# eXtreme Gradient Boost

**-**

what calculus enthusiasts love the most

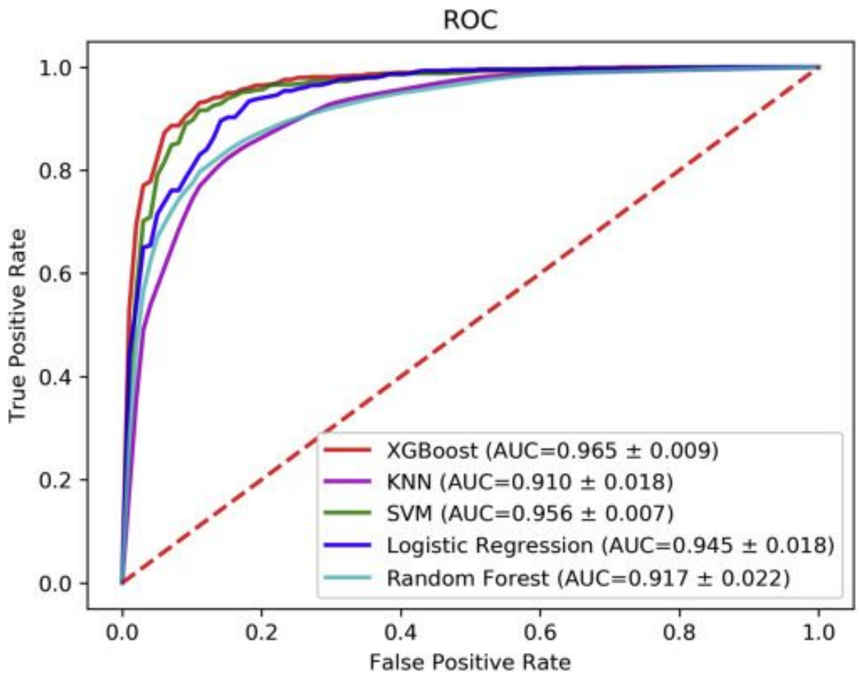
*WhyR??*

# Tianqi Chen

- Github: github
- Twitter: Twitter
- Professor in the Machine Learning Department and Computer Science Department at Carnegie Mellon University.
- Research
    - Apache TVM
    - **XGBoost**
    - Apache MXNet

# What is it?

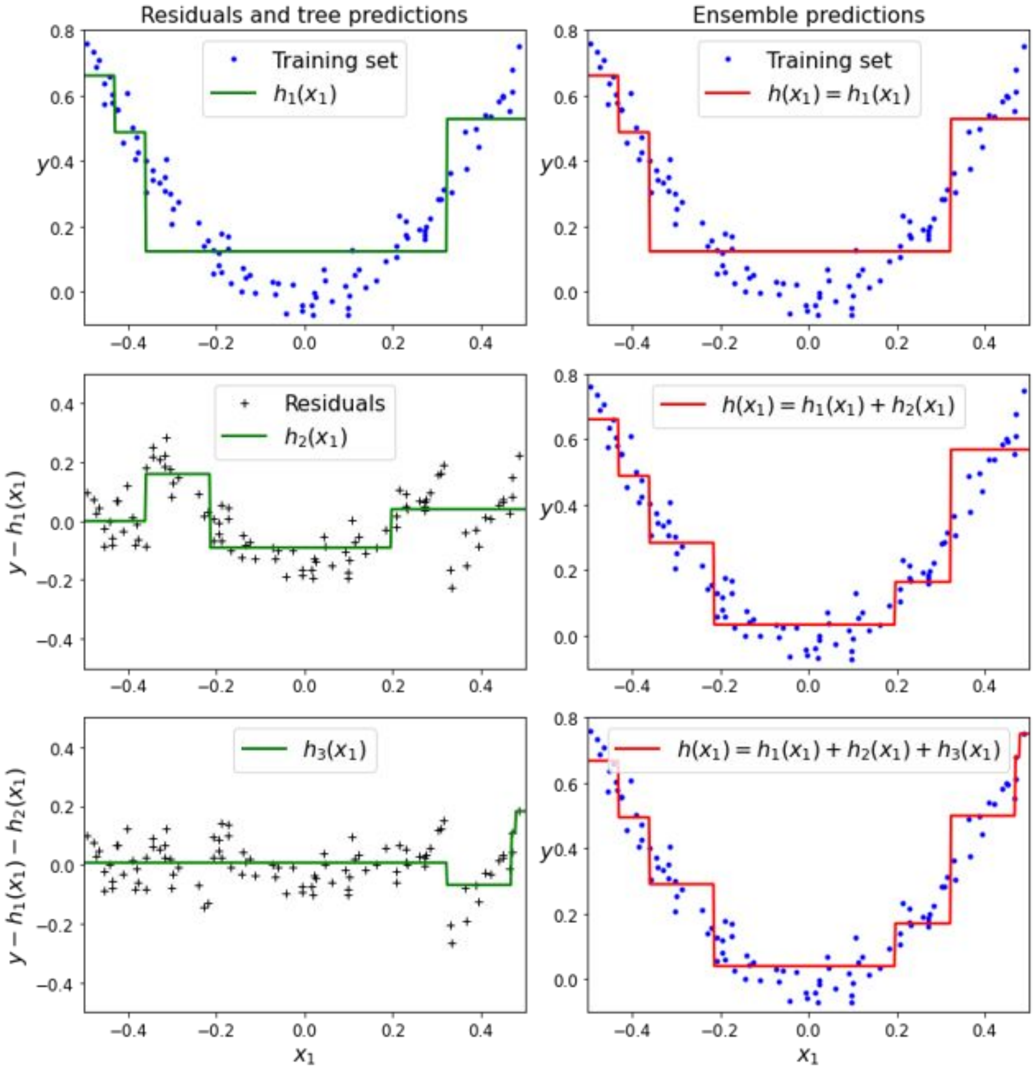
- One of the most liked machine learning algorithms in Kaggle.
- Teams with this algorithm win the competition.
- It can be used for supervised learning tasks such as regression and classification.



# Gradient boosting

# Gradient boosting for regression

```
In [7]: from sklearn.tree import DecisionTreeRegressor

        tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
        tree_reg1.fit(X, y)

Out[7]: DecisionTreeRegressor(max_depth=2, random_state=42)

In [34]: y2 = y - tree_reg1.predict(X)
         tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
         tree_reg2.fit(X, y2)

Out[34]: DecisionTreeRegressor(max_depth=2, random_state=42)

In [35]: y3 = y2 - tree_reg2.predict(X)
         tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
         tree_reg3.fit(X, y3)

Out[35]: DecisionTreeRegressor(max_depth=2, random_state=42)

In [37]: y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



Residuals and tree predictions
Ensemble predictions
Training set
$h_1(x_1)$
$h(x_1) = h_1(x_1)$
Residuals
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
$y$
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
$x_1$

# But what is a relationship between XGBoost and gradient?

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

# XGBoost Parameters

- General parameters:
  - **booster** - possible values:
    - gblinear
    - gbtree
    - gbdart (“Dropouts meet Multiple Additive Regression Trees”)
- Booster parameters
- Learning task parameters:
  - **objective**
- Command line parameters:
  - only used in the CLI version of XGBoost

# Booster parameters

- eta (learning_rate)
- gamma (min_split_loss)
- min_child_weight
- max_depth
- alpha (L2) & lambda (L1)

More info about available parameters can be found in the XGBoost documentation

# Preparing data for XGBoost

- Before applying XGBoost, we have to **convert** all **data** into **numeric** type
  - Label Encoding (e.g. Species Category in Iris dataset)

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|---|
| 1 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 80 | 80 | 5.7 | 2.6 | 3.5 | 1.0 | Iris-versicolor |
| 150 | 150 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species_encoded |
|---|---|---|---|---|---|---|
| 1 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 80 | 80 | 5.7 | 2.6 | 3.5 | 1.0 | 1 |
| 150 | 150 | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

# Preparing data for XGBoost

- Before applying XGBoost, we have to convert all data into numeric type
  - Label Encoding - code

```
df <- data.frame(read_csv('Iris.csv'))
categories <- df[['Species']]
lbl <- LabelEncoder$new()
lbl$fit(categories)
encoded <- lbl$transform(categories)
df_encoded <- data.frame(df[, 1:length(df)-1], encoded)
```

# Preparing data for XGBoost

- Before applying XGBoost, we have to convert all data into numeric type
  - Label Encoding
  - One Hot Encoding (e.g. in dataset regarding breast cancer cases)

| | X.40.49. | X.premeno. | X.15.19. | X.0.2. | X.yes. | X.3. | X.right. | X.left_up. | X.no. | X.recurrence.events. |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | '50-59' | 'ge40' | '15-19' | '0-2' | 'no' | '1' | 'right' | 'central' | 'no' | 'no-recurrence-events' |
| 2 | '50-59' | 'ge40' | '35-39' | '0-2' | 'no' | '2' | 'left' | 'left_low' | 'no' | 'recurrence-events' |
| 3 | '40-49' | 'premeno' | '35-39' | '0-2' | 'yes' | '3' | 'right' | 'left_low' | 'yes' | 'no-recurrence-events' |

| | X.3..2. | X.3..3. | X.right..left. | X.right..right. | X.left_up..central. | X.left_up..left_low. |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 | 1 |

| | X.left_up..left_up. | X.left_up..right_low. | X.left_up..right_up. | X.left_up.nan | X.no..no. |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 |

| | X.no..yes. | X.recurrence.events..no.recurrence.events. | X.recurrence.events..recurrence.events. |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 |
| 3 | 1 | 1 | 0 |

# Preparing data for XGBoost

- Before applying XGBoost, we have to convert all data into numeric type
  - Label Encoding
  - One Hot Encoding - code

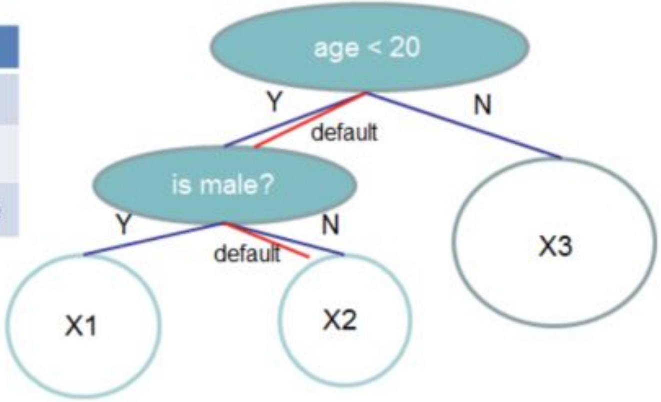
```r
# One Hot Encoding Breast Cancer
library(caret)

df2 <- data.frame(read_csv('breast-cancer.csv'))

dummy <- dummyVars(" ~ .", data=df2)
final_df2 <- data.frame(predict(dummy, newdata=df2))
```

# Preparing data for XGBoost

- Before applying XGBoost, we have to convert all data into numeric type
- Missing values **are dealt with automatically** by XGBoost during model training

**Data**

| Example | Age | Gender |
|---|---|---|
| X1 | ? | male |
| X2 | 15 | ? |
| X3 | 25 | female |



# Preparing data for XGBoost

- Before applying XGBoost, we have to convert all data into numeric type
- Missing values are dealt with automatically by XGBoost during model training
- XGBoost works with data in ***matrix*** type - ***DataFrame*** type is not supported
    - In order to convert Data Frame to a matrix, you can use this R function:

```
# Converting Data Frame to a Numeric Matrix

data.matrix(df)
```

# xgboost package

since 2016

90K download per month

1. Can solve regression, classification and ranking problems,
2. Two solvers are included: linear model and tree based model.
3. Can automatically do parallel computation on Windows and Linux.

# Package usage

**INSTALLATION**
install.packages(‘xgboost’)

**library(xgboost)**

*# zbiór z pakietu xgboost*
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test

**CLASSIFICATION**

bstSparse <- **xgboost**(data = train$data, label = train$label,
max.depth = 2, eta = 1, nthread = 2, nrounds = 2,
**objective = "binary:logistic"**)
pred <- **predict**(bstSparse, test$data)
**as.numeric(head(pred > 0.5))**

**REGRESSION**

bstSparse <- **xgboost**(data = train$data, label = train$label,
max.depth = 2, eta = 1, nthread = 2, nrounds = 2,
**objective = "reg:squarederror"**)
pred <- **predict**(bstSparse, test$data)

# Related literature


1. Short documentation with basic usage.
2. Detailed documentation.
3. Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." 2016.
4. Machine learning with XGBoost tutorial.

# Bibliography


https://www.doit-intl.com/xgboost-or-tensorflow/

https://towardsdatascience.com/xgboost-is-not-black-magic-56ca013144b4

https://xgboost.readthedocs.io/en/stable/faq.html

https://xgboost.readthedocs.io/en/stable/parameter.html

http://www.chengli.io/tutorials/gradient_boosting.pdf