XGBoost: the algorithm that wins every competition

Poznań Univeristy of Technology; April 28th, 2016

meet.ml #1 - Applied Big Data and Machine Learning

By Jarosław Szymczak

jarek.szymczak@gmail.com

Data science competitions

[topcoder]™

CrowdANALYTIX

TUNEDIT

kaggle

DRIVENDATA

HackerRank

datascience.net

Some recent competitions from Kaggle

AVITO CONTEXT AD CLICK PREDICTION

(Binary classification problem)

otto group

Product Classification Challenge
(Multi-label classification problem)

ROSSMANN

Store Sales
(Regression problem)

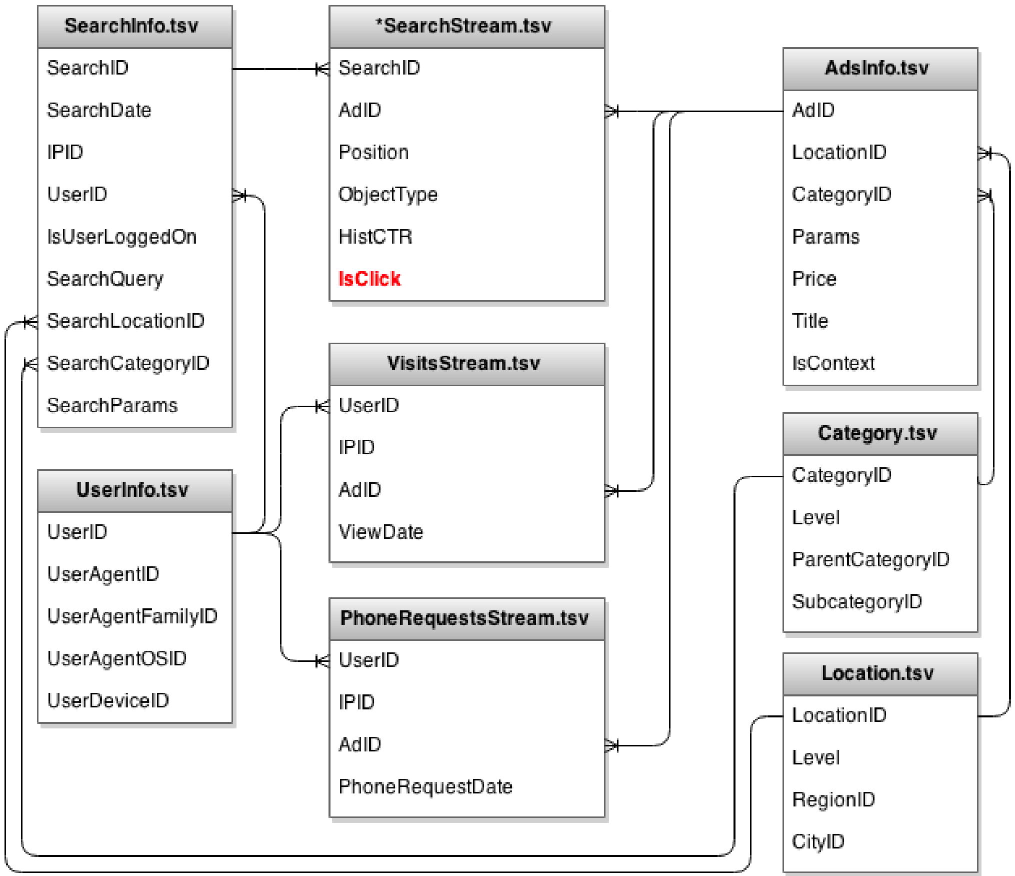
Some recent competitions from Kaggle

Features:

$\times$ 93

Some recent competitions from Kaggle

Features:



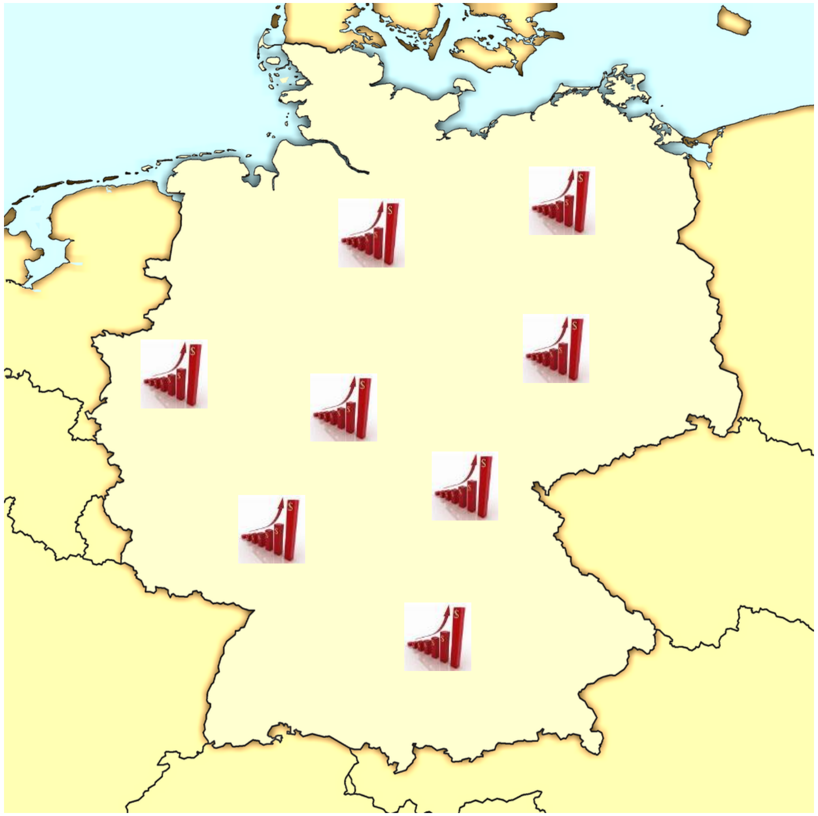
AVITO CONTEXT AD CLICK PREDICTION

CTR – click through rate
(~ equal to probability of click)

Some recent competitions from Kaggle

ROSSMANN

Sales prediction for store at certain date

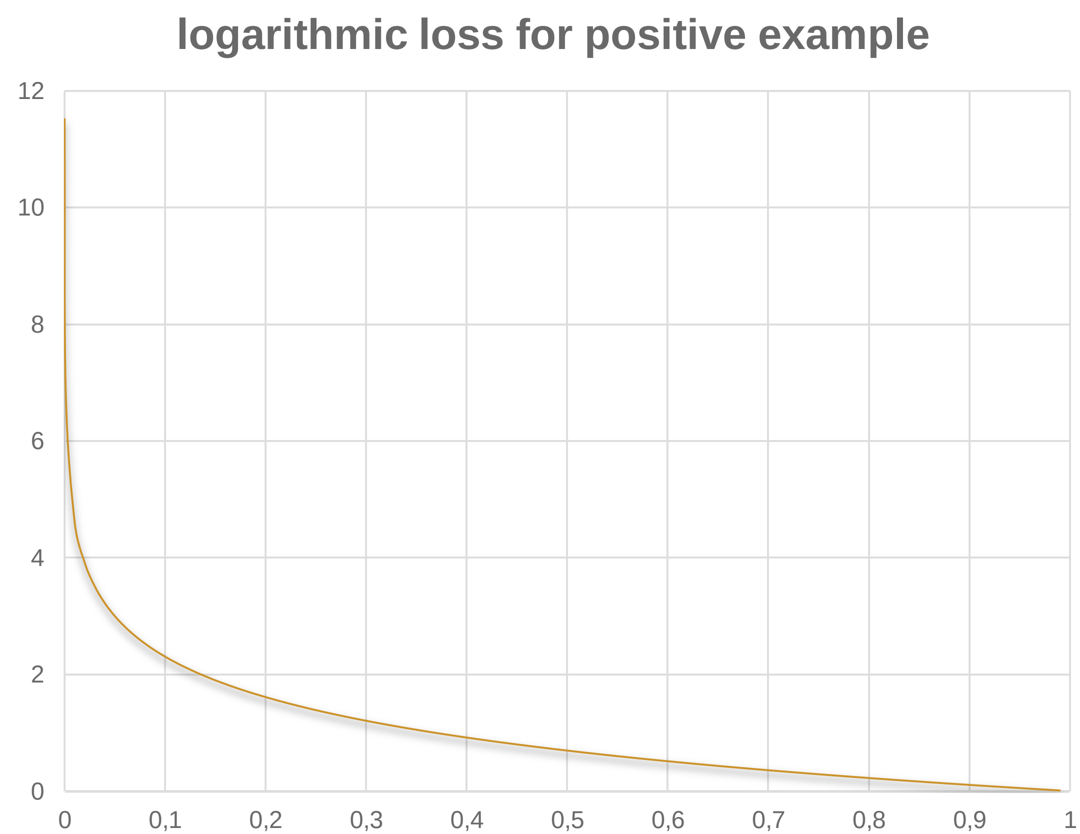


Features:

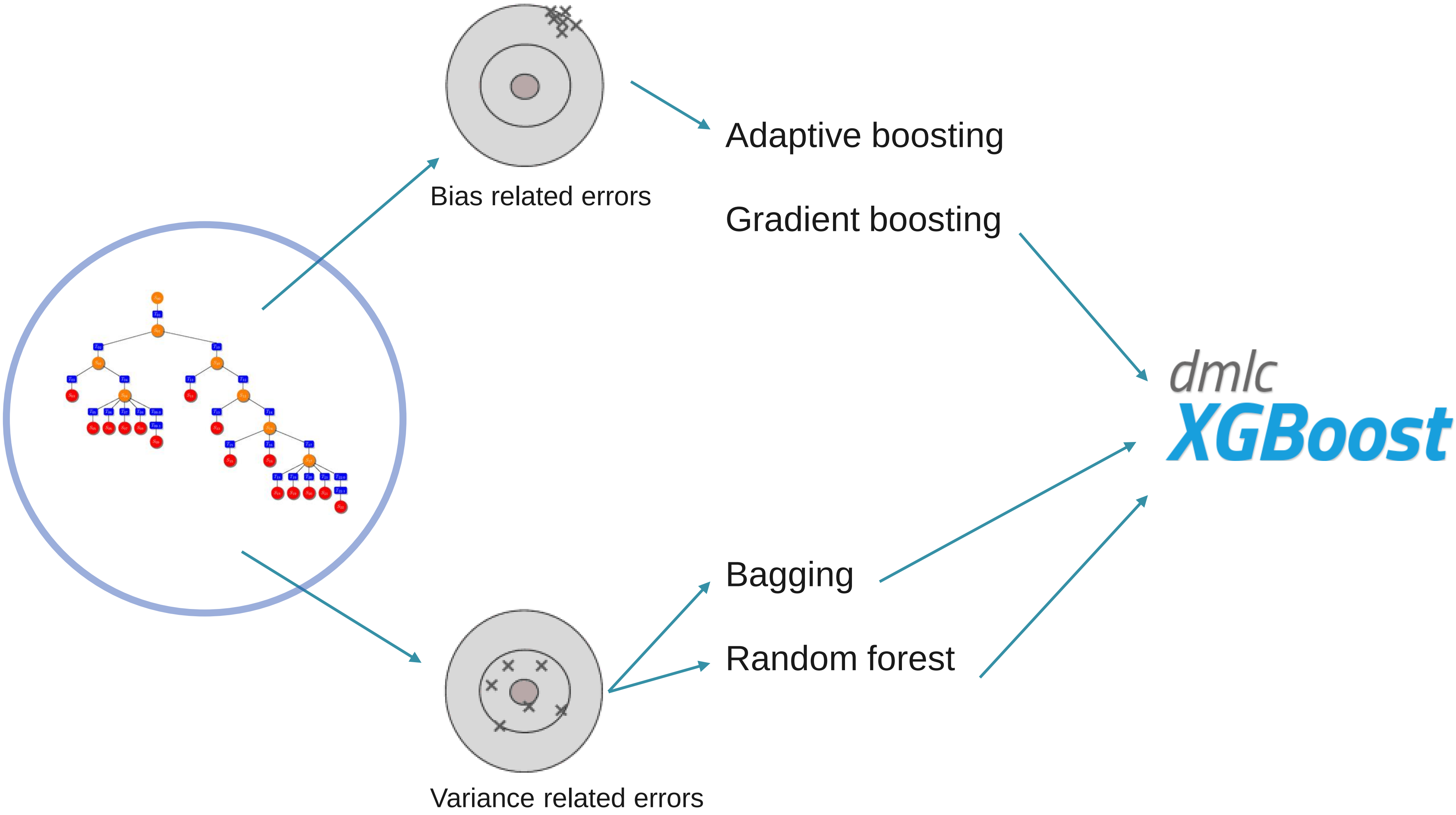
- store id
- date
- school and state holidays
- store type
- assortment
- promotions
- competition

Solutions evaluation

- Root Mean Square Percentage Error (RMSPE)
- Binary logarithmic loss (loss is capped at ~35 for a single observation)
- Multi-class logarithmic loss



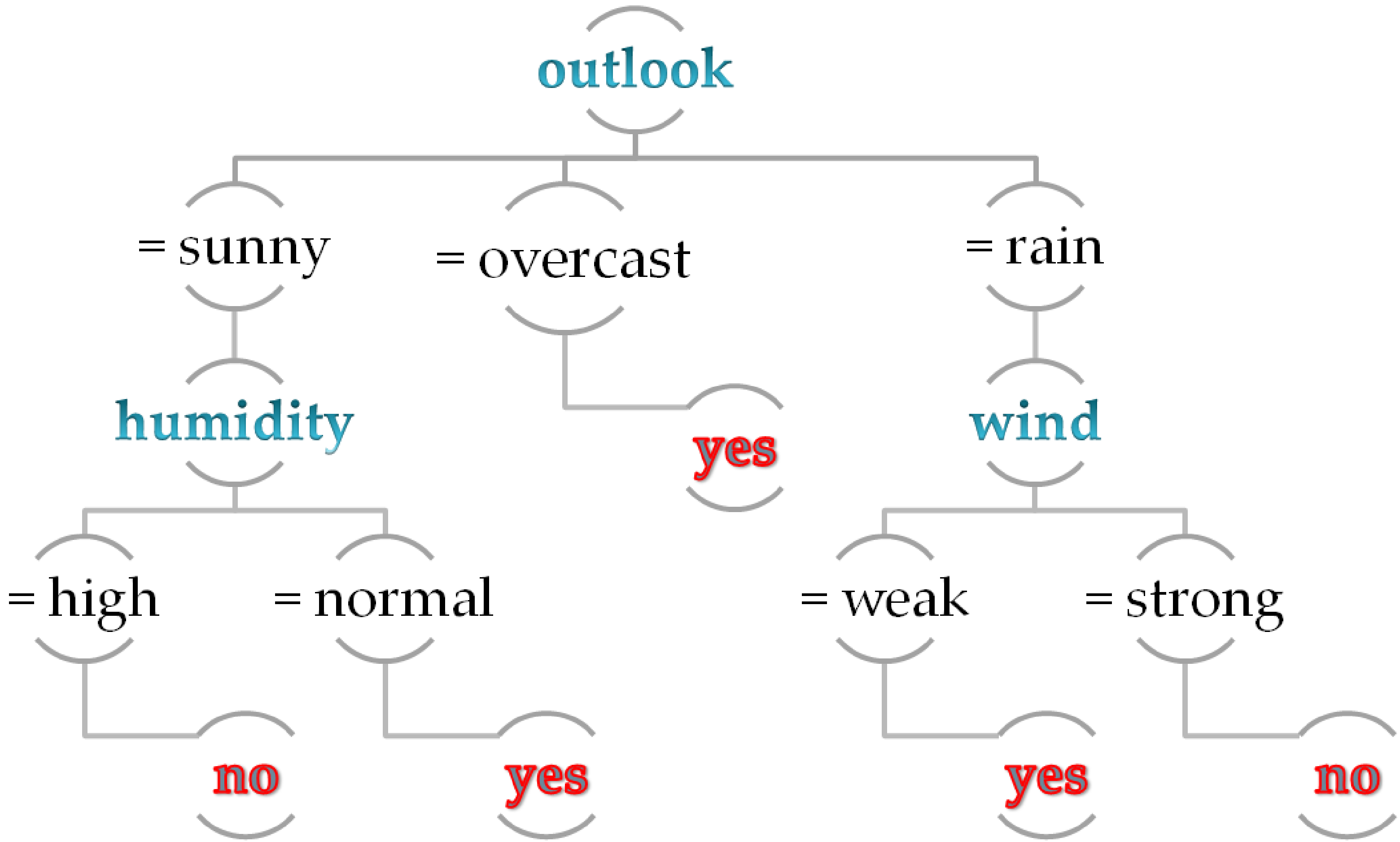
What algorithms we tackle the problems with?

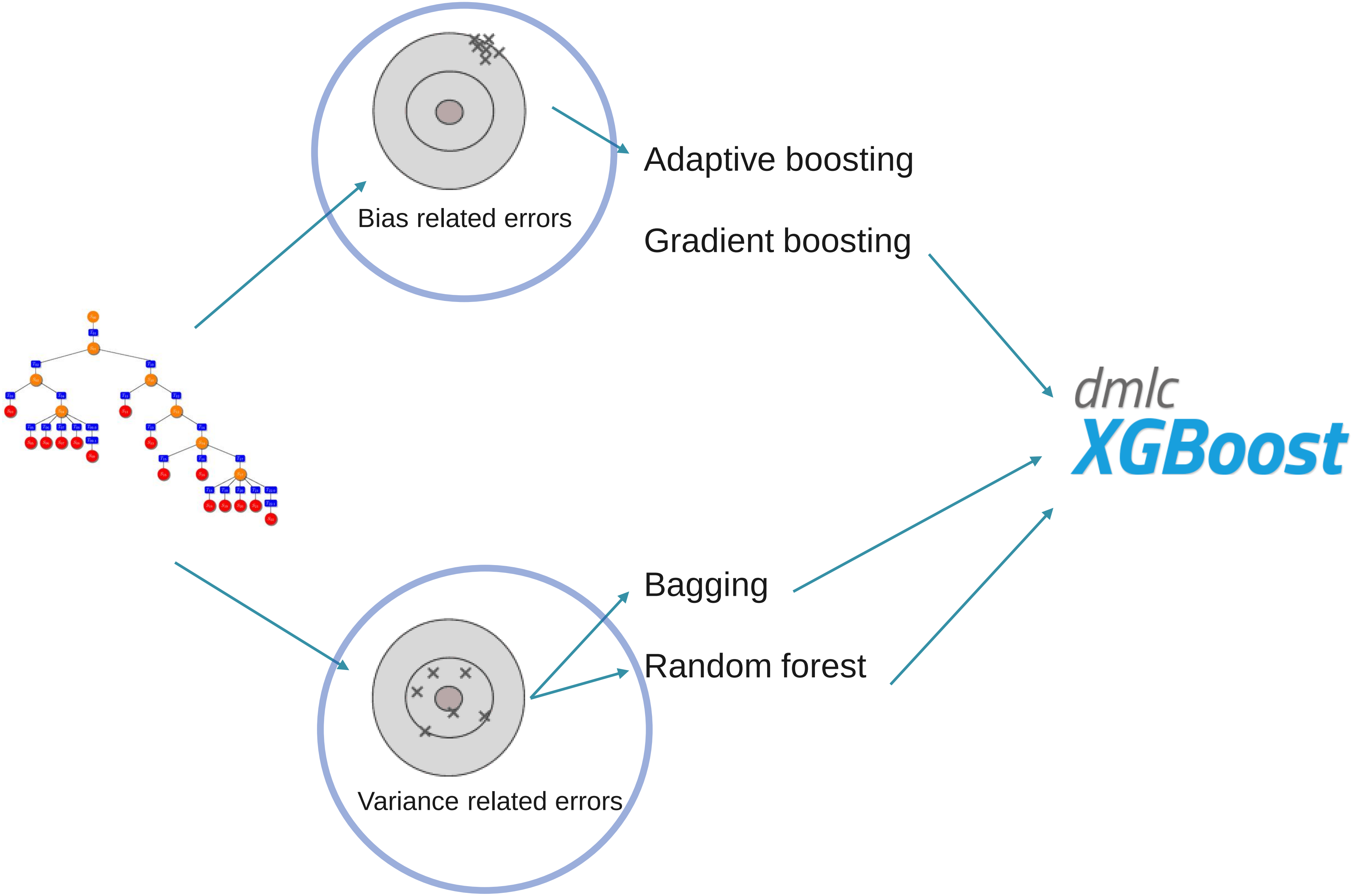

Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

Shall we play tenis?

Outlook	Temperature	Humidity	Wind	Play tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes

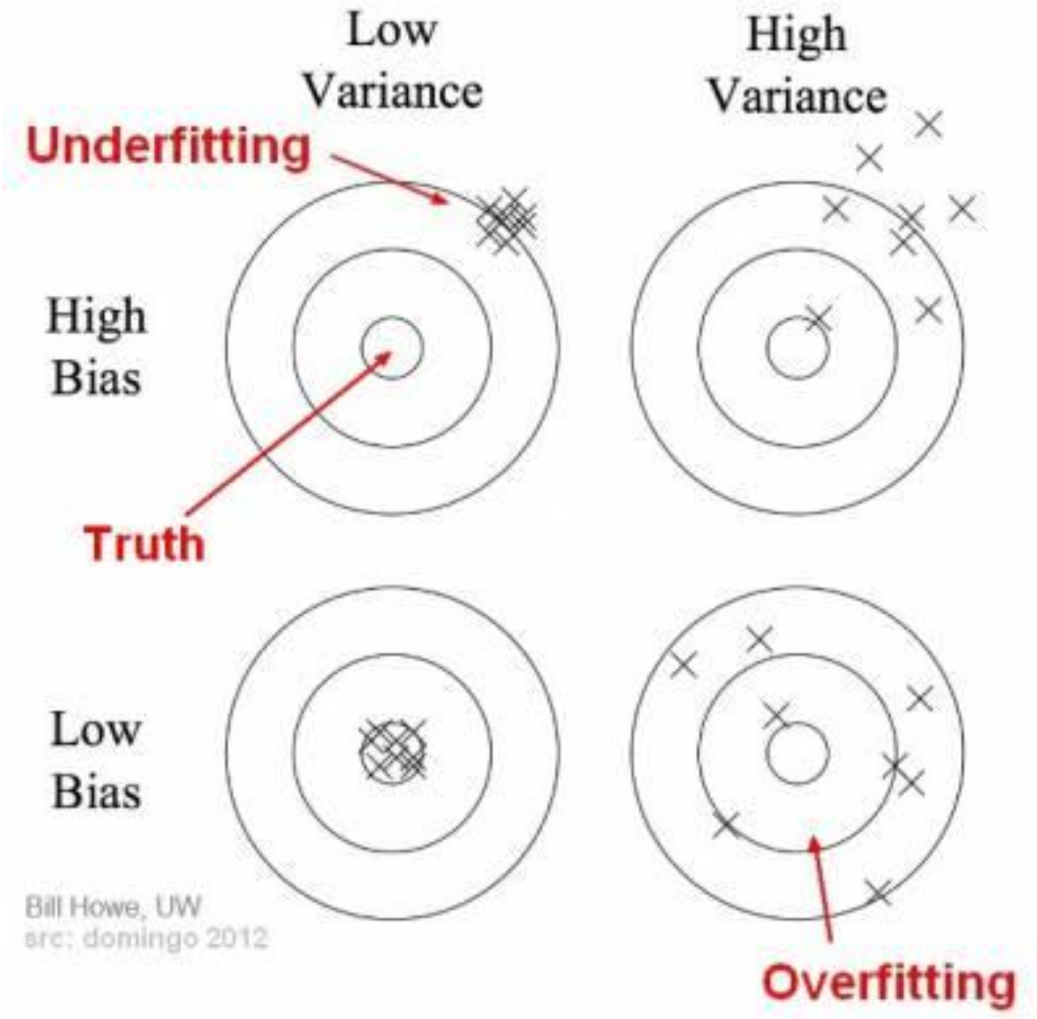
Decision tree standard example – shall we play tenis?

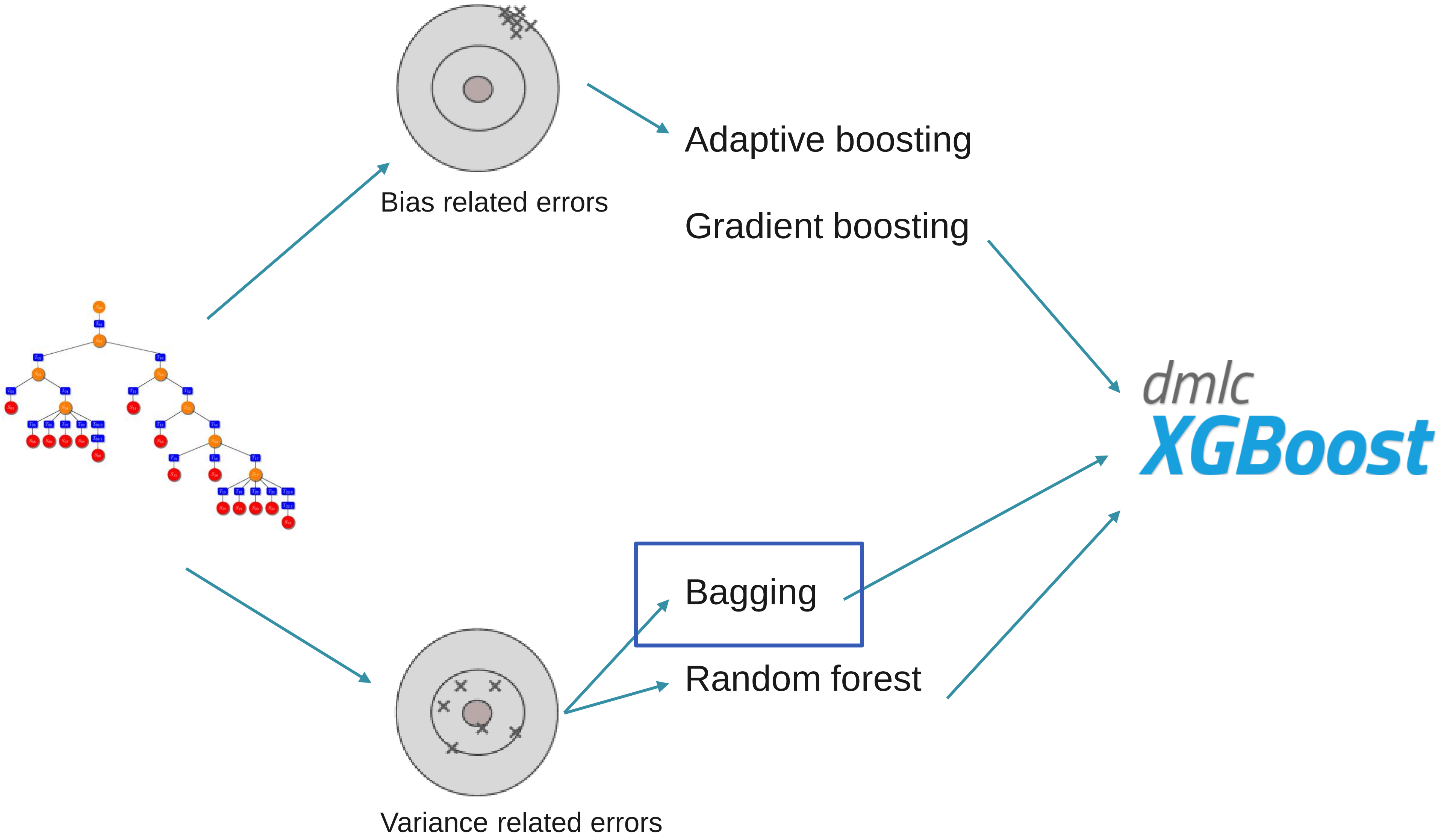



Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

But one tree for complicated problems is simply not enough, and we will need to reduce erros coming from:

- variance
- bias

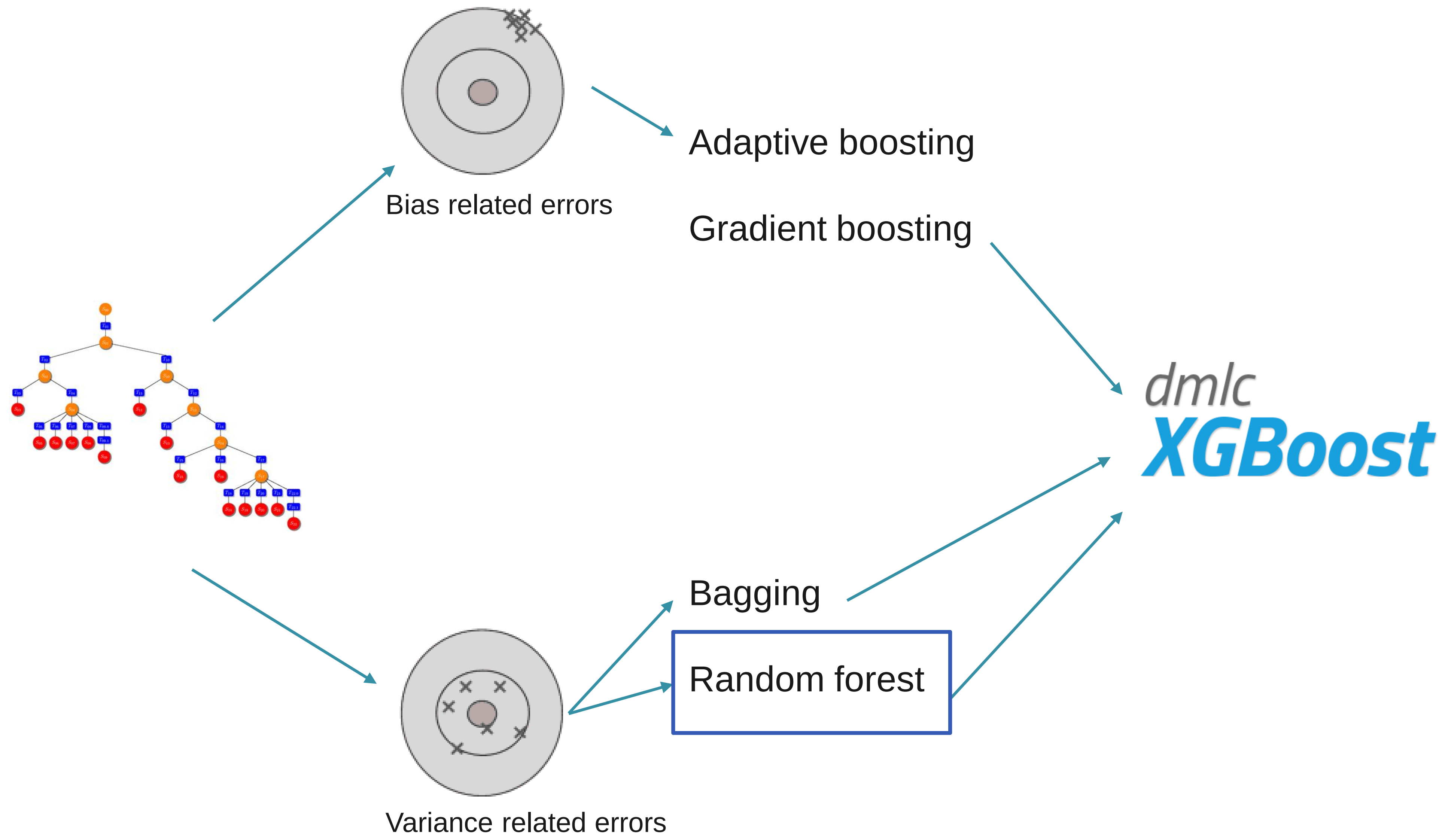



Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

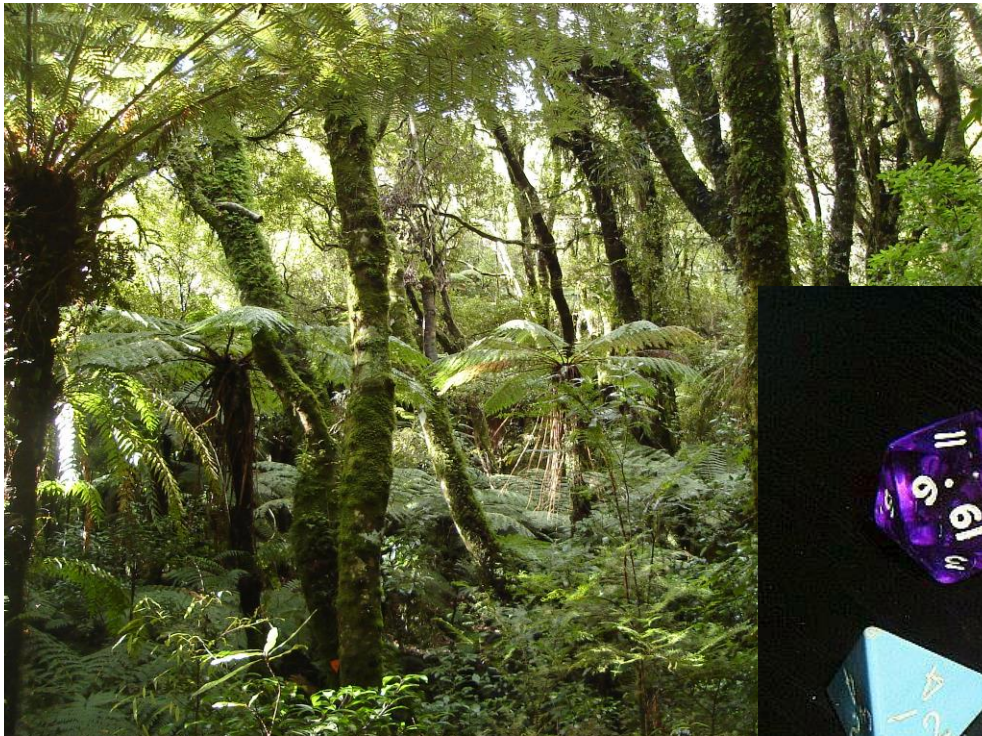
Bagging – bootstrap aggregation

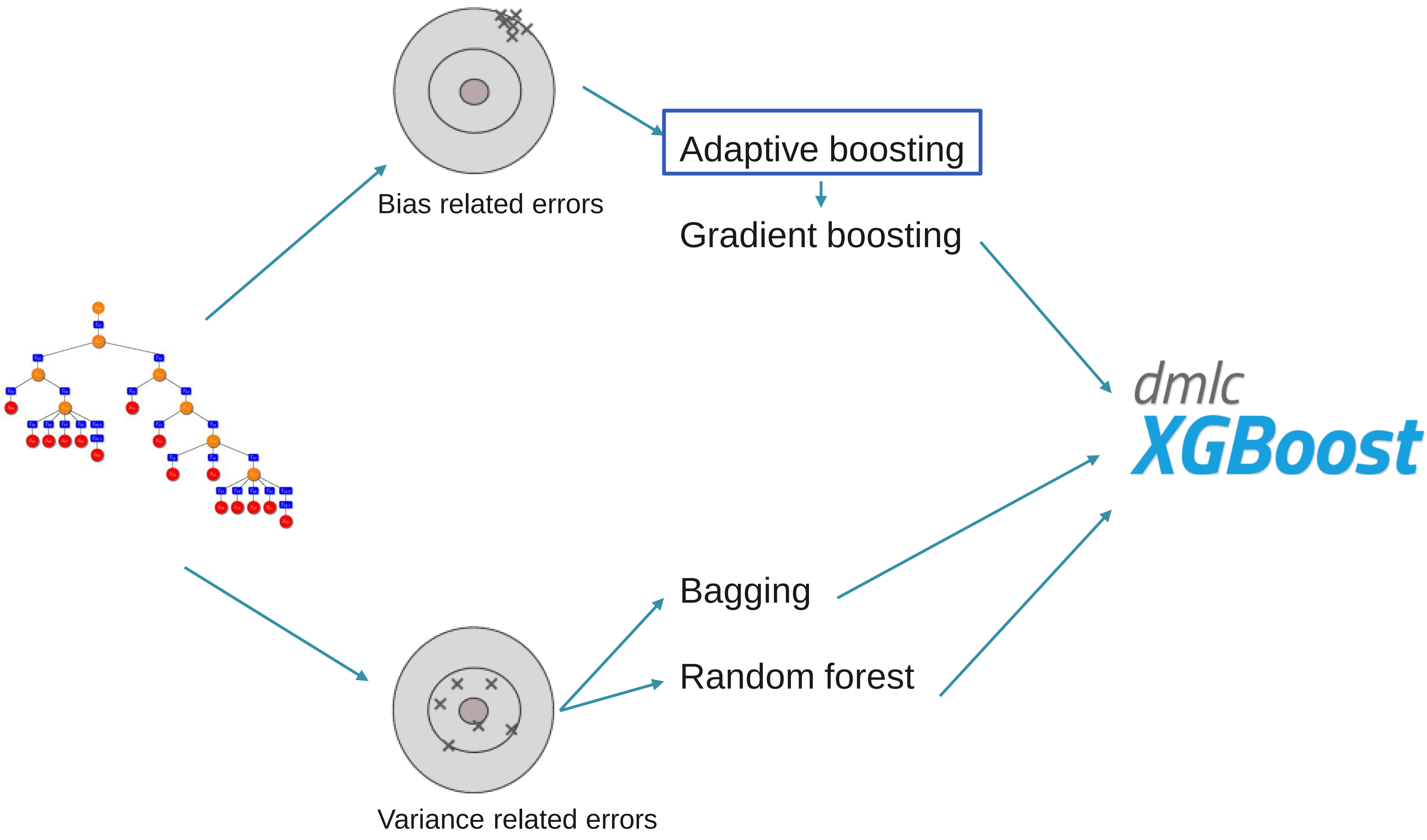
Our initial dataset:	1	2	3	4	5	6	
First draw:	1	2	2	2	3	5	Model 1
Second draw:	1	1	3	4	4	6	Model 2
Third draw:	2	3	3	4	5	5	Model 3
Fourth draw:	2	4	4	5	5	6	Model 4

Average: Final model

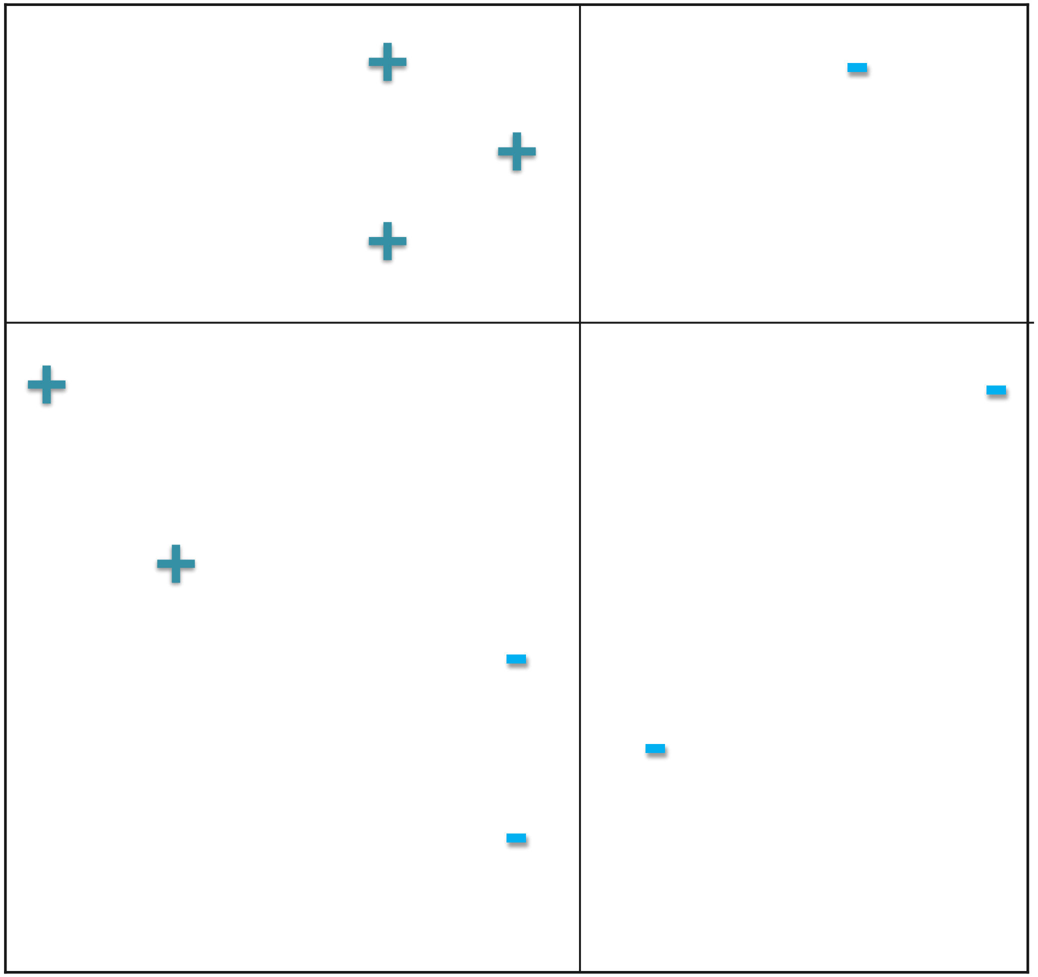

Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

Random forest




Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

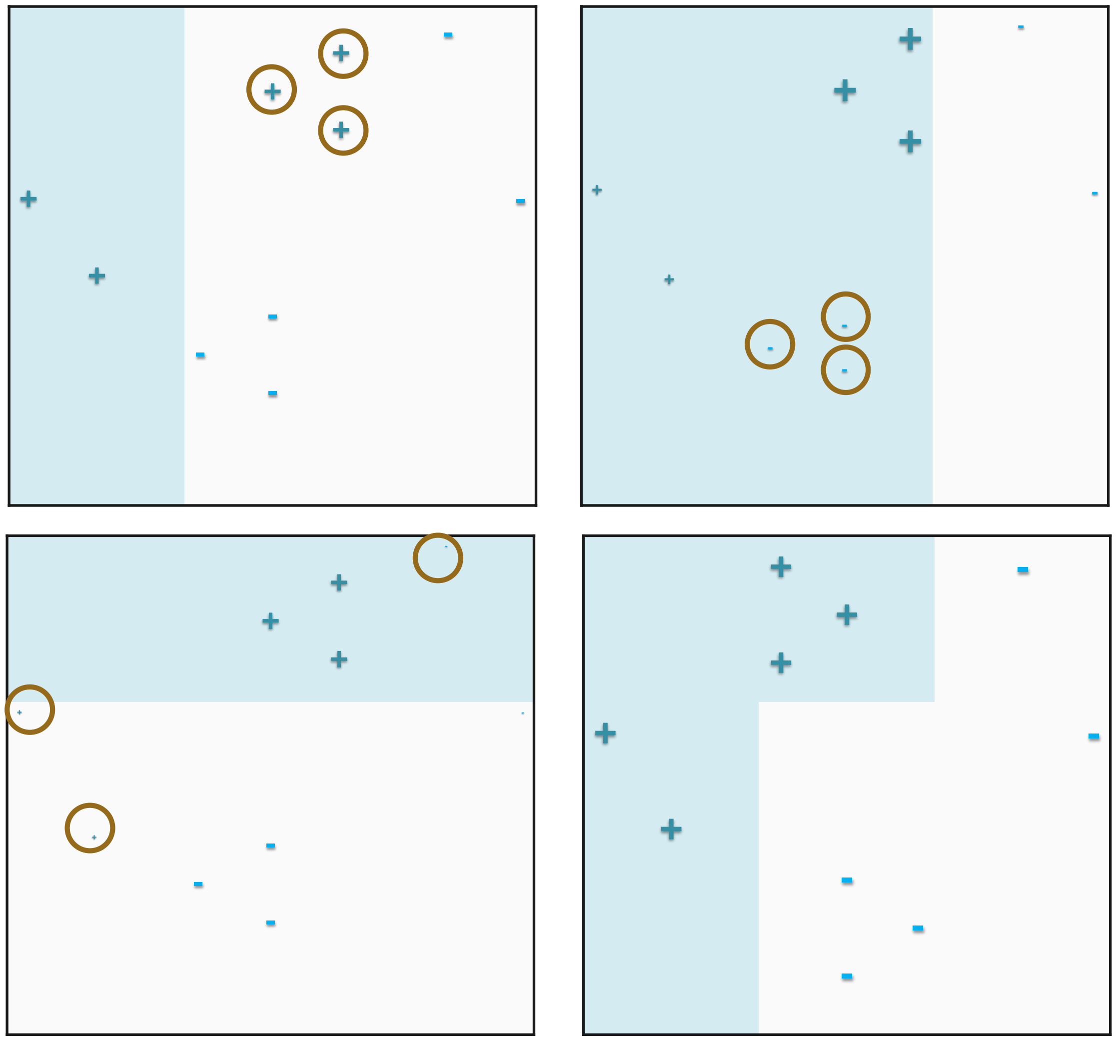
AdaBoost – adaptive boosting



AdaBoost – adaptive boosting

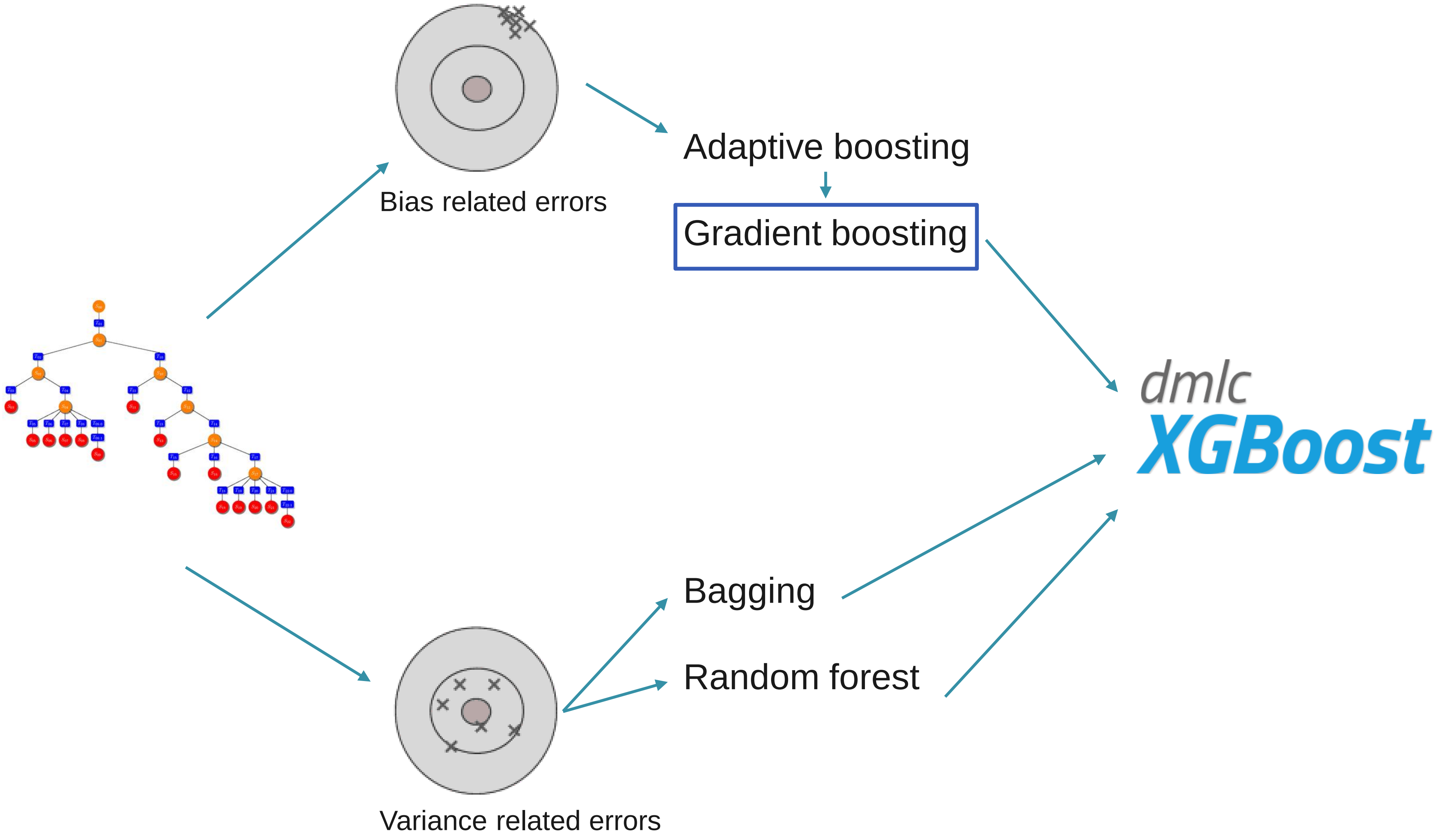
h_1
$\varepsilon_1 = 0{,}30$
$\alpha_1 = 0{,}42$

h_2
$\varepsilon_2 = 0{,}21$
$\alpha_2 = 0{,}65$



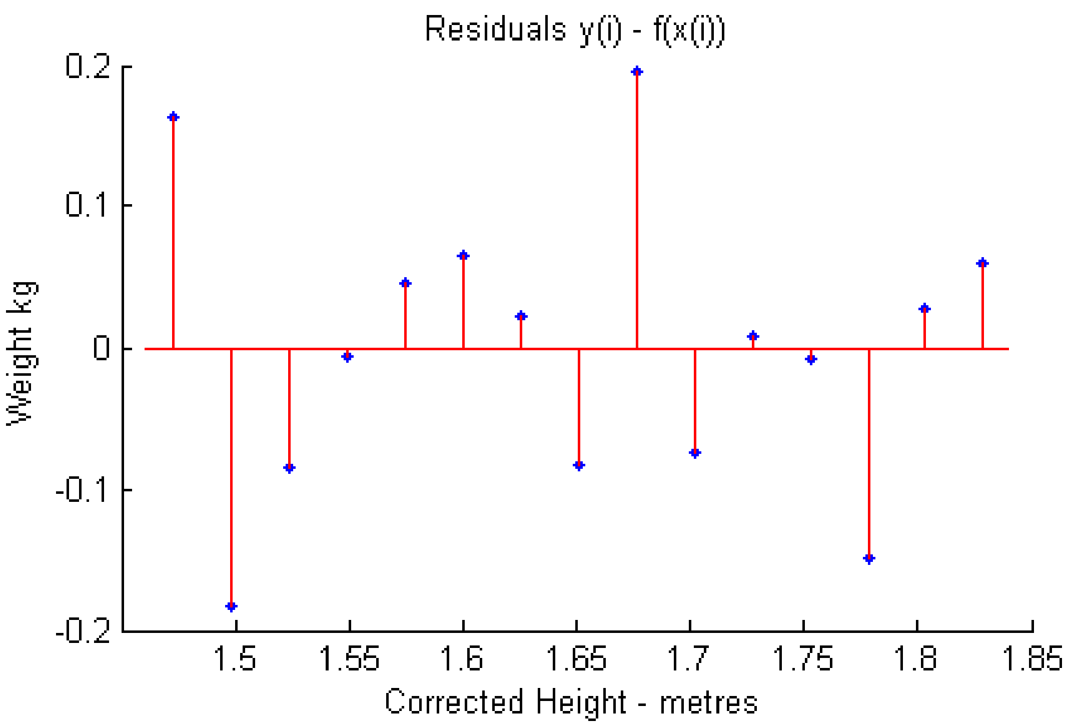
h_3
$\varepsilon_3 = 0{,}14$
$\alpha_3 = 0{,}92$

$$H = 0{,}42h_1 + 0{,}65h_2 + 0{,}92h_3$$


Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

Another boosting approach

What if we, instead of reweighting examples, made some corrections to prediction errors directly?



We add new model, to the one we already have, so:

Y_{pred} = X1(Y) + NEW(Y)

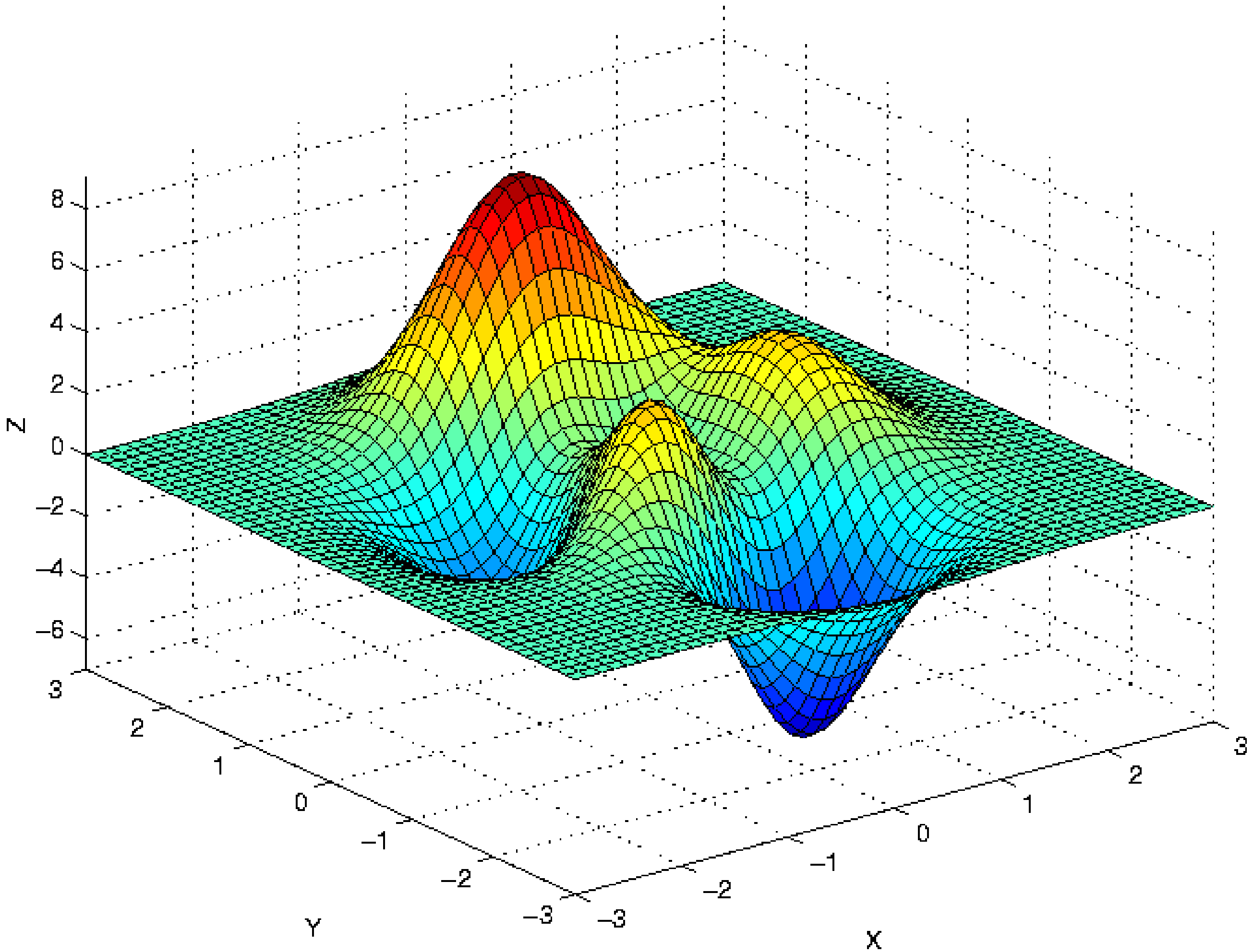
NEW(Y) = X1(Y) - Y_{pred}

our residual

Note:

Residual is a gradient of single observation error contribution in one of the most common evaluation measure for regression: root mean squared error (RMSE)

Gradient descent



Gradient boosting

Fit model to initial data

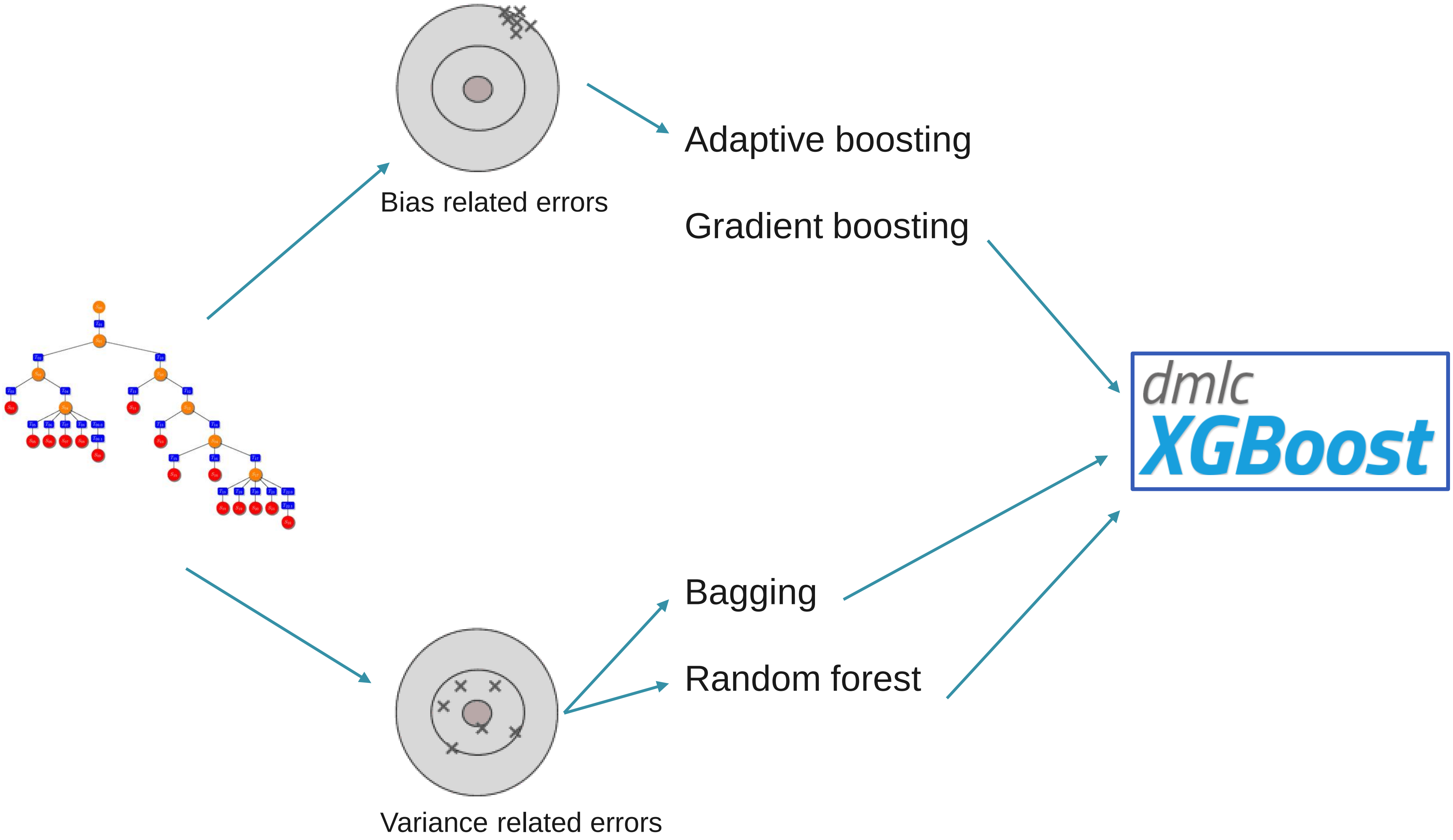
It can be:
- same algorithm as for further steps
- or something very simple (like uniform probabilities or average target in regression)

Fit pseudo-residuals

For any function that:
- agrregates the error from examples (e.g. log-loss, RMSE, but not AUC)
- you can calculate gradient on example level (it is called pseudo-residual)

Finalization

Sum up all the models


Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

XGBoost

eXtreme Gradient Boosting

custom tree building algorithm

XGBoost

by Tianqi Chen

Used for:

- classification
- regression
- ranking

with custom loss functions

Interfaces for Python and R, can be executed on YARN

XGBoost – handling the features

Numeric values

- for each numeric value, XGBoost finds the best available split (it is always a binary split)
- algorithm is designed to work with numeric values only

Nominal values

- need to be converted to numeric ones
- classic way is to perform one-hot-encoding / get dummies (for all values)
- for variables with large cardinality, some other, more sophisticated methods may need to be used

Missing values

- XGBoost handles missing values separately
- missing value is always added to a branch in which it would minimize the loss (so it is either treated as very large / very small value)
- it is a great advantage over, e.g. scikit-learn gradient boosting implementation

XGBoost – loss functions

Used as optimization objective

- logloss (optimized in complicated manner) for objectives binary:logistic and reg:logistic (they only differ by evaluation measure, all the rest is the same)
- softmax (which is logloss transofrmed to handle multiple classes) for objective multi:softmax
- RMSE (root mean squared error) for objective reg:linear

Only measured

- AUC – area under ROC curve
- MAE – mean absolute error
- error / merror – error rate for binary / multi-class classification

Custom

- only measured
  - simply implement the evaluation function
- used as optimization objective
  - provided that you have a measure that is aggregation of element-wise losses and is differentiable
  - you create a function for element-wise gradient (derivative of prediction function)
  - then hessian (second derivative of prediction function)
  - that’s it, algorithm will approximate your loss function with Taylor series (pre-defined loss functions are implemented in the same manner)

XGBoost – setup for Python environment

Installing XGBoost in your environment

docker	Check out this wonderful tutorial to install Docker with kaggle Python stack: http://goo.gl/wSCkmC
Linux	http://xgboost.readthedocs.org/en/latest/build.html
Microsoft Windows + MinGW	Link as above, expect potential troubles
Microsoft Windows + Visual Studio	Clone: https://github.com/dmlc/xgboost At revision: 219e58d453daf26d6717e2266b83fca0071f3129 And follow the instructions in it (it is of course older version)

XGBoost code example on Otto data

Train dataset

	id	feat_1	feat_2	feat_3	feat_4	...	feat_90	feat_91	feat_92	feat_93	target
0	1	1	0	0	0	...	0	0	0	0	Class_1
1	2	0	0	0	0	...	0	0	0	0	Class_1
2	3	0	0	0	0	...	0	0	0	0	Class_1

3 rows × 95 columns

Test dataset

	id	feat_1	feat_2	feat_3	feat_4	...	feat_89	feat_90	feat_91	feat_92	feat_93
0	1	0	0	0	0	...	0	0	0	0	0
1	2	2	2	14	16	...	4	0	0	2	0
2	3	0	1	12	1	...	0	0	0	0	1

3 rows × 94 columns

Sample submission

	id	Class_1	Class_2	Class_3	...	Class_6	Class_7	Class_8	Class_9
0	1	1	0	0	...	0	0	0	0
1	2	1	0	0	...	0	0	0	0
2	3	1	0	0	...	0	0	0	0

3 rows × 10 columns

XGBoost – simple use example (direct)

```
import pandas as pd
import numpy as np
import xgboost as xgb

train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")
submission = pd.read_csv("../input/sampleSubmission.csv")
#target is class_1, ..., class_9 - needs to be converted to 0, ..., 8
train['target'] = train['target'].apply(lambda val: np.int64(val[-1:]))-1

Xy_train = train.as_matrix()
X_train = Xy_train[:,1:-1]
y_train = Xy_train[:,-1:].ravel()

X_test = test.as_matrix()[:,1:]

dtrain = xgb.DMatrix(X_train, y_train, missing=np.NaN)
dtest = xgb.DMatrix(X_test, missing=np.NaN)

params = {"objective": "multi:softprob", "eval_metric": "mlogloss", "booster" : "gbtree",
          "eta": 0.05, "max_depth": 3, "subsample": 0.6, "colsample_bytree": 0.7, "num_class": 9}

num_boost_round = 100

gbm = xgb.train(params, dtrain, num_boost_round)
pred = gbm.predict(dtest)

print(gbm.eval(dtrain))
submission.iloc[:,1:] = pred

submission.to_csv("submission.csv", index=False)
```

```
b'[0]\teval-mlogloss:0.761362'
```

scikit-learn conventional names

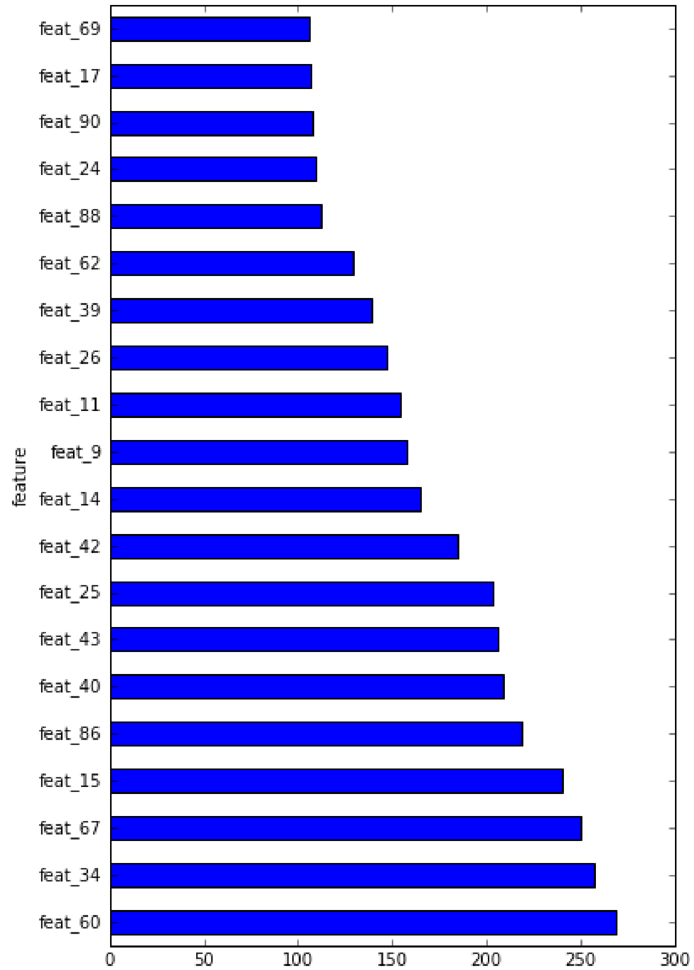
to account for examples importance we can assign weights to them in DMatrix (not done here)

for direct use we need to specify number of classes

Link to script: https://goo.gl/vwQeXs

XGBoost – feature importance

```
%matplotlib inline

importance = gbm.get_fscore()

fdict = {}
for key, name in enumerate(train.columns[1:-1]):
    fdict['f{0}'.format(key)] = name

importance_with_names = []

for key, value in importance.items():
    importance_with_names.append((fdict[key], value))


pd.DataFrame(importance_with_names, columns=['feature', 'fscore']).\
set_index('feature').sort_values(['fscore'], ascending=[0])[:20].\
plot(kind="barh", legend=False, figsize=(6, 10))
```



Link to script: http://pastebin.com/uPK5aNkf

XGBoost – simple use example (in scikit-learn style)

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import log_loss

train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")
submission = pd.read_csv("../input/sampleSubmission.csv")
#target is class_1, ..., class_9 - needs to be converted to 0, ..., 8
train['target'] = train['target'].apply(lambda val: np.int64(val[-1:]))-1

Xy_train = train.as_matrix()
X_train = Xy_train[:,1:-1]
y_train = Xy_train[:,-1:].ravel()

X_test = test.as_matrix()[:,1:]

num_boost_round = 100

gbm = xgb.XGBClassifier(max_depth=3, learning_rate=0.05, objective="multi:softprob", subsample=0.6,
                  colsample_bytree=0.7, n_estimators=num_boost_round)

gbm = gbm.fit(X_train, y_train)

pred = gbm.predict_proba(test)

y_hat_train = gbm.predict_proba(X_train)
print(log_loss(y_train, y_hat_train))

submission.iloc[:,1:] = pred
submission.to_csv("submission_sklearn.csv", index=False)
```

```
0.758585101098
```

Link to script: https://goo.gl/IPxKh5

XGBoost – most common parameters for tree booster

subsample
- ratio of instances to take
- example values: 0.6, 0.9

colsample_by_tree
- ratio of columns used for whole tree creation
- example values: 0.1, …, 0.9

colsample_by_level
- ratio of columns sampled for each split
- example values: 0.1, …, 0.9

eta
- how fast algorithm will learn (shrinkage)
- example values: 0.01, 0.05, 0.1

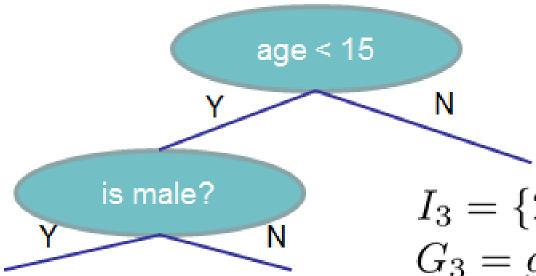
max_depth
- maximum numer of consecutive splits
- example values: 1, …, 15 (for dozen or so or more, needs to be set with regularization parametrs)

min_child_weight
- minimum weight of children in leaf, needs to be adopted for each measure
- example values: 1 (for linear regression it would be one example, for classification it is gradient of pseudo-residual)

XGBoost – regularization parameters for tree booster

alpha	lambda	gamma
• L1 norm (simple average) of weights for whole objective function	• L2 norm (root from average of squares) of weights, added as penalty to objective function	• L0 norm, multiplied by numer of leafs in a tree is used to decide whether to make a split

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



$I_1 = \{1\}$
$G_1 = g_1$
$H_1 = h_1$

$I_2 = \{4\}$
$G_2 = g_4$
$H_4 = h_4$

$I_3 = \{2, 3, 5\}$
$G_3 = g_2 + g_3 + g_5$
$H_3 = h_2 + h_3 + h_5$

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

lambda is in denominator because of various transformation, in original objective it is „classic" L2 penalty

XGBoost – some cool things not necesarilly present elswhere

Watchlist

You can create some small validation set to prevent overfitting without need to guess how many iterations would be enough

Warm start

Thanks to the nature of gradient boosting, you can take some already woking solution and start the „improvement" from it

Easy cut-off at any round when classyfing

Simple but powerful while tuning the parameters, allows you to recover from overfitted model in easy way

Last, but not least – who won?

Winning solutions

ROSSMANN

The winning solution consists of over 20 xgboost models that each need about two hours to train when running three models in parallel on my laptop. So I think it could be done within 24 hours. Most of the models individually achieve a very competitive (top 3 leaderboard) score.

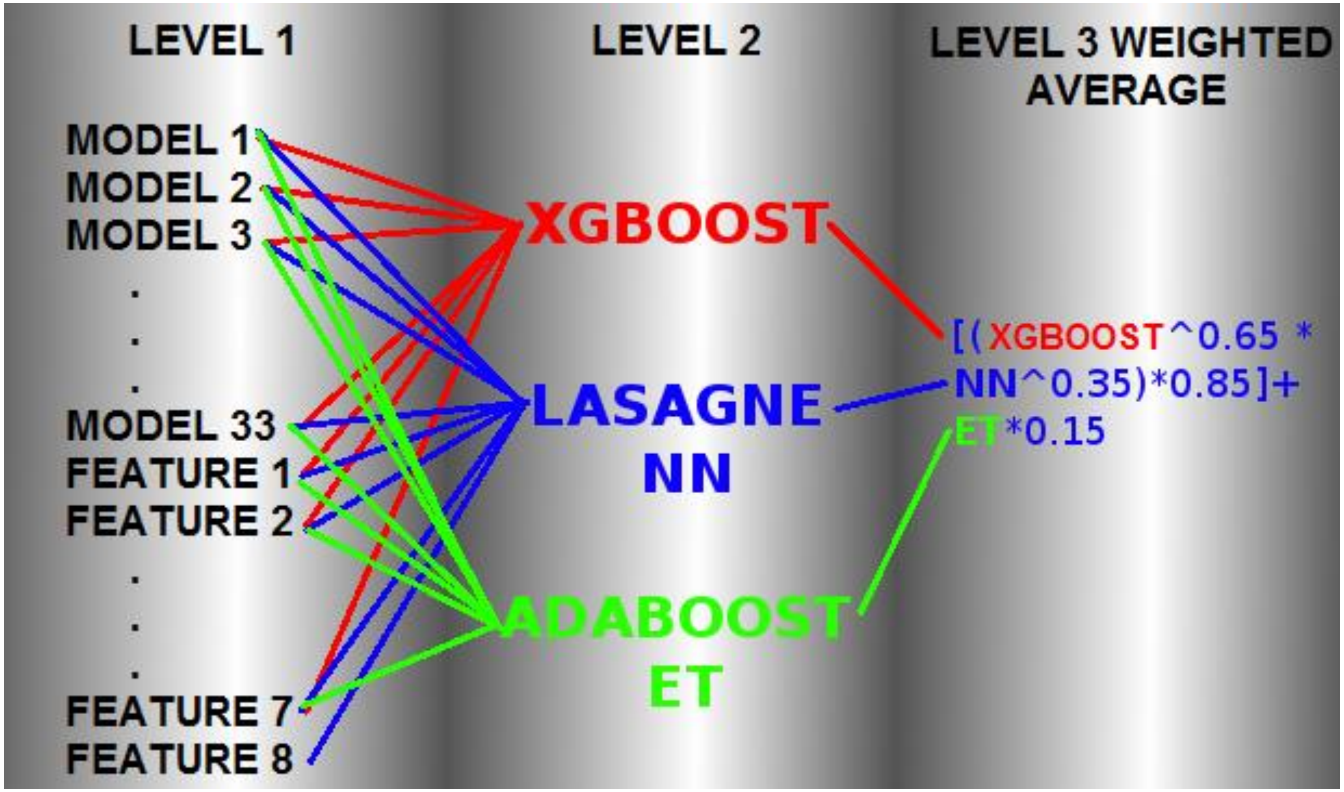
by Gert Jacobusse

AVITO CONTEXT AD CLICK PREDICTION

I did quite a bit of manual feature engineering and my models are entirely based on xgboost. Feature engineering is a combination of “brutal force” (trying different transformations, etc that I know) and “heuristic” (trying to think about drivers of the target in real world settings). One thing I learned recently was entropy based features, which were useful in my model.

by Owen Zhang

Winning solutions



by Gilberto Titericz Jr and Stanislav Semenov

otto group

Even deeper dive

- XGBoost:
  - https://github.com/dmlc/xgboost/blob/master/demo/README.md

- More info about machine learning:
  - Uczenie maszynowe i sieci neuronowe.
    Krawiec K., Stefanowski J., Wydawnictwo PP, Poznań, 2003. (2 wydanie 2004)
  - The Elements of Statistical Learning: Data Mining, Inference, and Prediction
    by Trevor Hastie, Robert Tibshirani, Jerome Friedman (available on-line for free)

- Sci-kit learn and pandas official pages:
  - http://scikit-learn.org/
  - http://pandas.pydata.org/

- Kaggle blog and forum:
  - http://blog.kaggle.com/ (no free hunch)
  - https://www.datacamp.com/courses/kaggle-python-tutorial-on-machine-learning
  - https://www.datacamp.com/courses/intro-to-python-for-data-science

- Python basics:
  - https://www.coursera.org/learn/interactive-python-1
  - https://www.coursera.org/learn/interactive-python-2