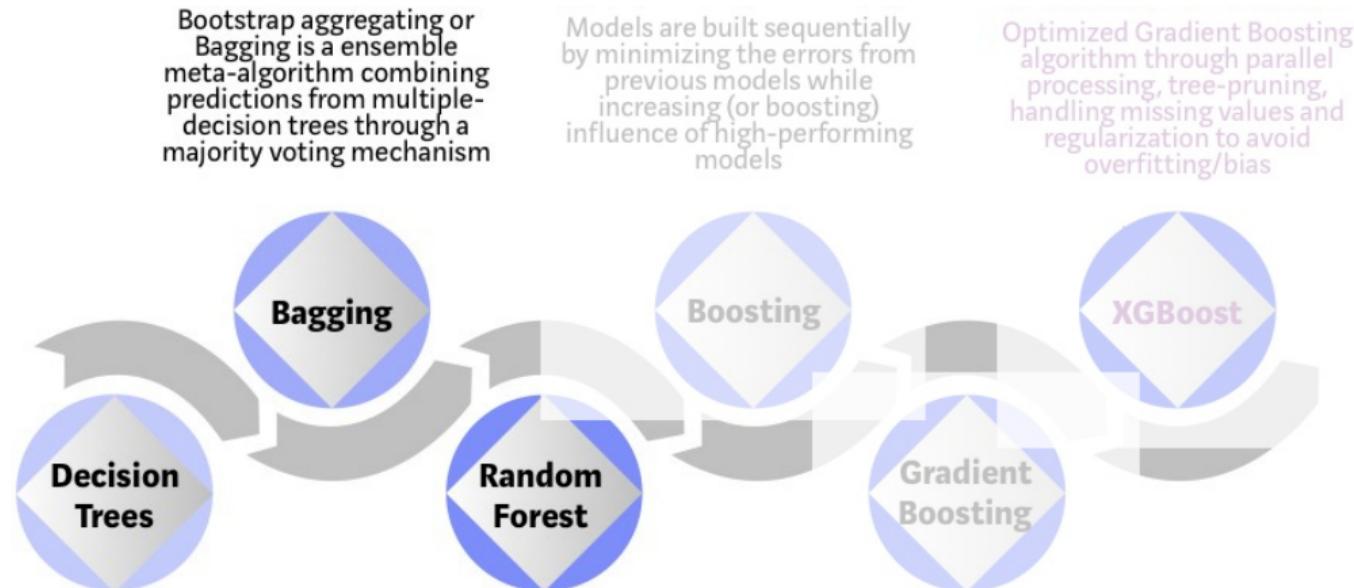
- ▶ can also use different subsets of features across sub-classifiers.

## Bootstrapping Benefits



- ▶ A bootstrapped ensemble generally has a similar bias but lower variance than a single predictor trained on all the data.
- ▶ Predictors can be trained in parallel using separate CPU cores.

# XGBoost Ingredients: Random Forests



A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

## Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier()  
rfc.fit(X,y)
```

## Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier()  
rfc.fit(X,y)
```

1. Each voting tree gets its own sample of data.

## Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier()  
rfc.fit(X,y)
```

1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.

## Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier()  
rfc.fit(X,y)
```

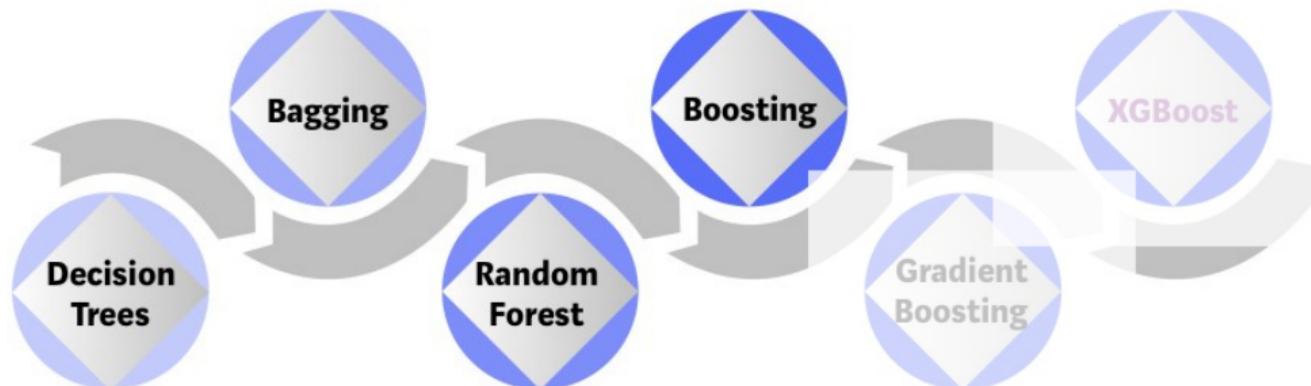
1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.
3. For each tree, error rate is computed using data outside its bootstrap sample.

# XGBoost Ingredients: Boosting

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias



A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

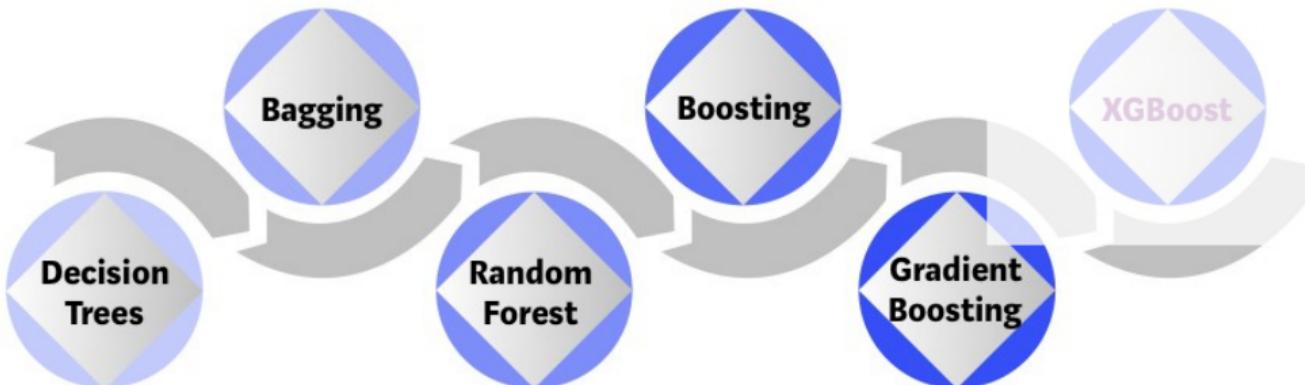
- ▶ works by sequentially adding predictors to an ensemble – fits the new predictor to the residual errors made by the previous predictor to gradually improve the model.

# XGBoost Ingredients: Gradient Boosting

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias



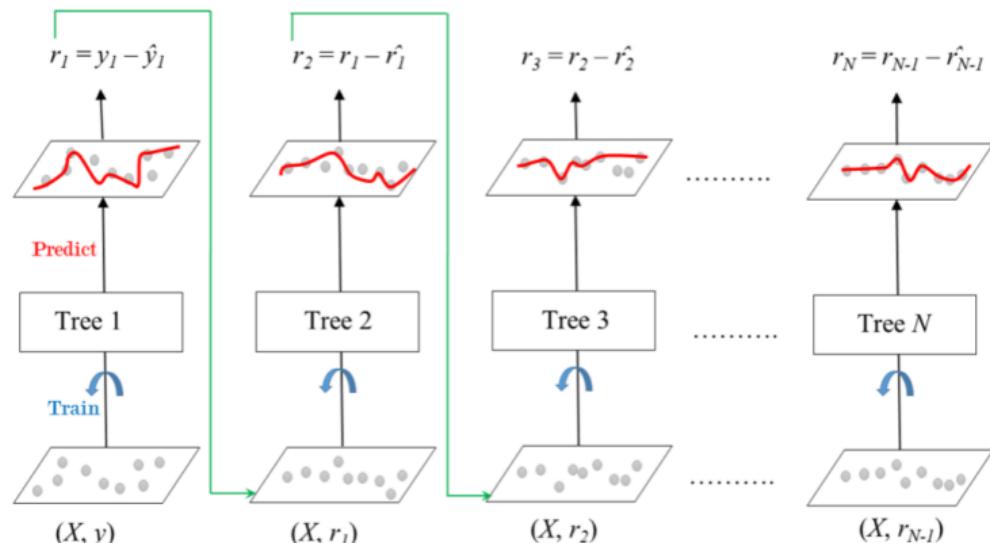
A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

# Gradient Boosting Machines

- Gradient boosting refers to an additive ensemble of trees:



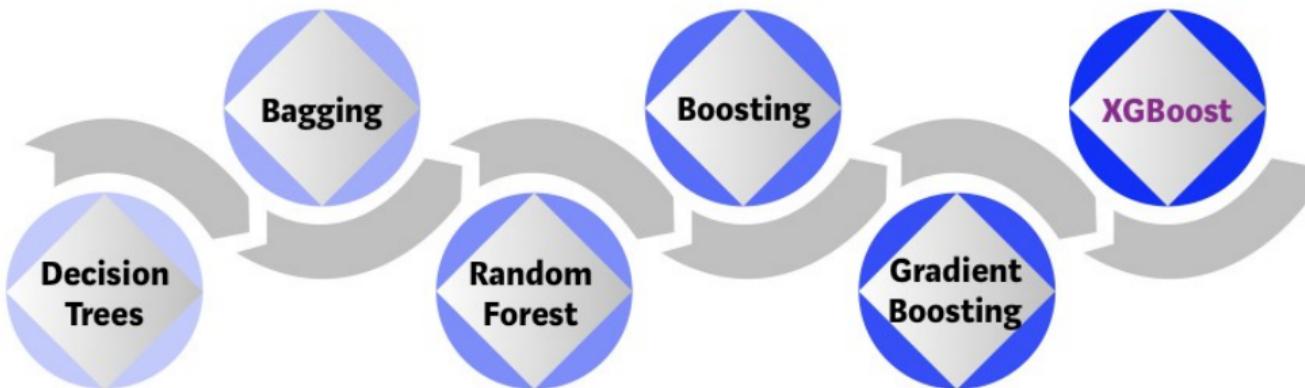
- Adds additional layers of trees to fit the residuals of the first layers

# XGBoost Ingredients

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias



A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

## XGBoost

- ▶ Feurer et al (2018) find that XGBoost beats a sophisticated AutoML procedure with grid search over 15 classifiers and 18 data preprocessors.
- ▶ A good starting point for any machine learning task:
  - ▶ easy to use
  - ▶ actively developed
  - ▶ efficient / parallelizable
  - ▶ provides model explanations
  - ▶ takes sparse matrices as input

## Complicated in Theory, Easy in practice

```
from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(X_train, y_train,
           early_stopping_rounds=10,
           eval_metric="logloss",
           eval_set=[(X_eval, y_eval)])
)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

- ▶ See more detailed examples/explanation in the “XGBoost Explained” reading, and this week’s example code notebook.

## Early Stopping with Validation Set

- ▶ Early stopping is an elegant regularization technique, used in xgboost and in neural nets:
  - ▶ gradually train the model gradients while checking fit in a held-out validation set.
    - ▶ for xgboost, this means adding more layers of trees to fit the residuals.
  - ▶ when model starts overfitting, stop training.

## Early Stopping with Validation Set

- ▶ Early stopping is an elegant regularization technique, used in xgboost and in neural nets:
  - ▶ gradually train the model gradients while checking fit in a held-out validation set.
    - ▶ for xgboost, this means adding more layers of trees to fit the residuals.
    - ▶ when model starts overfitting, stop training.
- ▶ Requires user to specify two additional parameters:
  - ▶ `eval_set` = validation set (separate from training set and test set)
  - ▶ `early_stopping_rounds`: stop training if we have done this many epochs without improving validation set performance.

```
from sklearn.model_selection import train_test_split
X_val_train, X_test, y_val_train, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_val_train, y_val_train)
```

## Early Stopping with Validation Set

- ▶ Early stopping is an elegant regularization technique, used in xgboost and in neural nets:
  - ▶ gradually train the model gradients while checking fit in a held-out validation set.
    - ▶ for xgboost, this means adding more layers of trees to fit the residuals.
  - ▶ when model starts overfitting, stop training.
- ▶ Requires user to specify two additional parameters:
  - ▶ `eval_set` = validation set (separate from training set and test set)
  - ▶ `early_stopping_rounds`: stop training if we have done this many epochs without improving validation set performance.

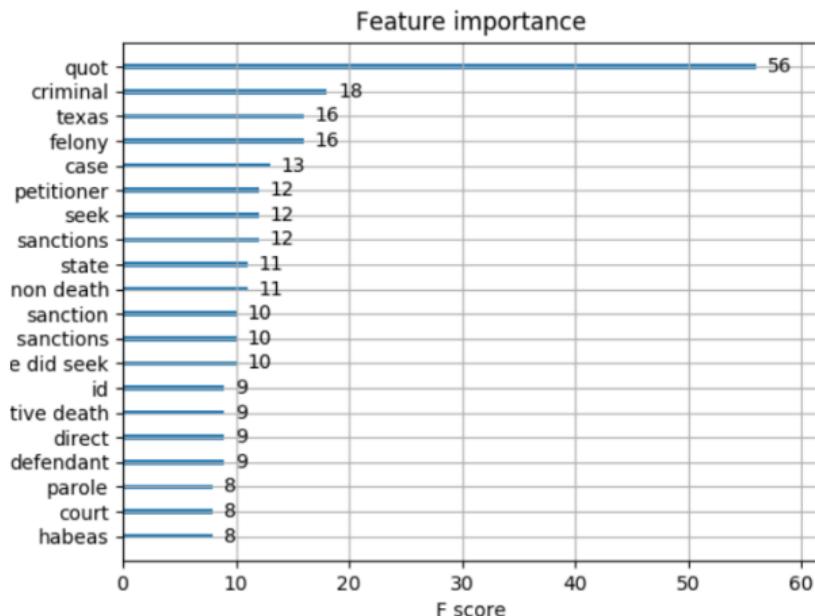
```
from sklearn.model_selection import train_test_split
X_val_train, X_test, y_val_train, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_val_train, y_val_train)

from xgboost import XGBClassifier
xgb = XGBClassifier(eval_set = [X_val, y_val], early_stopping_rounds=10)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

## Feature Importance

```
from xgboost import plot_importance  
plot_importance(xgb_reg, max_num_features=20)
```

<IPython.core.display.Javascript object>



1

- ▶ XGBoost provides a metric of feature importance that summarizes how well each feature contributes to predictive accuracy. (will return to this in week 9)

# Outline

Binary Classifier Metrics

Multi-Class Models

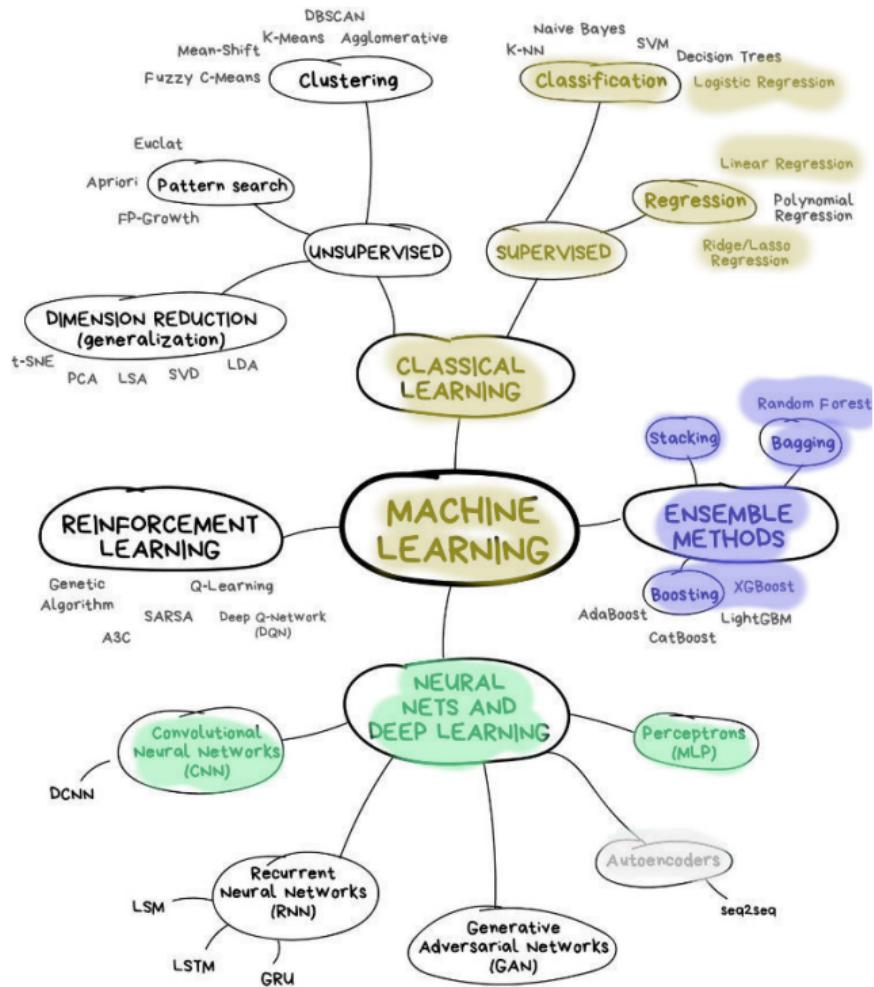
Ensemble Learning with XGBoost

Introduction to Neural Nets

Basics

Regularizing neural nets

Applications



# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

**Introduction to Neural Nets**

Basics

Regularizing neural nets

Applications

- ▶ Neural networks ↔ deep learning models
  - ▶ solve machine learning problems, just like logistic regression or gradient boosted machines
  - ▶ use tensorflow/keras or torch, rather than sklearn or xgboost.

- ▶ Neural networks ↔ deep learning models
  - ▶ solve machine learning problems, just like logistic regression or gradient boosted machines
  - ▶ use tensorflow/keras or torch, rather than sklearn or xgboost.
- ▶ **why use neural nets?**
  - ▶ sometimes outperform standard ML techniques on standard problems
  - ▶ greatly outperform standard ML techniques on some problems, for example image recognition / text generation

- ▶ Neural networks ↔ deep learning models
  - ▶ solve machine learning problems, just like logistic regression or gradient boosted machines
  - ▶ use tensorflow/keras or torch, rather than sklearn or xgboost.
- ▶ **why use neural nets?**
  - ▶ sometimes outperform standard ML techniques on standard problems
  - ▶ greatly outperform standard ML techniques on some problems, for example image recognition / text generation
- ▶ **why not use neural nets?**
  - ▶ usually worse than standard ML on standard problems, and harder to implement.
  - ▶ Computational constraints: Recent models like OpenAI's GPT-3 would take ETH Deep Learning Cluster 18 months to train.

## “Neural Networks”, “Deep Learning”

- ▶ “**Neural**”:
  - ▶ NN's do not work like the brain – such metaphors are misleading.

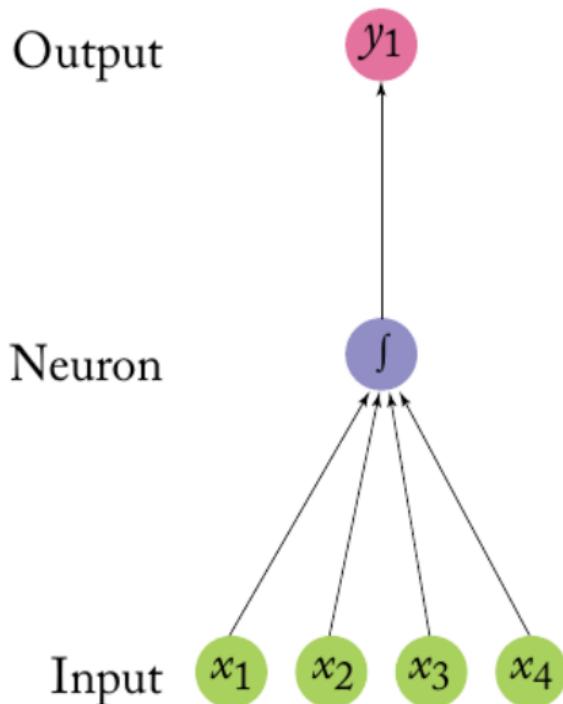
## “Neural Networks”, “Deep Learning”

- ▶ “**Neural**”:
  - ▶ NN’s do not work like the brain – such metaphors are misleading.
- ▶ “**Networks**”:
  - ▶ NNs are not “networks” as that is understood in mathematical network theory or social science.

## “Neural Networks”, “Deep Learning”

- ▶ “**Neural**”:
  - ▶ NN's do not work like the brain – such metaphors are misleading.
- ▶ “**Networks**”:
  - ▶ NNs are not “networks” as that is understood in mathematical network theory or social science.
- ▶ “**Deep**” Learning:
  - ▶ does not speak to profundity or effectiveness.
  - ▶ a banal origin, and a source of hype.

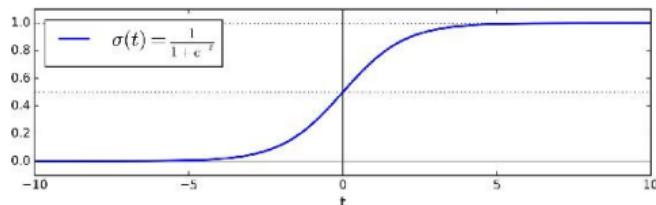
## A “Neuron”



- ▶ applies dot product to vector of numerical inputs:
  - ▶ multiplies each input by a learned weight (parameter or coefficient)
  - ▶ sums these products
- ▶ applies a non-linear “activation function” to the sum
  - ▶ (e.g., the  $\int$  shape indicates a sigmoid transformation)
- ▶ passes the output.

# “Neuron” = Logistic Regression

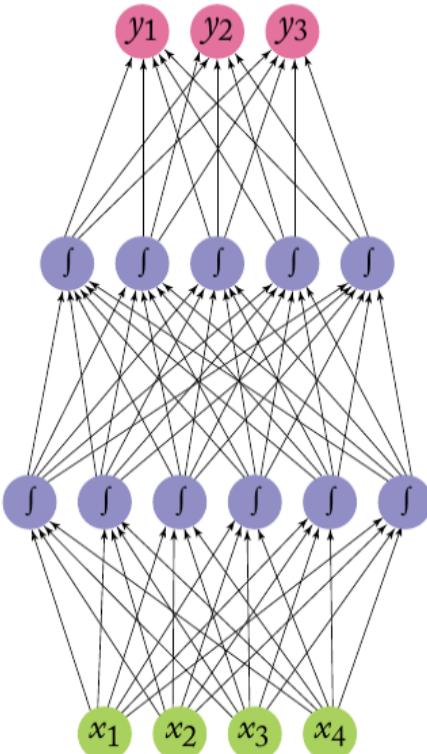
$$\hat{y} = \text{sigmoid}(\mathbf{x} \cdot \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\mathbf{x} \cdot \boldsymbol{\theta})}$$



- ▶ applies dot product to vector of numerical inputs:
  - ▶ multiplies each input by a learned weight (parameter or coefficient)
  - ▶ sums these products
- ▶ applies a non-linear “activation function” (sigmoid) to the sum
- ▶ passes the output.

# Feed-Forward Neural Network (FFN)

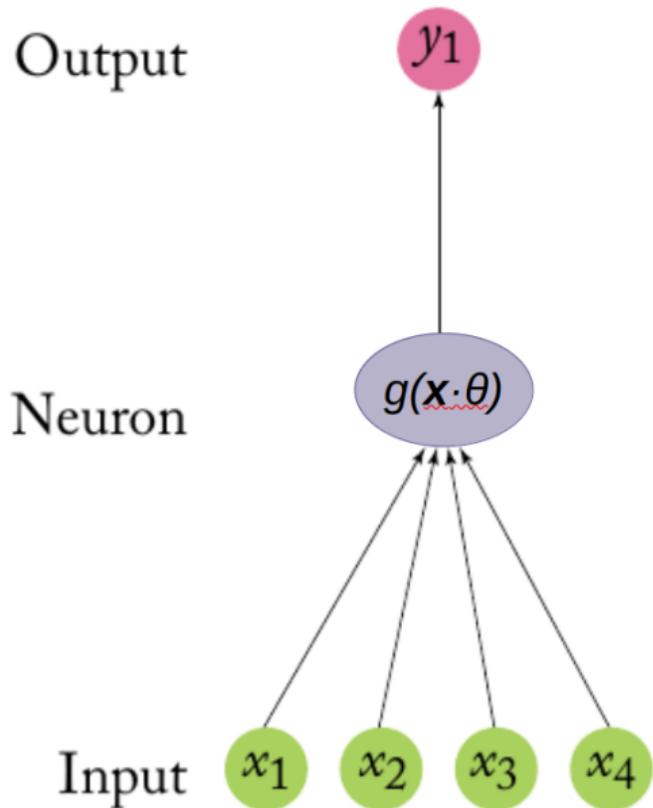
Output layer



- ▶ A feed-forward network (also called a multi-layer perceptron or sequential model) stacks neurons horizontally and vertically.
- ▶ alternatively, think of it as a stacked ensemble of logistic regression models.
- ▶ this vertical stacking is the “deep” in “deep learning”!

- ▶ FFN's are composed of "Dense" layers – means that all neurons are connected.
- ▶ FFN with a single hidden layer, with sigmoid activation, can approximate any continuous function on a closed and bounded subset of  $\mathbb{R}^n$ , and any mapping from one finite discrete space to another finite discrete space (Hornik et al 1989, Cybenko 1989).
  - ▶ But NN would have to be exponentially large in some cases (Telgarsky 2016) .

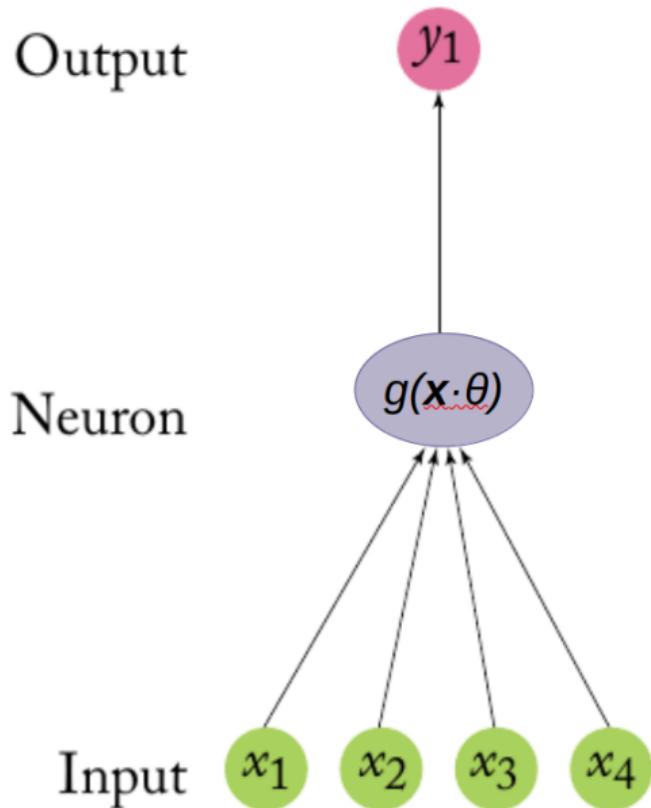
## Activation functions $g(\mathbf{x} \cdot \theta)$



Previously we had

$$g(\mathbf{x} \cdot \theta) = \text{sigmoid}(\mathbf{x} \cdot \theta) = \frac{1}{1 + \exp(-\mathbf{x} \cdot \theta)}$$

## Activation functions $g(\mathbf{x} \cdot \theta)$

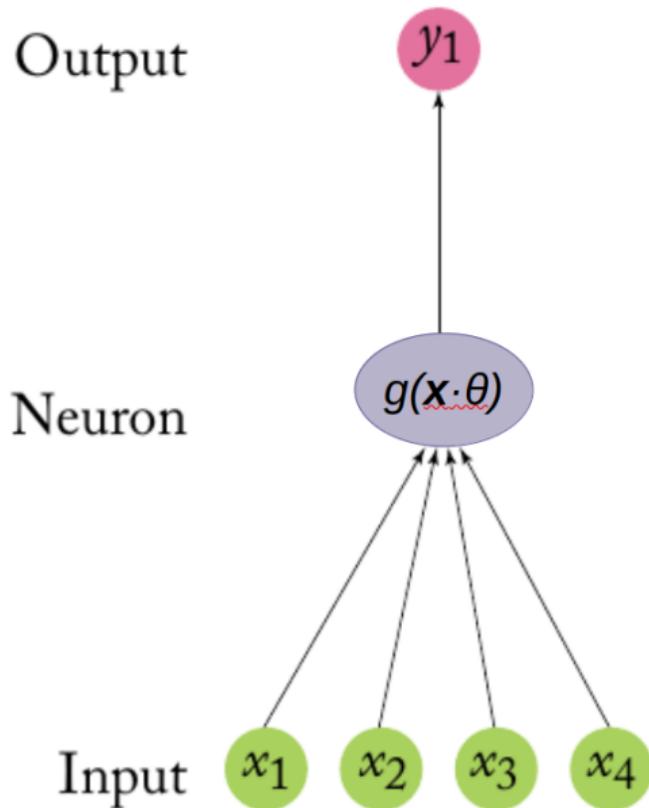


Previously we had

$$g(\mathbf{x} \cdot \theta) = \text{sigmoid}(\mathbf{x} \cdot \theta) = \frac{1}{1 + \exp(-\mathbf{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

## Activation functions $g(\mathbf{x} \cdot \theta)$



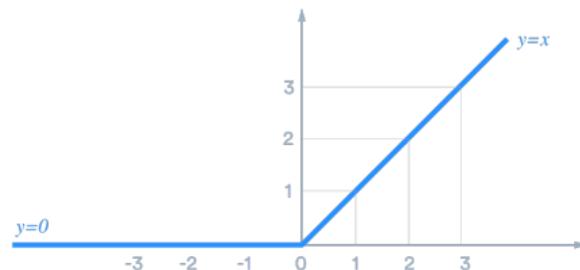
Previously we had

$$g(\mathbf{x} \cdot \theta) = \text{sigmoid}(\mathbf{x} \cdot \theta) = \frac{1}{1 + \exp(-\mathbf{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

**ReLU (rectified linear unit) function:**

$$g(\mathbf{x} \cdot \theta) = \text{ReLU}(\mathbf{x} \cdot \theta) = \max\{0, \mathbf{x} \cdot \theta\}$$



# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

Introduction to Neural Nets

Basics

Regularizing neural nets

Applications

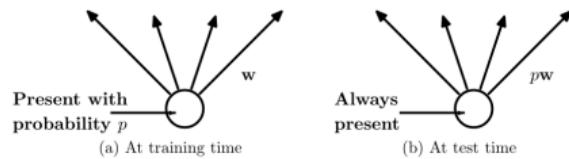
## Early stopping

```
from tensorflow.keras.callbacks import EarlyStopping  
earlystop = EarlyStopping(monitor='loss', patience=3)  
model.fit(x_train, y_train,  
          epochs=100,  
          validation_data=(x_val, y_val)),  
          callbacks=[earlystop])
```

As done with xgboost, a standard regularization approach for NNs is **early stopping**:

- ▶ Split data into three sets: training, validation, and test.
- ▶ stop training when validation-set loss stops improving
- ▶ evaluate model in test set.

# Dropout



Source: Srivastava et al, JMLR 2014

An elegant regularization technique:

- ▶ at every training step, every neuron has some probability (typically  $p = 0.5$ ) of being temporarily dropped out, so that it will be ignored at this step.
- ▶ at test time, neurons don't get dropped anymore but coefficients are down-weighted by  $p$ .

▶ add after dense layers:

```
from tensorflow.keras.layers import Dropout  
model.add(Dropout(.5))
```

## Why Dropout Works

- ▶ Approximates an ensemble of  $N$  models (where  $N$  is the number of neurons).

## Why Dropout Works

- ▶ Approximates an ensemble of  $N$  models (where  $N$  is the number of neurons).
- ▶ Neurons cannot co-adapt with neighboring neurons and must be independently useful.
- ▶ Layers cannot rely excessively on just a few input neurons; they have to pay attention to all input neurons.
  - ▶ Makes the model less sensitive to slight changes in the inputs.

## How to choose among so many options?

- ▶ the # of layers, # of neurons, regularization, dropout, etc are all tunable hyperparameters.
  - ▶ can pick these with cross-validation as we did previously.
- ▶ neural nets have many many dimensions for tuning.
  - ▶ this is a serious downside of neural nets, compared to the standard scikit-learn models.
- ▶ see the Geron book for advice on this point.
  - ▶ in general, make a big model (too many layers, too many neurons) and regularize with dropout/early stopping.

# Outline

Binary Classifier Metrics

Multi-Class Models

Ensemble Learning with XGBoost

Introduction to Neural Nets

Basics

Regularizing neural nets

Applications

## Predicting Mortgage Default with FFNs (Sirignano, Sadhwani, & Giesecke 2018)

- ▶ Analyze mortgage risk using data from over 120 million loans for U.S. borrowers, 1995-2014
- ▶ Estimate deep learning model to predict loan status changes:
  - ▶ current; late; foreclosure
- ▶ Predictors:
  - ▶ loan variables at origination
  - ▶ loan performance variables over time
  - ▶ local economic variables

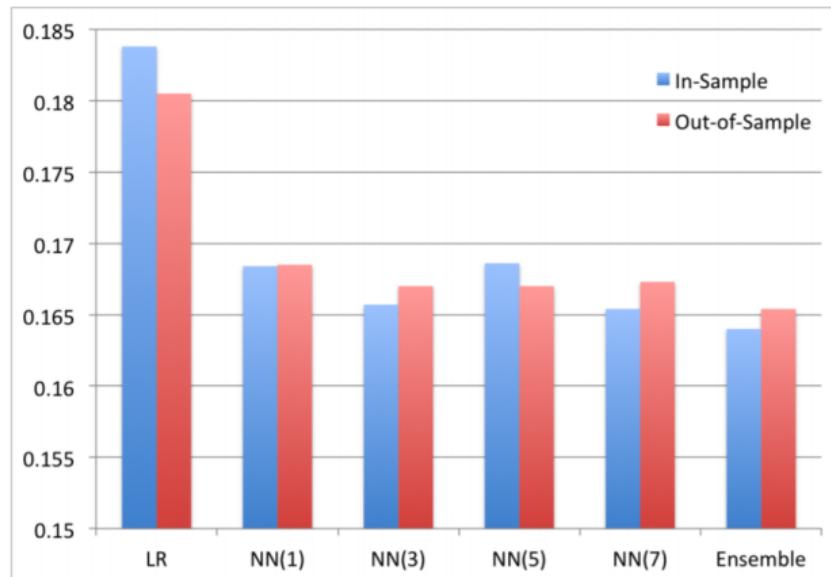
## Monthly Transition Matrix (Outcome)

	Current	30	60	90+	Foreclosure
Current	97	1.4	0	0	.001
30 days	34.6	44.6	19	0	.004
60 days	12	16.8	34.5	34	1.6
90+ days	4.1	1.4	2.6	80.2	10
Foreclosure	1.9	.3	.1	6.8	87

## Modeling

- ▶ Dataset is 350 billion loan-month transitions.
  - ▶ 294 predictors.
- ▶ Feed-forward network:
  - ▶ cross-validation picks 5 layers, ~200 neurons each, ReLU activation.
  - ▶ compare to logistic regression baseline

## In- and out-of-sample errors vs. network depth



## Global variable ranking by “leave-one-out”

Variable	Test Loss
State unemployment rate	1.160
Current outstanding balance	.303
Original interest rate	.233
FICO score	.204
Number of times 60dd in last 12 months	.179
Number of times current in last 12 months	.175
Original loan balance	.175
Total days delinquent $\geq 160$	.171
Lien type = first lien	.171
Original interest rate - national mortgage rate	.170
LTV ratio	.169
Time since origination	.168
Debt-to-income ratio	.168
:	:

## Short Required Readings for Next Week

- ▶ For group discussion (links in syllabus):
  - ▶ “Home office to scrape ‘racist algorithm’ for UK visa applicants”, [bit.ly/UK-visas](http://bit.ly/UK-visas)
  - ▶ “Controversial exams algorithm to set 97% of GCSE results”, [bit.ly/UK-exams](http://bit.ly/UK-exams)