Supervised ML 2

Bank of Uganda

University of Cape Town

Short Course

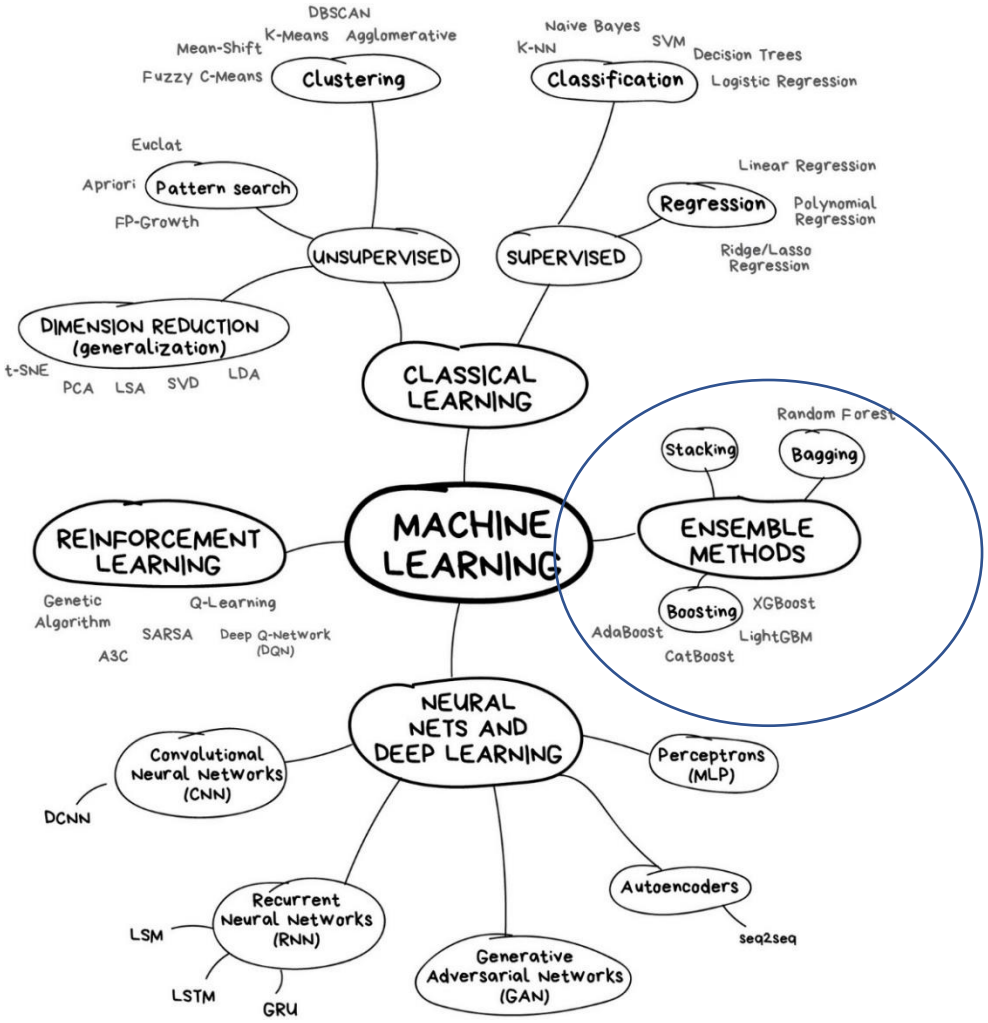
November 2019

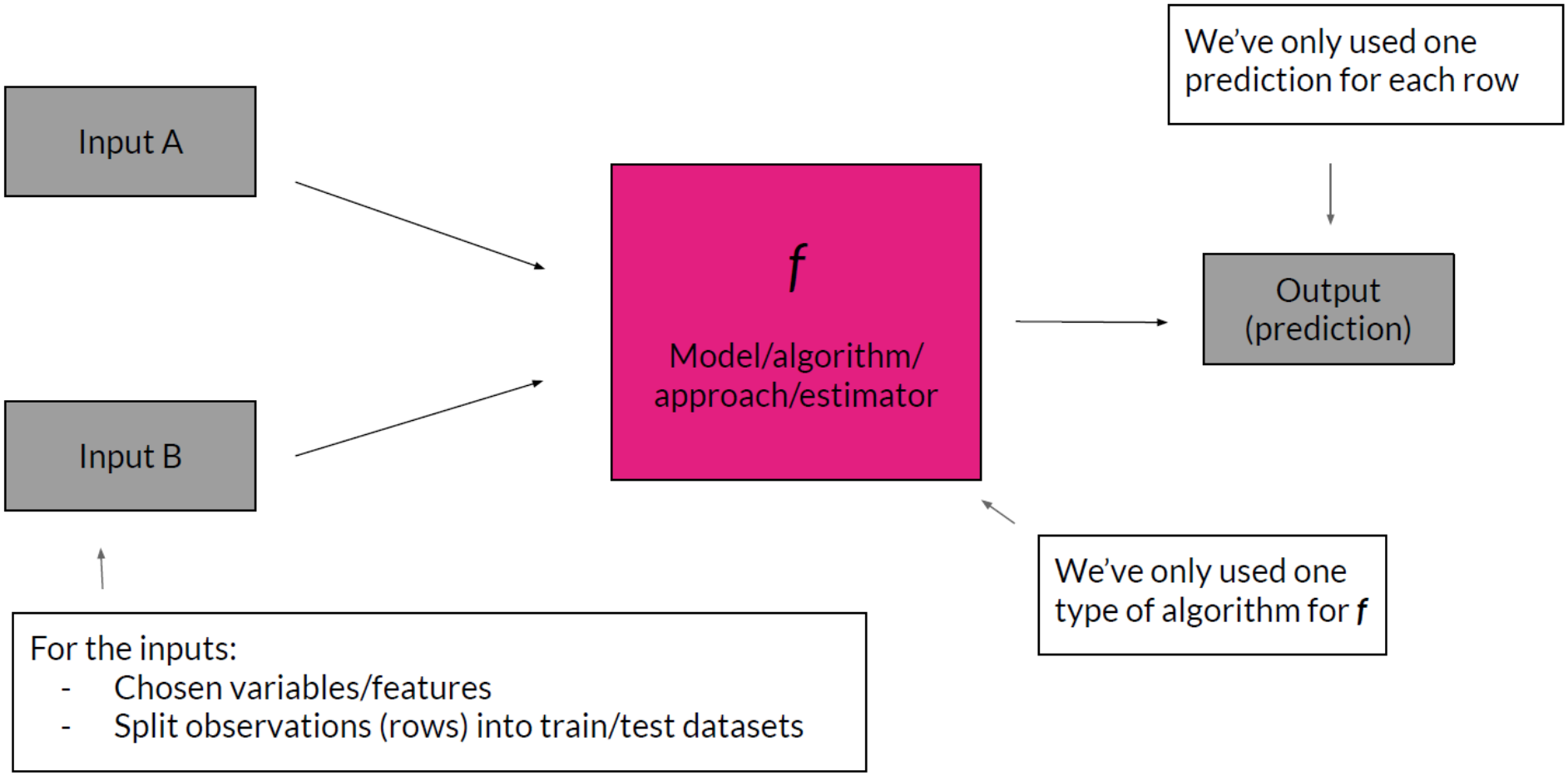
Learning outcomes

- More ‘sophisticated’ supervised machine learning methods
  - Ensemble methods including:
    - Random forests
    - XGBoost

- Implement these ML methods

Where are we?


MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
SUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Pattern search
Euclat
Apriori
FP-Growth
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq


We've only used one prediction for each row
Input A
f
Model/algorithm/ approach/estimator
Output (prediction)
Input B
We've only used one type of algorithm for f
For the inputs:
- Chosen variables/features
- Split observations (rows) into train/test datasets

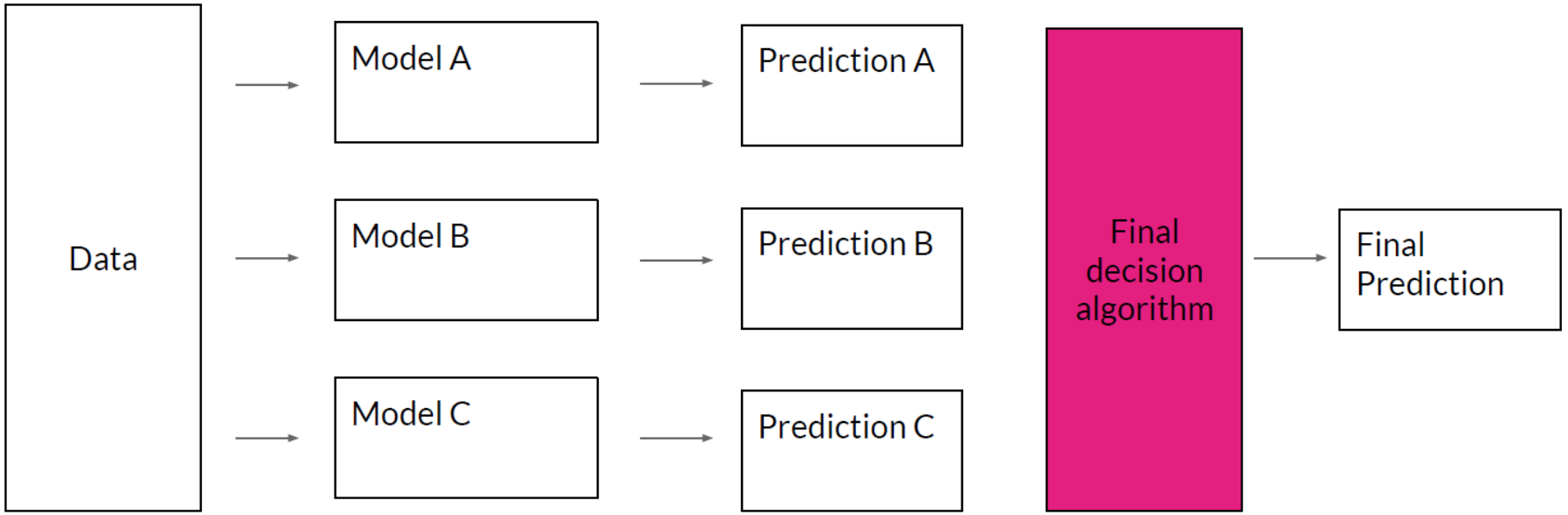
Ensemble models

- Improve performance of models by aggregating multiple (base) models
- Must be some explanatory power in the base models
- Helpful if the base models are diverse
  - They make different mistakes

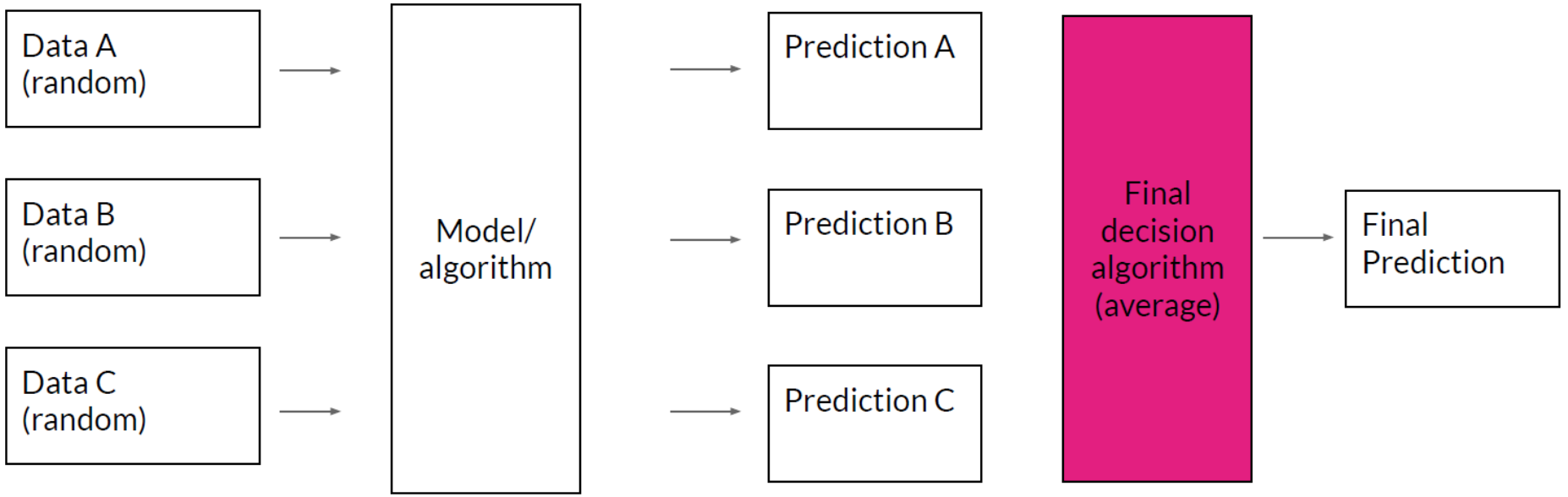
Types

- Stacking
- Bagging
- Boosting
- Random forest
- XGBoost

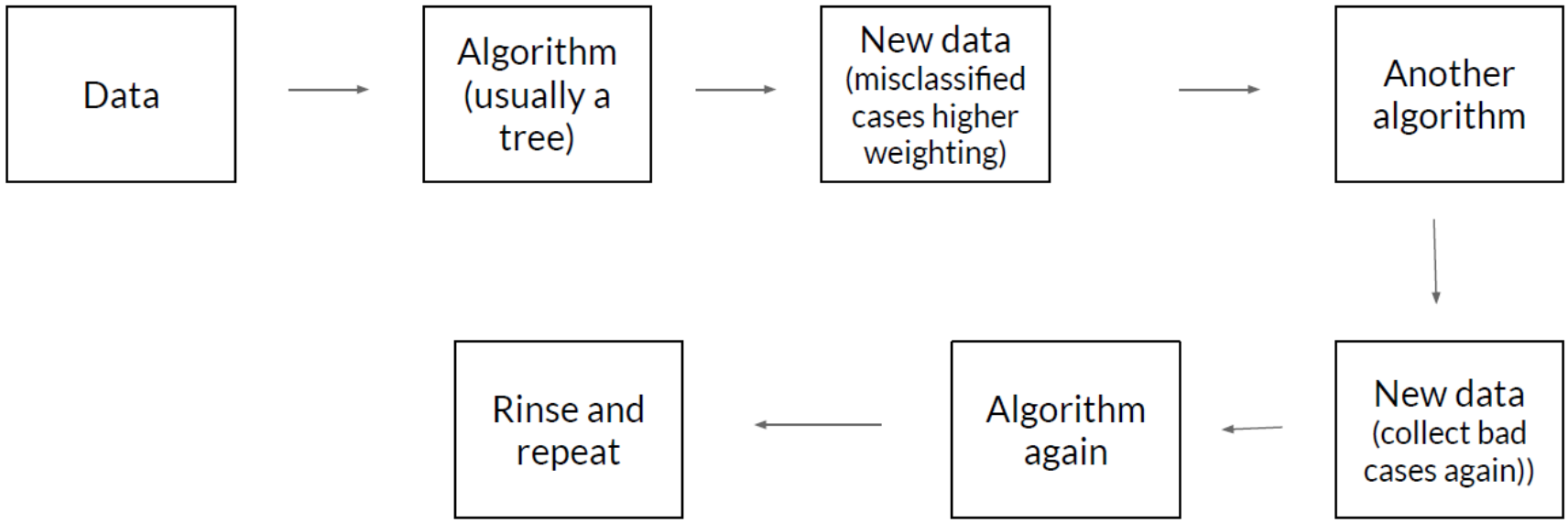
Stacking


Data
Model A
Model B
Model C
Prediction A
Prediction B
Prediction C
Final decision algorithm
Final Prediction

Bagging


Data A
(random)
Data B
(random)
Data C
(random)
Model/
algorithm
Prediction A
Prediction B
Prediction C
Final
decision
algorithm
(average)
Final
Prediction

Boosting


Data
Algorithm (usually a tree)
New data (misclassified cases higher weighting)
Another algorithm
New data (collect bad cases again))
Algorithm again
Rinse and repeat

Random forest

- A version of bagging
- Many trees == forest
- Random over two dimensions
  - Random draws of data (bagging)
  - Random selection of variables

XGBoost (extreme gradient boosting)

- Optimized distributed gradient boosting library
  https://xgboost.readthedocs.io/en/latest/
- Useful to take a detour to think about bias-variance trade-off
  http://www.r2d3.us/visual-intro-to-machine-learning-part-2/
- Want models which are predictive but also simple

XGBoost

- Boosting - new models added to correct the errors made by existing models
- Higher weights data points which are harder to predict
- Gradient boosting:
  - New models are added to predict residuals (errors)
  - Uses gradient descent to minimise the loss function when adding new models
- Trevor Hastie (a South African) on gradient descent

Hyper-parameter tuning

A number of ‘hyper-parameter’ need to be ‘tuned’ (or altered) to find the ones which are best for the model.

XGBoost hyper-parameters:

- **booster [default=gbtree]**

- Select the type of model to run at each iteration. It has 2 options:
    - gbtree: tree-based models
    - gblinear: linear models

- **silent [default=0]:**
    - Silent mode is activated is set to 1, i.e. no running messages will be printed.
    - It’s generally good to keep it 0 as the messages might help in understanding the model.

- **nthread [default to maximum number of threads available if not set]**
    - This is used for parallel processing and number of cores in the system should be entered
    - If you wish to run on all cores, value should not be entered and algorithm will detect automatically

Hyper-parameter tuning

- eta [default=0.3] Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient.

- Analogous to learning rate

- Makes the model more robust by shrinking the weights on each step

- Typical final values to be used: 0.01-0.2

- min_child_weight [default=1]

- Defines the minimum sum of weights of all observations required in a child.

- Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

- Too high values can lead to under-fitting hence, it should be tuned using CV.

Hyper-parameter tuning

● **max_depth [default=6]**
- The maximum depth of a tree.
- Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- Should be tuned using CV.
- Typical values: 3-10

● **max_leaf_nodes**
- The maximum number of terminal nodes or leaves in a tree.
- Can be defined in place of max_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.

Hyper-parameter tuning

- gamma [default=0]
- A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.

- Makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.

- max_delta_step [default=0]
- In maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative.
- Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced.

- subsample [default=1]
- Denotes the fraction of observations to be randomly samples for each tree.
- Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.
- Typical values: 0.5-1

Hyper-parameter tuning

- colsample_bytree [default=1]

- Denotes the fraction of columns to be randomly samples for each tree.

- Typical values: 0.5-1

- colsample_bylevel [default=1]

- Denotes the subsample ratio of columns for each split, in each level.

- Not often used

Hyper-parameter tuning

- lambda [default=1]

- L2 regularization term on weights (analogous to Ridge regression)

- This used to handle the regularization part of XGBoost.

- alpha [default=0]

- L1 regularization term on weight (analogous to Lasso regression)

- Can be used in case of very high dimensionality so that the algorithm runs faster when implemented

- scale_pos_weight [default=1]

- A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

Practical application

Use XGBoost to predict house prices

Resources

- The competition

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

- XGBoost

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/