Machine Learning Graz

Graz University of Technology

A theoretical introduction to Boosting

Anand Subramoney

anand [at] igi.tugraz.at

Institute for Theoretical Computer Science, TU Graz

http://www.igi.tugraz.at/

Machine Learning Graz Meetup

21th November 2018

Motivation

- A gambler wants an algorithm to accurately predict the winner of a horse race based on some features:
  - the no. of races recently won by each horse
  - Betting odds for each horse etc.
- He asks an expert to write down various rules-of-thumb for each set of races for which data is available E.g.
  - “Bet on the horse that has recently won the most races”
  - “Bet on the horse with the most favored odds”
- Each rule-of-thumb is crude and inaccurate, but does slightly better than chance
- There are two problems faced by the gambler:
  - How should s/he choose the data presented to the expert to extract the rules-of-thumb that will be the most useful?
  - How can s/he combine the many rules-of-thumb he has collected into a single highly accurate prediction?

[Freund & Schapire 1996]

Boosting

- Boosting refers to this general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb
- The rules-of-thumb are called “weak learners/hypotheses”
  - is defined to be a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing)
- We want to construct a “strong learner”
  - a strong learner is a classifier that is arbitrarily well-correlated with the true classification.
- [Kearns and Valiant 1988]: Is this even possible?
- [Schapire 1990]: Yes it is possible!

AdaBoost

AdaBoost (Adaptive Boosting)

- Considers the task of binary classification.
- The general algorithm is the following:

Do for many weak learners:

1. Train a new weak learner with the set of training points weighted according to some weights
2. Calculate the error of this weak learner
3. Update the weights based on this error
4. Train a new weak learner with the new set of weights

Final prediction is the weighted sum of the predictions of each weak learner

AdaBoost algorithm

Input

- A sequence of N labelled examples $< (x_1, y_1), \ldots, (x_N, y_N) >$
- Distribution D over the N examples
- Weak learning algorithm **WeakLearn**
- Integer T specifying number of iterations

[Freund & Schapire 1996]

AdaBoost algorithm

Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, \dots, N$

Do for $t = 1,2, \dots, T$

1. Set $\boldsymbol{p}^t = \frac{\boldsymbol{w}^t}{\sum_{i=1}^N w_i^t}$ (Normalized weights)
2. Call **WeakLearn** providing it with the distribution $\boldsymbol{p}^t$; (Data weighted by $\boldsymbol{p}^t$ when learning) get back hypothesis $h_t: X \to [0,1]$
3. Calculate the error of h_t: $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|$.
4. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
5. Set the new weights vector to be:

$$w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i)-y_i|}$$ (Increase weight for “hard” points)

AdaBoost algorithm

Output the hypothesis:

$$h_f(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{T} \left(\log \frac{1}{\beta_t}\right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log \frac{1}{\beta_t} \\ 0 & \text{otherwise} \end{cases}$$

i.e it uses the weighted sum of the weak hypotheses

(weighted by $\log \frac{1}{\beta_t}$: greater weight is given to hypothesis with lower error)

Theoretical Guarantees

- Can achieve arbitrary accuracy
- More specifically:

  the error of the final hypothesis (with respect to the given set of examples) is bounded by:

  $$\exp(-2\sum_{t=1}^{T}\gamma_t^2)$$

  where $\epsilon_t = \frac{1}{2} - \gamma_t$ is the error of the tth weak hypothesis

  γ_t measures the accuracy of the tth weak hypothesis relative to random guessing
- The *training* error of the final hypothesis drops exponentially fast with more weak classifiers
- The accuracy of the final hypothesis improves when any of the weak hypotheses is improved.
- If the weak hypotheses are “simple” and T is “not too large”, then *test* error is also theoretically bounded

Generalization

Goal of learning or optimization

- A sequence of N labelled examples

$$< (x_1, y_1), \ldots, (x_N, y_N) > \sim < \mathrm{X}, \mathrm{Y} >$$

(the examples are samples from the full domain of X, Y)

- Task is to learn

an **estimate** or **approximation** $\hat{F}$

of a **true unknown** function $F^*: X \to Y$

that minimizes some loss function $L\big(y, F(\boldsymbol{x})\big)$

over all joint distribution of all $(y, \boldsymbol{x})$-values

$$F^* = \mathrm{argmin}_F E_{y,\boldsymbol{x}} L\big(y, F(\boldsymbol{x})\big)$$

[Friedman 2001]

P

Numerical optimization in parameter space

- Restrict $F(\boldsymbol{x})$ to be a member of a **parameterized** class of functions $F(\boldsymbol{x}; \boldsymbol{P})$ where $\boldsymbol{P} = \{P_1, P_2, \dots\}$ is a finite set of parameters

- Then we transform our task to a parameter optimization problem:

$$\boldsymbol{P}^* = \operatorname{argmin}_P \Phi(\boldsymbol{P})$$

$$\Phi(\boldsymbol{P}) = E_{y,\boldsymbol{x}} L\big(y, F(\boldsymbol{x}; \boldsymbol{P})\big)$$

$$F^*(\boldsymbol{x}) = F(\boldsymbol{x}; \boldsymbol{P}^*)$$

- Can be solved with gradient descent!

P

Gradient Descent in parameter space

- Start with some initial guess $\boldsymbol{p}_0$

$$\boldsymbol{P}^* = \sum_{m=0}^{M} \boldsymbol{p}_m$$

$$\boldsymbol{p}_m = -\rho_m \, \boldsymbol{g}_m$$

$$\boldsymbol{g}_m = \{\mathrm{g_{jm}}\} = \left[\frac{\partial \Phi(\boldsymbol{P})}{\partial P_j}\right]_{\boldsymbol{P}=\boldsymbol{P}_{m-1}}$$

$\boldsymbol{g}_m$ is the gradient

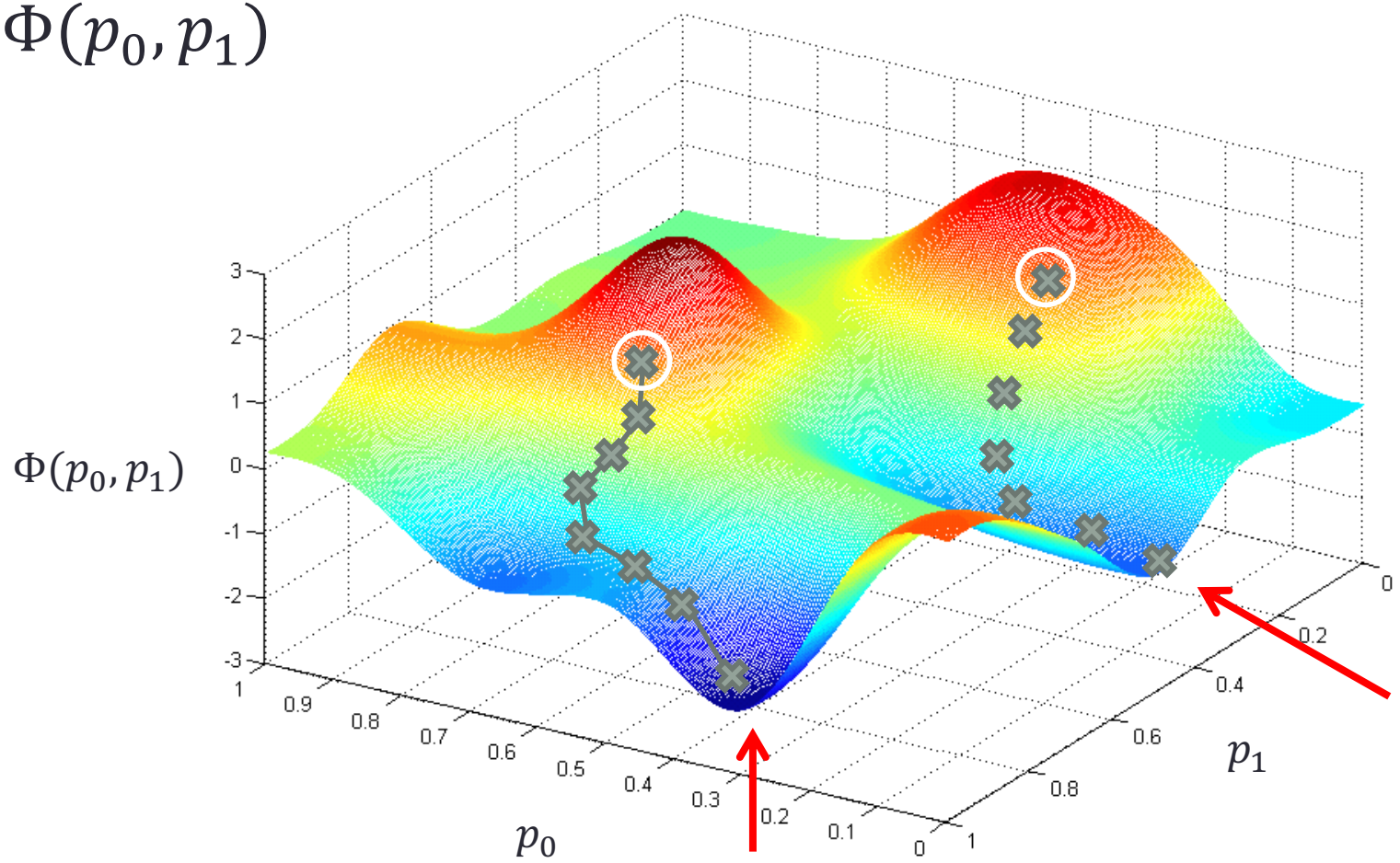
- Equivalent to:

$$\boldsymbol{P}_m = \boldsymbol{P}_{m-1} - \rho_m \, \boldsymbol{g}_m$$

$$\boldsymbol{P}_{m-1} = \sum_{i=0}^{m-1} \boldsymbol{p}_i$$

Gradient Descent

Gradient descent on $\Phi(p_0, p_1)$



F

Numerical optimization in function space

- $F(\boldsymbol{x})$ itself is the parameter!
- Not parametric

- Then we transform our task to a parameter optimization problem:

$$\Phi(F) = E_{y,\boldsymbol{x}} L\big(y, F(\boldsymbol{x})\big)$$

$$\phi\big(F(\boldsymbol{x})\big) = E_y\big[L\big(y, F(\boldsymbol{x})\big)\big|\boldsymbol{x}\big]$$

- Can also be solved with gradient descent!

P

Numerical optimization in parameter space

- Restrict $F(\boldsymbol{x})$ to be a member of a **parameterized** class of functions $F(\boldsymbol{x}; \boldsymbol{P})$ where $\boldsymbol{P} = \{P_1, P_2, \dots\}$ is a finite set of parameters

- Then we transform our task to a parameter optimization problem:

$$\boldsymbol{P}^* = \operatorname{argmin}_P \Phi(\boldsymbol{P})$$

$$\Phi(\boldsymbol{P}) = E_{y,\boldsymbol{x}} L\big(y, F(\boldsymbol{x}; \boldsymbol{P})\big)$$

$$F^*(\boldsymbol{x}) = F(\boldsymbol{x}; \boldsymbol{P}^*)$$

- Can be solved with gradient descent!

F

Gradient Descent in function space

- Start with some initial guess $\boldsymbol{p}_0$

$$F^*(\boldsymbol{x}) = \sum_{m=0}^{M} f_m(\boldsymbol{x})$$

$$f_m(\boldsymbol{x}) = -\rho_m \, g_m(\boldsymbol{x})$$

$$g_m(\boldsymbol{x}) = \left[\frac{\partial \phi(F(\boldsymbol{x}))}{\partial F(\boldsymbol{x})}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$$

$g_m$is the gradient

- Equivalent to:

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) - \rho_m \, g_m(\boldsymbol{x})$$

$$F_{m-1} = \sum_{i=0}^{m-1} f_i(\boldsymbol{x})$$

P

Gradient Descent in parameter space

- Start with some initial guess $\boldsymbol{p}_0$

$$\boldsymbol{P}^* = \sum_{m=0}^{M} \boldsymbol{p}_m$$

$$\boldsymbol{p}_m = -\rho_m \, \boldsymbol{g}_m$$

$$\boldsymbol{g}_m = \{\mathrm{g_{jm}}\} = \left[\frac{\partial \Phi(\boldsymbol{P})}{\partial P_j}\right]_{\boldsymbol{P}=\boldsymbol{P}_{m-1}}$$

$\boldsymbol{g}_m$ is the gradient

- Equivalent to:

$$\boldsymbol{P}_m = \boldsymbol{P}_{m-1} - \rho_m \, \boldsymbol{g}_m$$

$$\boldsymbol{P}_{m-1} = \sum_{i=0}^{m-1} \boldsymbol{p}_i$$

Finite data

- $(y, \boldsymbol{x})$ is estimated by a finite data sample $\{y_i, \boldsymbol{x}_i\}_1^N$
- $E_y[.|\boldsymbol{x}]$ cannot be estimated accurately by its data value at each $\boldsymbol{x}_i$
- Also would like to estimate $F^*(\boldsymbol{x})$ at points outside these data points

- One solution: Assume a parametrized form:

$$F(x; \{\beta_m, \boldsymbol{a}_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(\boldsymbol{x}; \boldsymbol{a}_m)$$

- $h(x; a)$ is the "weak learner" or "base learner"
  - E.g. could be a decision tree, or a linear function or a neural network
  - Can perform classification or regression

Finite data and optimization in parametrized space

F

- Estimate best parameters at each stage m using the given data points

$$(\beta_m, \boldsymbol{a}_m) = \text{argmin}_{\beta,\boldsymbol{a}} \sum_{i=1}^{N} L\big(y_i, F_{m-1}(\boldsymbol{x}_i) + \beta h(\boldsymbol{x}_i; \boldsymbol{a})\big)$$

  - Possibly by gradient descent or any other optimization method
- Then update your total estimator

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \beta_m h(\boldsymbol{x}; \boldsymbol{a}_m)$$

- This is a “greedy stagewise” strategy
- Exactly equivalent to AdaBoost if $L(y, F) = e^{-yF}$!

Finite data and optimization in parametrized space

F

- Estimate best parameters at each stage m using the given data points

$$(\beta_m, \boldsymbol{a}_m) = \text{argmin}_{\beta,\boldsymbol{a}} \sum_{i=1}^{N} L\big(y_i, F_{m-1}(\boldsymbol{x}_i) + \beta h(\boldsymbol{x}_i; \boldsymbol{a})\big)$$

  - Possibly by **gradient descent** or any other optimization method
- Then update your total estimator

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \beta_m h(\boldsymbol{x}; \boldsymbol{a}_m)$$

- This is a “greedy stagewise” strategy
- Exactly equivalent to AdaBoost if $L(y, F) = e^{-yF}$!

F

Gradient descent

$$-g_m(\boldsymbol{x}_i) = -\left[\frac{\partial L\left(y_i, F(\boldsymbol{x}_i)\right)}{\partial F(\boldsymbol{x}_i)}\right]_{\mathbf{F(x)=F_{m-1}(x)}}$$

- This N-dimensional gradient $-\boldsymbol{g}_m = \{-g_m(\boldsymbol{x}_i)\}_1^N$ is only defined at the data points $\{\boldsymbol{x}_i\}_1^N$ and cannot generalize to other data values
- So find the $h(x; a)$ that is "most correlated" with $-g_m(x)$ over the data distribution
- "Most correlated" defined as low mean squared error between gradient $-\boldsymbol{g}_m$ and $\beta h(\boldsymbol{x}_i; \boldsymbol{a})$

F

Gradient boosting

$$\boldsymbol{a}_m = \text{argmin}_{\boldsymbol{a},\beta} \sum_{i=1}^{N} [-g_m(\boldsymbol{x}_i) - \beta h(\boldsymbol{x}_i; \boldsymbol{a})]^2$$ **:MSE**

$$\rho_m = \text{argmin}_{\rho} \sum_{i=1}^{N} L\big(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m)\big)$$

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$$

F

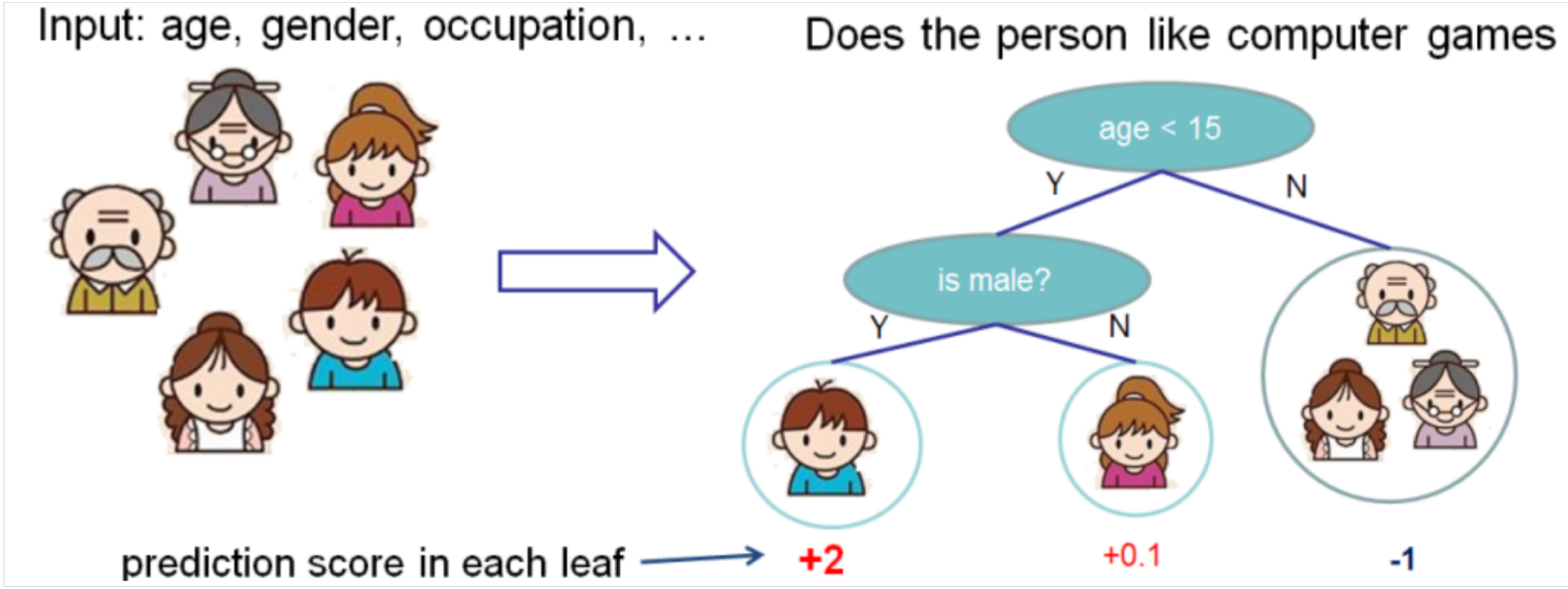
Gradient boosting algorithm

1. $F_0(x) = \text{argmin}_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For m $= 1$ to M do
3. $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$, i $= 1, \ldots,$ N
4. $\boldsymbol{a}_m = \text{argmin}_{\boldsymbol{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\boldsymbol{x}_i; \boldsymbol{a})]^2$
5. $\rho_m = \text{argmin}_\rho \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m)\right)$
6. $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$
7. endFor

Gradient tree boosting

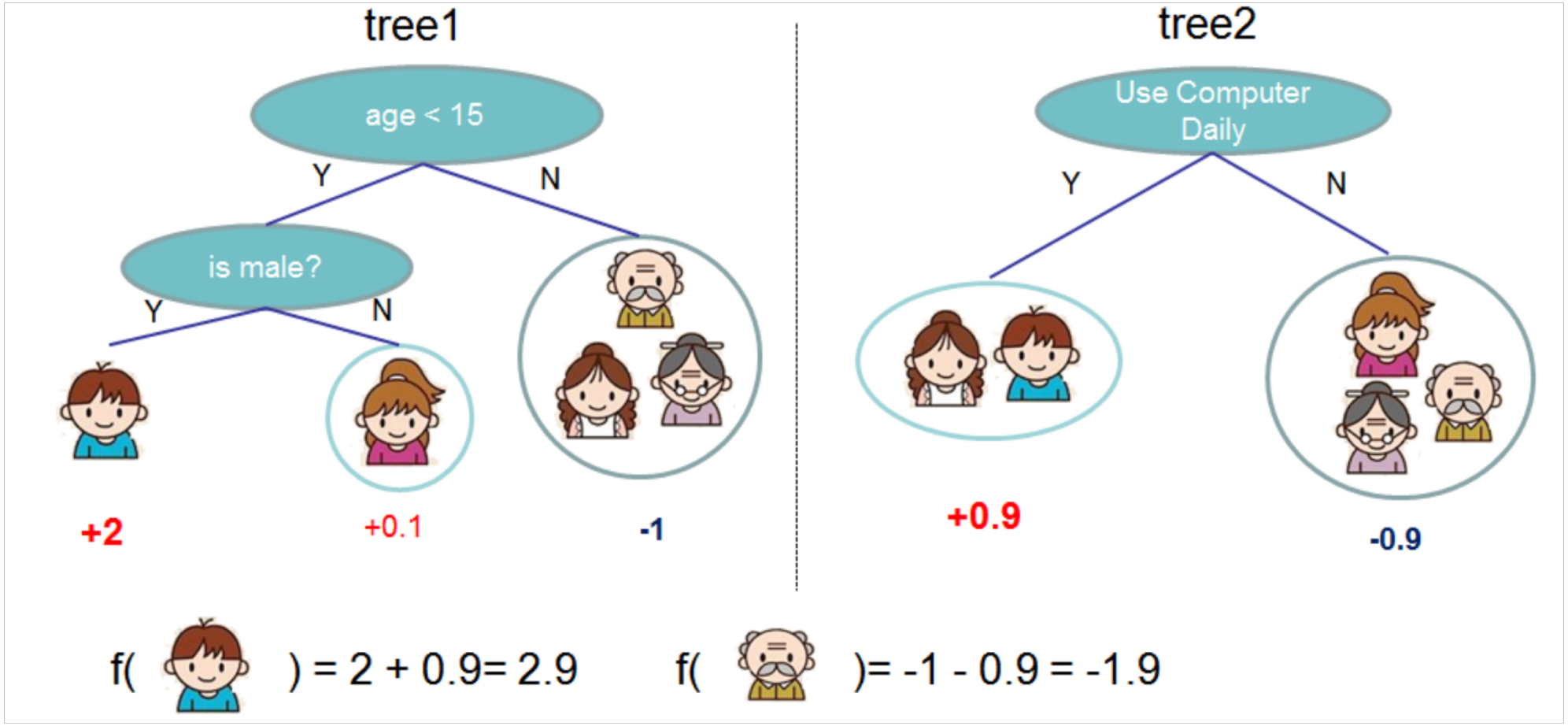
Gradient tree boosting

- Can use decision trees as weak learners (Gradient Tree Boosting)



https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Gradient tree boosting



References

- [Freund & Schapire 1996] : Freund, Y., & Schapire, R. (1996). Experiments with a New Boosting Algorithm (pp. 148–156). Presented at the International Conference on Machine Learning. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.3868
- [Friedman 2001] : Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, *29*(5), 1189–1232.
- Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi (Ed.), *Computational Learning Theory* (pp. 23–37). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-59119-2_166
- Mason, L., Baxter, J., Bartlett, P. L., & Frean, M. R. (2000). Boosting Algorithms as Gradient Descent. In S. A. Solla, T. K. Leen, & K. Müller (Eds.), *Advances in Neural Information Processing Systems 12* (pp. 512–518). MIT Press. Retrieved from http://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent.pdf

Machine Learning Graz

Gradient Boosting

21-Nov-2018

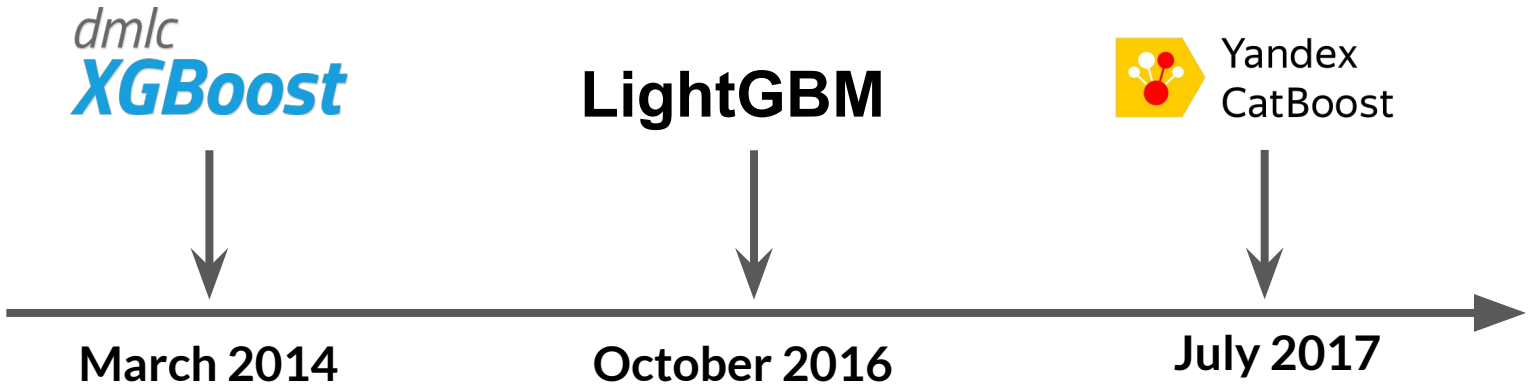
Adrian Spataru
Data Scientist at Know-Center
adrian@spataru.at

Anand Subramoney
Researcher at TU Graz
anand@igi.tugraz.at

Outline

- Available Implementations
- How to run
- Kaggle Case Studies
- Benchmark
- Random Forest

Show Timeline of Released Libraries

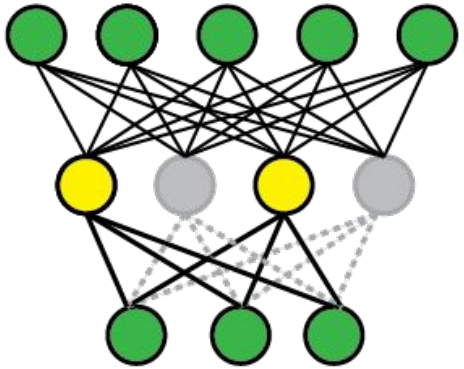


XGBOOST

- Has GBT and Linear models
- L1 and L2 Regularization
- Handling sparse data
- Parallel learning + GPU
- Out-of-core Computing
- Continuous Training
- Wins in Kaggle Competition

Dropout Additive Regression Trees

- Inspired from Neural Network Dropouts
- A method for pruning tree to avoid overfitting
- Trees added early are significant
- Trees added late are likely unimportant
- Next tree built from the residual of **a sample** of previous trees.



Training/Predicting

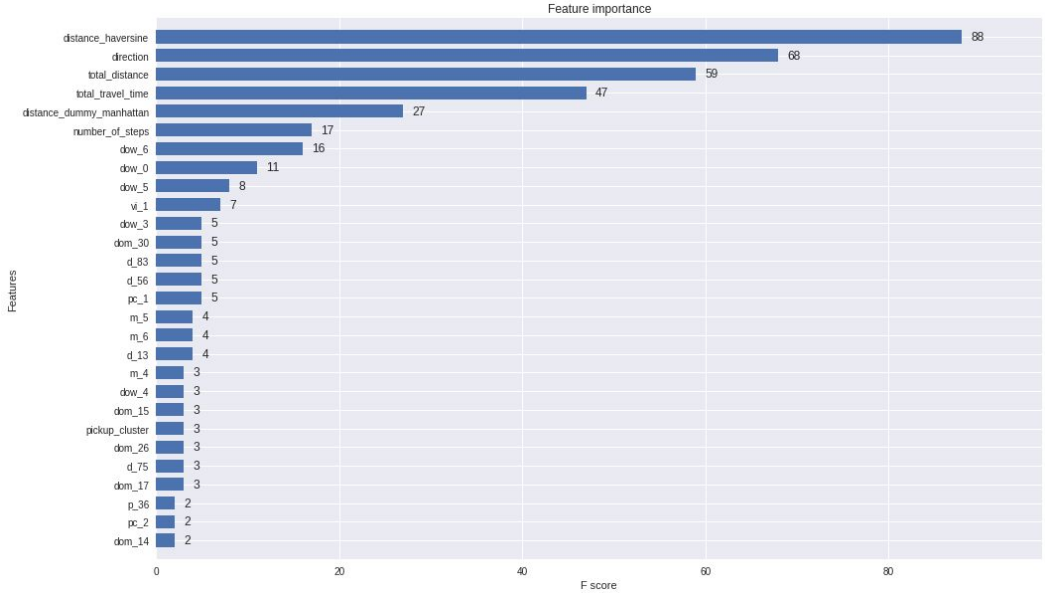
```
from xgboost import XGBClassifier


model = XGBClassifier()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

Feature Importance

```
xgb.plot_importance(model, max_num_features=28)
```



Controlling Overfitting

- max_depth - Depth of the tree.
  - The bigger the bigger the likehood of overfitting.
- eta - The learning rate.
  - Lower the better. However needs more iterations.
- gamma - minimum loss reduction threshold.
  - A node is split only when the resulting split gives a positive reduction in the loss function.
- min_child_weight - stop splitting once your sample size in a node goes below a given threshold
  - Too high, leads to underfitting.

If you can, just GridSearch/Bayesian Optimization

Mercedes-Benz Greener Manufacturing

- Goal: Based on car features ->predict the time it takes to pass testing.
- Around 400 Features
- Winning Solution used a blend of 2 XGBOOST models

LightGBM

- uses Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value
- uses Exclusive Feature Bundling - reduces complexity when using categorical data.
- Faster Training
- Low Memory Usage
- GPU and Parallel Learning Supported

Gradient-based One-Side Sampling (GOSS)

- Reduce the number of data instances
- While keeping the accuracy for learned decision tree
- Keep all instances with large gradients
- Perform random sampling on instances

Gradient-based One-Side Sampling (GOSS)

Row id	Gradients
4	-9
3	5
2	0.3
6	0.2
5	0.1
1	-0.2

Gradient-based One-Side Sampling (GOSS)

Row id	Gradients
4	-9
3	5
2	0.3
6	0.2
5	0.1
1	-0.2

Row id	Gradients	Weights
4	-9	1
3	5	1
2	0.3	2
1	-0.2	2

Training

```
import lightgbm as lgb

train_data = lightgbm.Dataset(x, label=y, categorical_feature=categorical_features)
test_data = lightgbm.Dataset(x_test, label=y_test)

parameters = {
    'application': 'binary',
    'objective': 'binary',
    'metric': 'auc',
    'is_unbalance': 'true',
}


model = lightgbm.train(parameters, train_data, valid_sets=test_data)
```

Porto Seguro’s Safe Driver Prediction

- Predict if a driver will file an insurance claim next year.
- A blend of 1 LGBM and several NN.

CATBOOST

- Deals with categorical data out of the box.
- Fast GPU and multi-GPU support for training
- Data Visualization tools included
- Overfit Detector

CATBOOST - Categorical Algorithm

- If the column has only 2 categories, one hot encoding is used
- Else the categorical column is converted to numerical column
- How? **Target statistics**
- Idea: Replace the value with the expected target variable given the category.

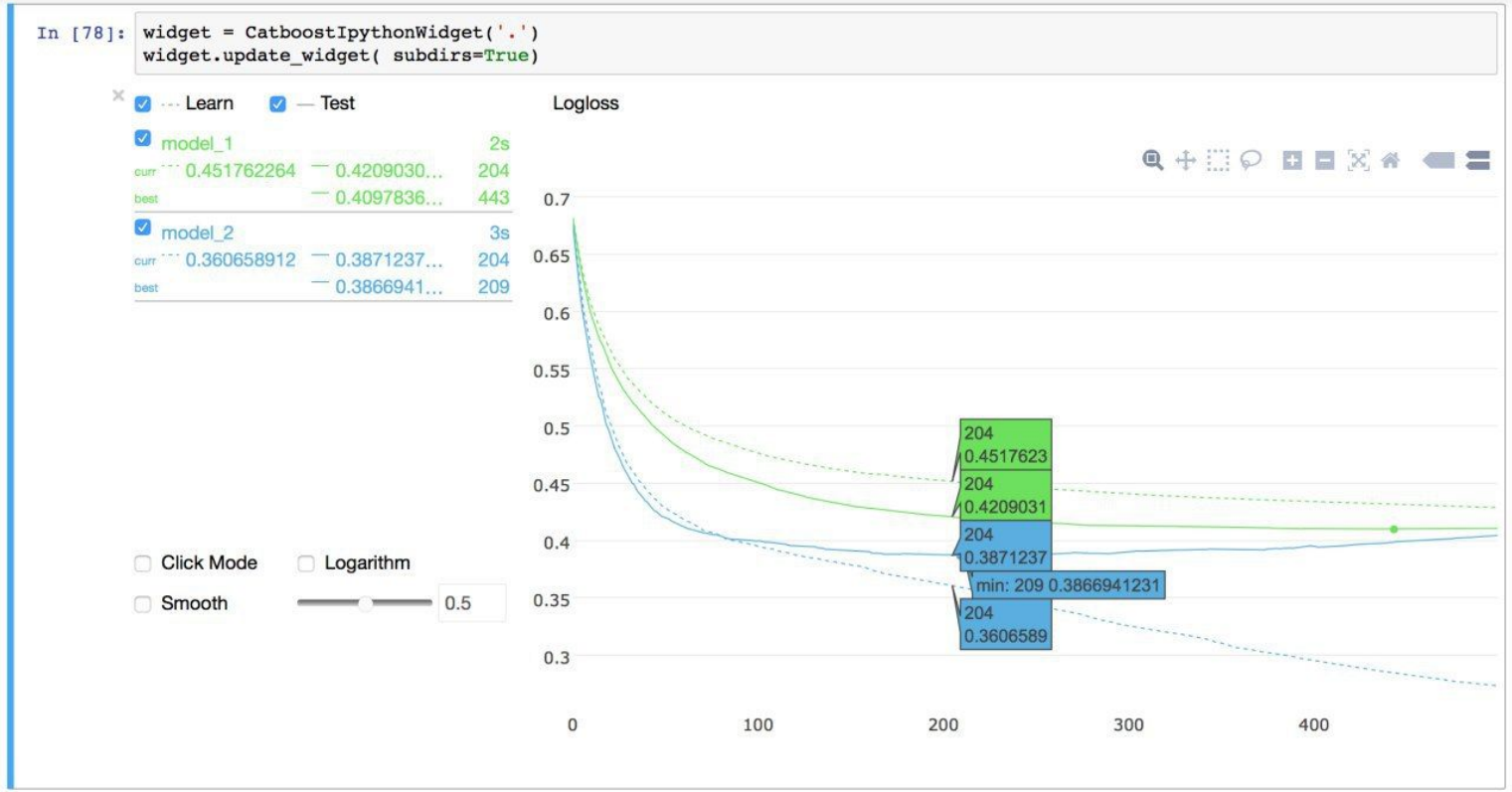
Training..

```
from catboost import CatBoostClassifier

model = CatBoostClassifier(loss_function='Logloss')
model.fit(X_train, y_train, cat_features)
```

```
y_test = model.predict(X_test)
```

CATBOOST VIEWER



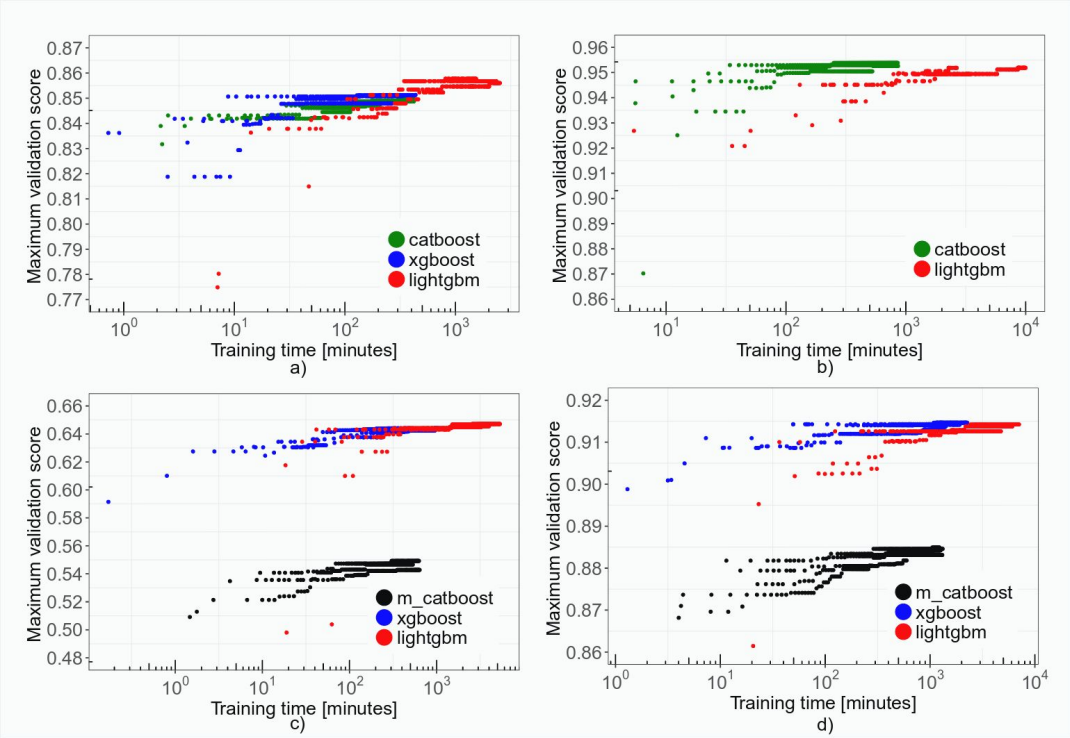
Ubaar Competition

- Ubaar is a trucking platform.
- Predict transport costs based on transported loads.
- Objective MAPE (Mean absolute percentage error)
- Bagged results of 30 LightGBM runs - 15.03
- Bagged results of 30 CatBoost runs - 15.00
- Bagged results of 30 XGBoost runs - 14.98
- Avg Blend (LightGBM,Catboost,XGBoost) - 14.58

Benchmarking ACC

Dataset	Baseline		XGBoost		LightGBM		Catboost	
	Test	Val	Test	Val	Test	Val	Test	Val
Higgs	0.4996	0.5005	0.8353	0.8512	*0.8573*	0.8577	0.8498	0.8496
Epsilon	0.4976	0.5008	-	-	0.9513	0.9518	*0.9537*	0.9538
Microsoft	0.2251	0.3974	*0.4917*	0.6443	0.4871	0.6473	0.3782	0.5492
Yahoo	0.5802	0.8106	*0.7983*	0.9146	0.7965	0.9142	0.7351	0.8849

Benchmarking TIME

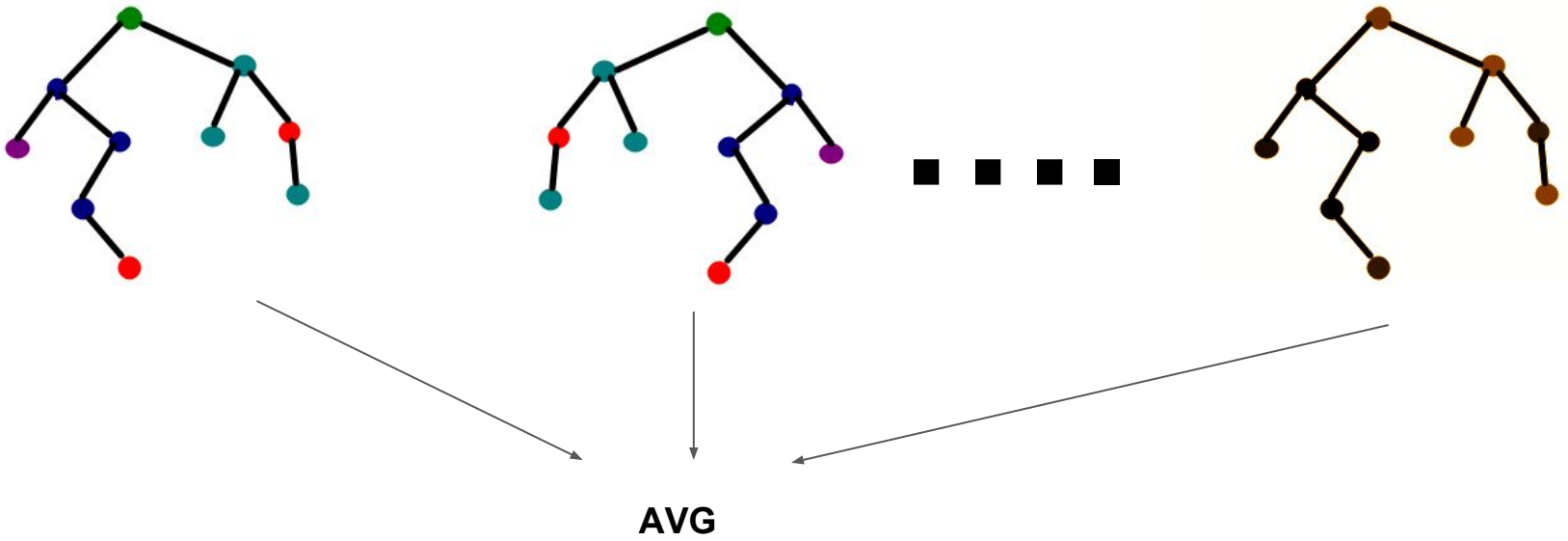


https://arxiv.org/pdf/1809.04559.pdf

Scikit-learn Gradient Boosting

- Written in pure Python/Numpy (easy to extend)
- Builds on top of sklearn.tree.DecisionTreeRegressor

Bagging & Bootstrapping



Random Forest

- Random Forest is a good baseline
- Not easy to overfit.
- Minimal Parameter tuning
- Gradient Boosting in general outperforms RF
- However depending on the dataset, it’s not trivial

Resources

Implementations

- XGBOOST - https://github.com/dmlc/xgboost
- LightGBM - https://github.com/Microsoft/LightGBM
- CatBoost - https://github.com/catboost/catboost
- Random Forest - https://scikit-learn.org/stable/
- Scikit-optimize - https://scikit-optimize.github.io/

Papers

- XGBOOST - https://arxiv.org/pdf/1603.02754.pdf
- LightGBM - https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
- CatBoost - http://learningsys.org/nips17/assets/papers/paper_11.pdf
- DART - http://proceedings.mlr.press/v38/korlakaivinayak15.pdf

Kaggle Solution

- Ubar - https://www.kaggle.com/c/ubaar-competition/discussion/60743
- Safe Driver Prediction - https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/discussion/44629
- Mercedes-Benz --https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion/37700

Questions?

Adrian Spataru
Data Scientist at Know-Center
adrian@spataru.at

Anand Subramoney
Researcher at TU Graz
anand@igi.tugraz.at