Lec.2. XGBoost

Machine Learning II

Aidos Sarsembayev, IITU, Almaty, 2019

XGBoost

Before to get started w/ XGB you have to know:

1. What supervised learning is
2. Decision trees
3. Boosting

Supervised learning

- Relies on the labeled data
- Have some understanding of past behaviour

Supervised learning example

- Does a specific image contain a person’s face?

- **Training data: vectors of pixel values**
- **Labels: 1 or 0**

Supervised learning

In majority of the cases you have either a classification or a regression problem

- Classification does a binary or multiclass classification
- Regression predicts a real value

Supervised learning. Binary classification

When dealing with the binary supervised classification problem, AUC is one of the most common choices.

Supervised learning. Binary classification

When dealing with the binary supervised classification problem, AUC is one of the most common choices.

AUC – Area under the ROC (Receiver Operating Characteristic Curve) curve.

Supervised learning. Binary classification

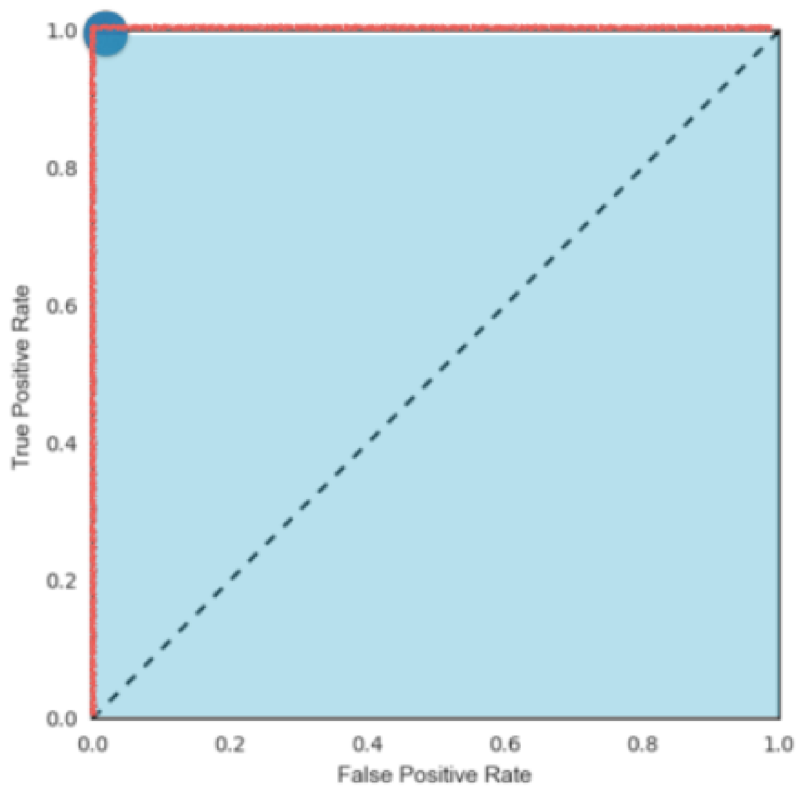
When dealing with the binary supervised classification problem, AUC is one of the most common choices.

AUC – Area under the ROC (Receiver Operating Characteristic Curve) curve.

AUC is simply the probability that a randomly chosen positive data point will have a higher rank than a randomly chosen negative data point for your learning problem

Supervised learning. AUC.

Larger area under the ROC curve = better model



Supervised learning. AUC.

Some useful readings:

- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://towardsdatascience.com/data-science-performance-metrics-for-everyone-4d68f4859eef

Supervised learning. Multi-class classification

When dealing with the multi-class supervised classification problem, accuracy is the common choice for metrics. This metrics uses confusion matrix.

Supervised learning. Multi-class classification

When dealing with the multi-class supervised classification problem, accuracy is the common choice for metrics. This metrics uses confusion matrix.

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Accuracy: $$\frac{tp + tn}{tp + tn + fp + fn}$$

Supervised learning

- In most of the cases supervised learning requires the data to be structured as a table of feature vectors.

Supervised learning

- In most of the cases supervised learning requires the data to be structured as a table of feature vectors.
- Where the features (a.k.a. predictors or attributes) are **either numeric or categorical**

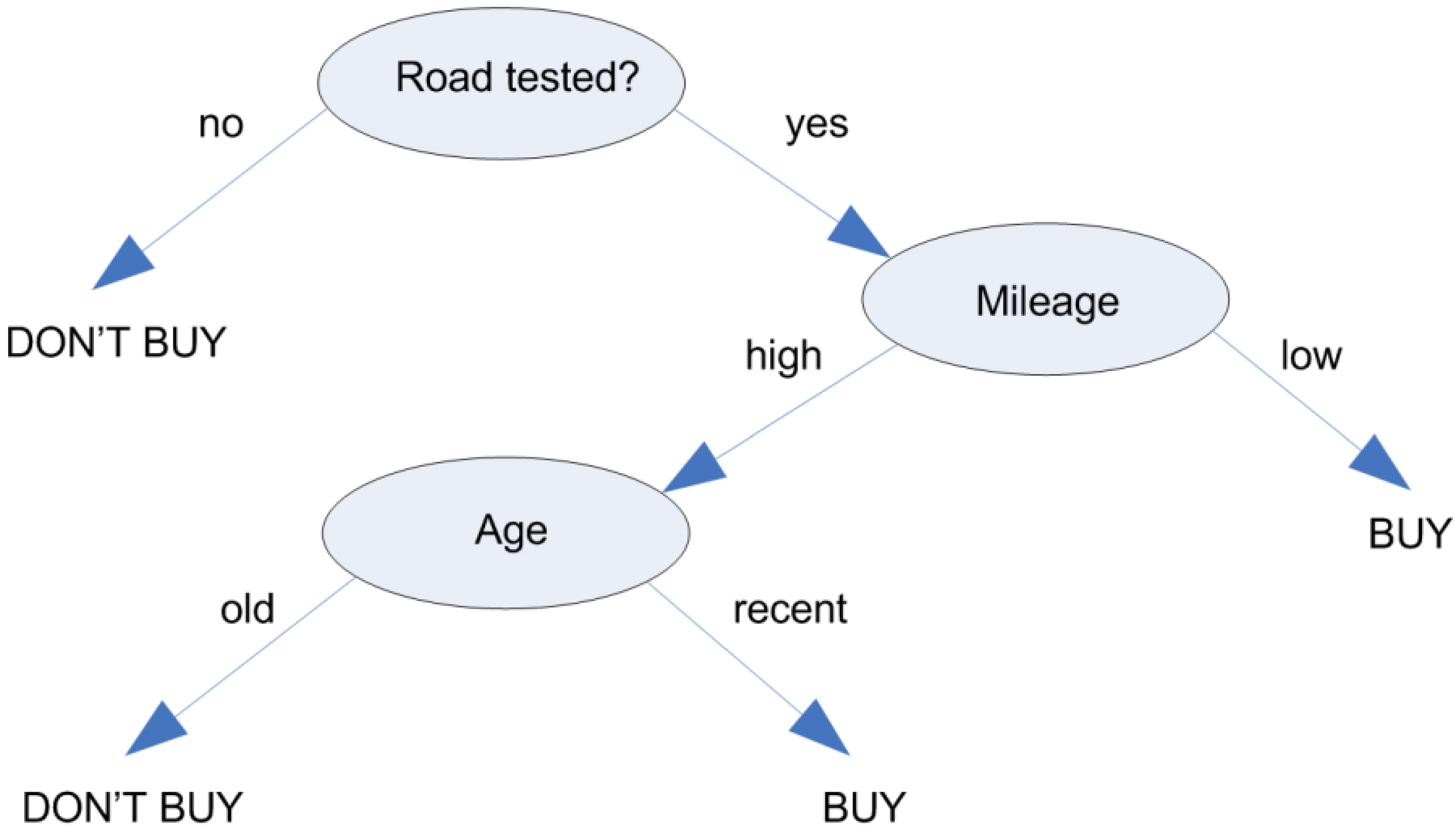
Supervised learning

- In most of the cases supervised learning requires the data to be structured as a table of feature vectors.
- Where the features (a.k.a. predictors or attributes) are **either numeric or categorical**
- Numeric features should be scaled (Z-scored, standardized)

Supervised learning

- In most of the cases supervised learning requires the data to be structured as a table of feature vectors.
- Where the features (a.k.a. predictors or attributes) are **either numeric or categorical**
- Numeric features should be scaled (Z-scored, standardized)
- Categorical features should be encoded ((label) one-hot)

Decision Trees



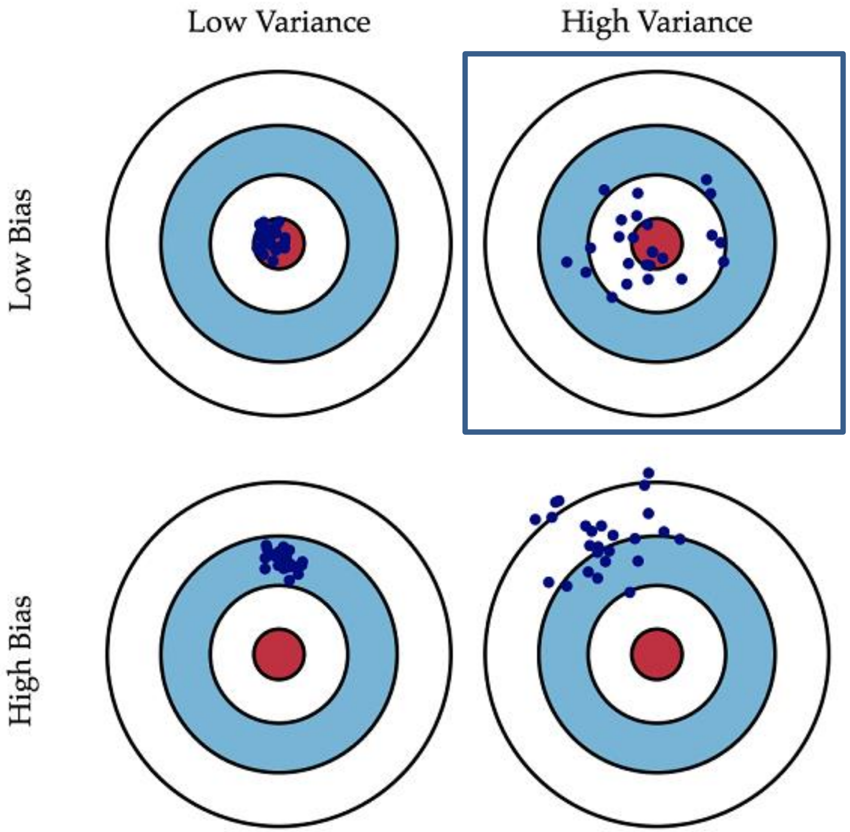
Decision trees as base learners

- Base learner - Individual learning algorithm in an ensemble algorithm
- Composed of a series of binary questions
- Predictions happen at the "leaves" of the tree

Decision trees and CART

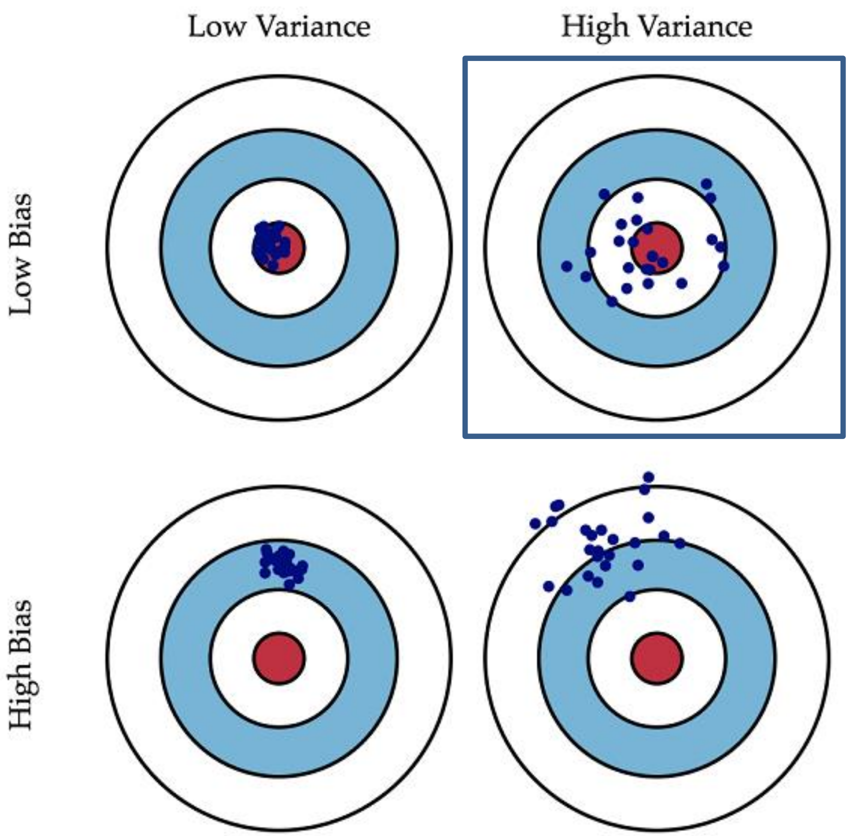
- Constructed iteratively (one decision at a time)
  - Until a stopping criterion is met

Individual decision trees tend to overfit



This means that they train good at some data you provide it with, but **<u>generalize very poorly</u>** on it. Therefore, it predicts not very well on new data.

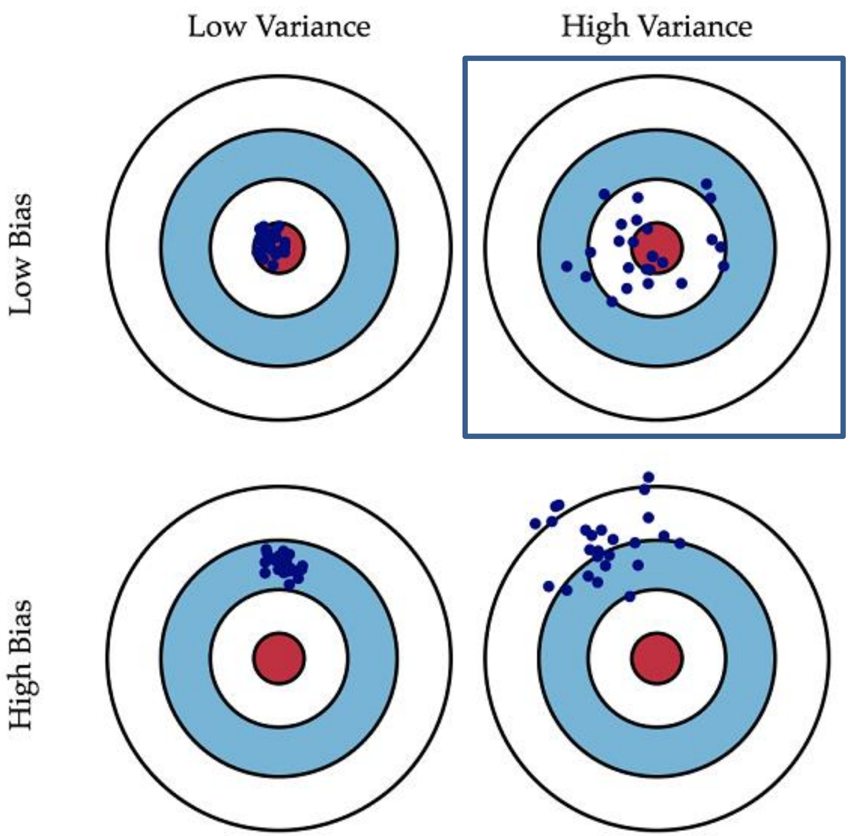
Individual decision trees tend to overfit



This means that they train good at some data you provide it with, but **<u>generalize very poorly</u>** on it. Therefore, it predicts not very well on new data.

XGBoost uses slightly different kind of a decision tree, called as CART. The individual decision trees leafs always contain decision values.

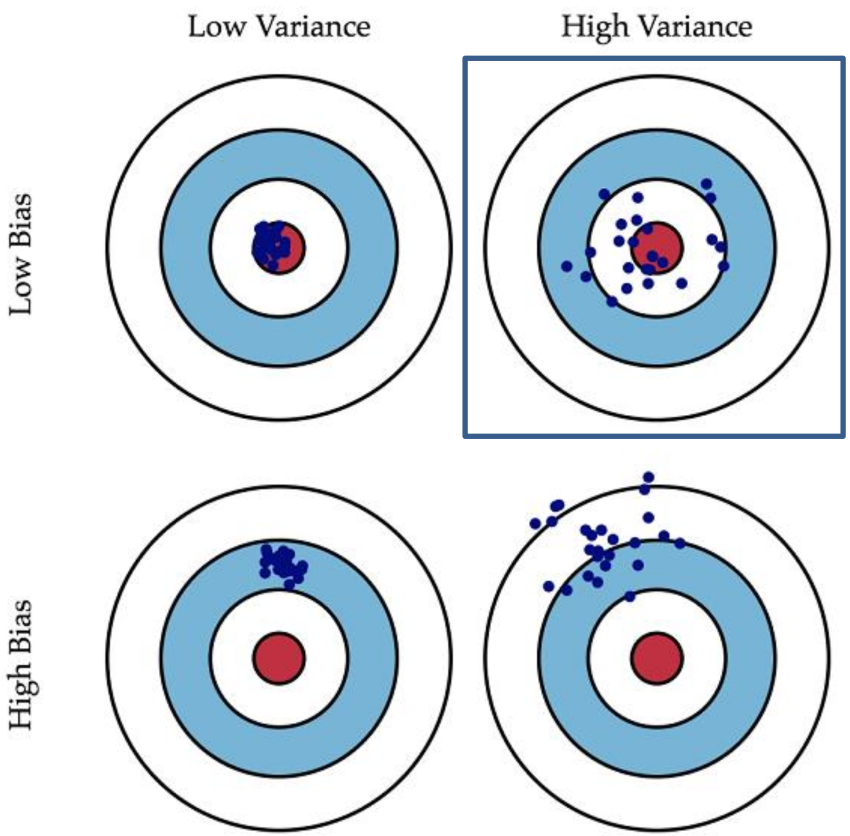
Individual decision trees tend to overfit



This means that they train good at some data you provide it with, but **<u>generalize very poorly</u>** on it. Therefore, it predicts not very well on new data.

XGBoost uses slightly different kind of a decision tree, called as CART. The individual decision trees leafs always contain decision values. Whereas CART contain only real-valued score in each leaf, regardless of whether they are used of regression or classification.

Individual decision trees tend to overfit



This means that they train good at some data you provide it with, but **<u>generalize very poorly</u>** on it. Therefore, it predicts not very well on new data.

XGBoost uses slightly different kind of a decision tree, called as CART. The individual decision trees leafs always contain decision values. Whereas CART contain only real-valued score in each leaf, regardless of whether they are used of regression or classification. The real-valued scores can then be thresholded to convert into categories for classification, if necessary

Boosting overview

- Not a specific machine learning algorithm

Boosting overview

- Not a specific machine learning algorithm
- Concept that can be applied to a set of machine learning models

    - "Meta-algorithm"

Boosting overview

- Not a specific machine learning algorithm
- Concept that can be applied to a set of machine learning models
  - "Meta-algorithm"
- Ensemble meta-algorithm used to convert many weak learners into a strong learner

Weak learners and strong learners

- Weak learner: ML algorithm that is slightly better than chance

 - Example: Decision tree whose predictions are slightly better than 50%

Weak learners and strong learners

- Weak learner: ML algorithm that is slightly better than chance

  - Example: Decision tree whose predictions are slightly better than 50%

- Boosting converts a collection of weak learners into a strong learner

Weak learners and strong learners

- Weak learner: ML algorithm that is slightly better than chance

  - Example: Decision tree whose predictions are slightly better than 50%

- Boosting converts a collection of weak learners into a strong learner
- Strong learner: Any algorithm that can be tuned to achieve good performance

How boosting is accomplished

- Iteratively learning a set of weak models on subsets of the data

How boosting is accomplished

- Iteratively learning a set of weak models on subsets of the data
- Weighing each weak prediction according to each weak learner's performance

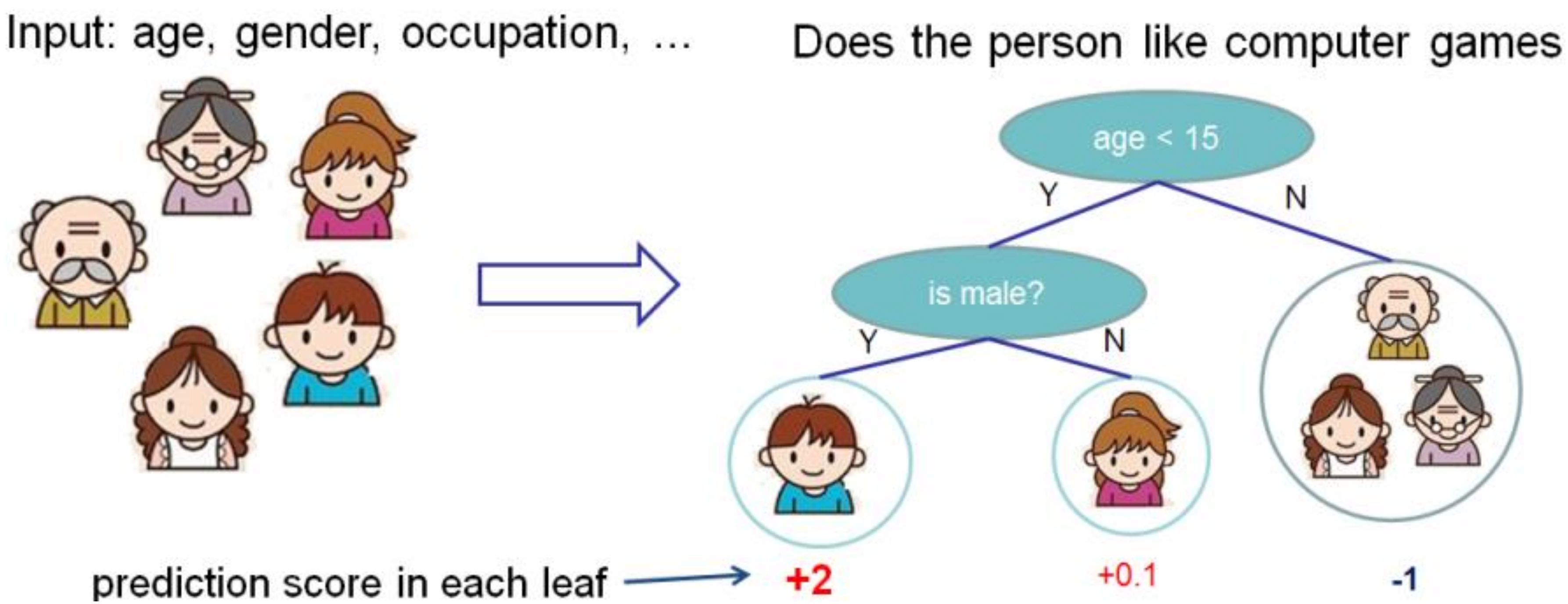
How boosting is accomplished

- Iteratively learning a set of weak models on subsets of the data
- Weighing each weak prediction according to each weak learner's performance
- Combine the weighted predictions to obtain a single weighted prediction

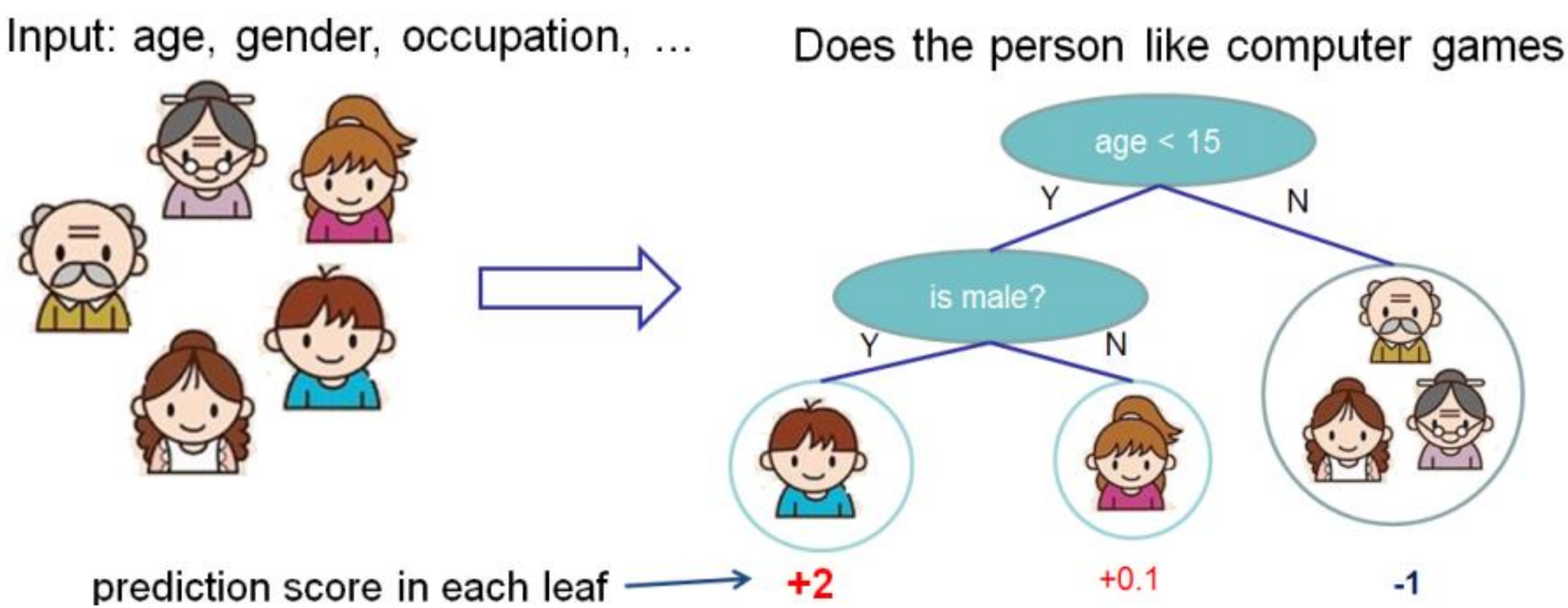
How boosting is accomplished

- Iteratively learning a set of weak models on subsets of the data
- Weighing each weak prediction according to each weak learner's performance
- Combine the weighted predictions to obtain a single weighted prediction
- ... that is much better than the individual predictions themselves!

Boosting example

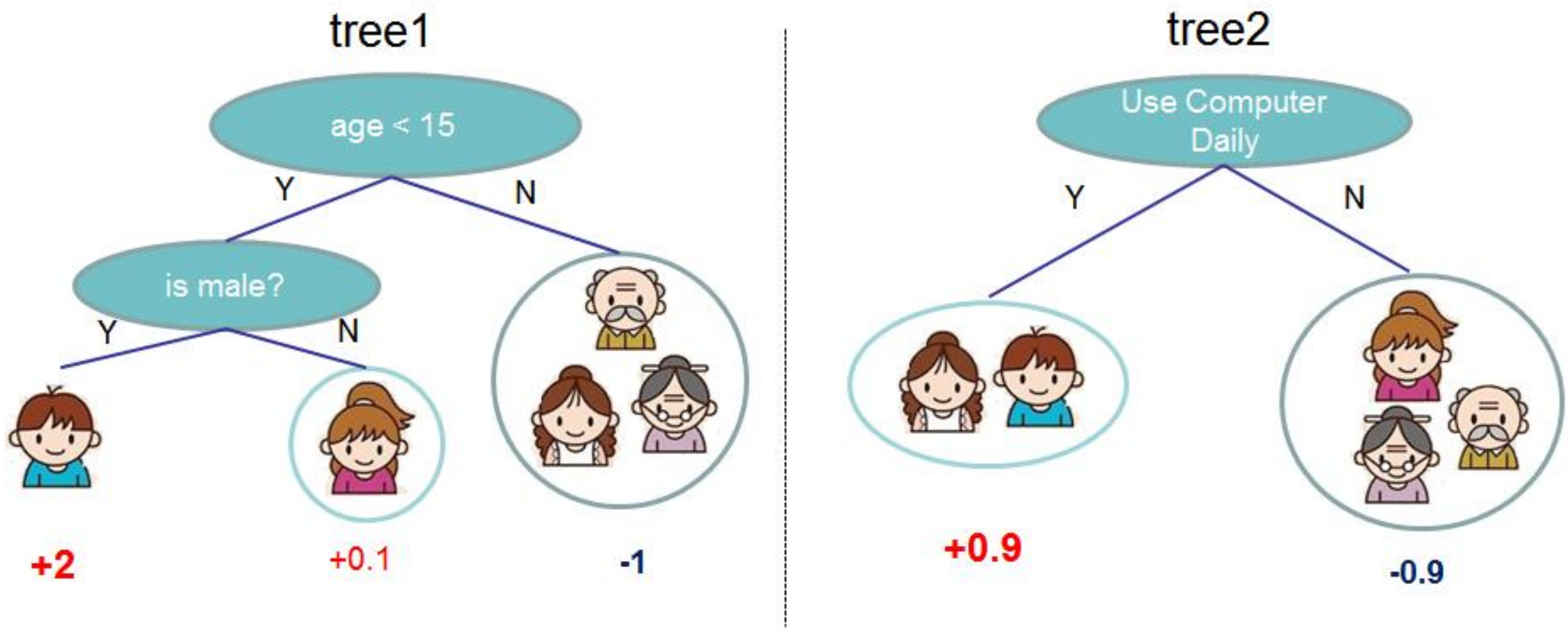


Boosting example



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

Boosting example



f() = 2 + 0.9= 2.9 f()= -1 - 0.9 = -1.9

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

Boosting example

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

where K is the number of trees, f is a function in the functional space F, and F is the set of all possible CARTs. The objective function to be optimized is given by

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where l is a loss and Ω is a complexity regularization

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

Model evaluation through cross-validation

- Cross-validation: Robust method for estimating the performance of a model on unseen data
- Generates many non-overlapping train/test splits on training data
- Reports the average test set performance across all data splits

Cross-validation in XGBoost example

```
In [1]: import xgboost as xgb

In [2]: import pandas as pd

In [3]: class_data = pd.read_csv("classification_data.csv")

In [4]: churn_dmatrix = xgb.DMatrix(data=churn_data.iloc[:,:-1],
        label=churn_data.month_5_still_here)

In [5]: params={"objective":"binary:logistic","max_depth":4}

In [6]: cv_results = xgb.cv(dtrain=churn_dmatrix, params=params, nfold=4,
        num_boost_round=10, metrics="error", as_pandas=True)

In [7]: print("Accuracy: %f" %((1-cv_results["test-error-mean"]).iloc[-1]))
Accuracy: 0.88315
```

XGBoost

- Optimized gradient-boosting machine learning library
- Originally written in C++
- Has API in several languages:
  - **Python**
  - R
  - Scala
  - Julia
  - Java

XGBoost is popular for

- Speed and performance
- Core algorithm is parallelizable
- State-of-the-art performance in many ML task comparing to many other single-algorithm models

XGBoost. Simple example

```
In [1]: import xgboost as xgb

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: from sklearn.model_selection import train_test_split

In [5]: class_data = pd.read_csv("classification_data.csv")

In [6]: X, y = class_data.iloc[:,:-1], class_data.iloc[:,-1]

In [7]: X_train, X_test, y_train, y_test= train_test_split(X, y,
        test_size=0.2, random_state=123)

In [8]: xg_cl = xgb.XGBClassifier(objective='binary:logistic',
        n_estimators=10, seed=123)

In [9]: xg_cl.fit(X_train, y_train)

In [10]: preds = xg_cl.predict(X_test)

In [11]: accuracy = float(np.sum(preds==y_test))/y_test.shape[0]

In [12]: print("accuracy: %f" % (accuracy))
accuracy: 0.78333
```

XGBoost. Simple example

```
In [1]: import xgboost as xgb

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: from sklearn.model_selection import train_test_split

In [5]: class_data = pd.read_csv("classification_data.csv")

In [6]: X, y = class_data.iloc[:,:-1], class_data.iloc[:,-1]

In [7]: X_train, X_test, y_train, y_test= train_test_split(X, y,
        test_size=0.2, random_state=123)

In [8]: xg_cl = xgb.XGBClassifier(objective='binary:logistic',
        n_estimators=10, seed=123)

In [9]: xg_cl.fit(X_train, y_train)

In [10]: preds = xg_cl.predict(X_test)

In [11]: accuracy = float(np.sum(preds==y_test))/y_test.shape[0]

In [12]: print("accuracy: %f" % (accuracy))
accuracy: 0.78333
```

It uses the fit/predict pattern

XGBoost. Simple example

```
In [1]: import xgboost as xgb

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: from sklearn.model_selection import train_test_split

In [5]: class_data = pd.read_csv("classification_data.csv")

In [6]: X, y = class_data.iloc[:,:-1], class_data.iloc[:,-1]

In [7]: X_train, X_test, y_train, y_test= train_test_split(X, y,
        test_size=0.2, random_state=123)

In [8]: xg_cl = xgb.XGBClassifier(objective='binary:logistic',
        n_estimators=10, seed=123)

In [9]: xg_cl.fit(X_train, y_train)

In [10]: preds = xg_cl.predict(X_test)

In [11]: accuracy = float(np.sum(preds==y_test))/y_test.shape[0]

In [12]: print("accuracy: %f" % (accuracy))
accuracy: 0.78333
```

To achieve better results you have to tune the parameters

When to use XGBoost

- You have a large number of training samples
  - Greater than 1000 training samples and less 100 features
  - The number of features < number of training samples
- You have a mixture of categorical and numeric features
  - Or just numeric features

When to NOT use XGBoost

- Image recognition
- Computer vision
- Natural language processing and understanding problems
- When the number of training samples is significantly smaller than the number of features

To be continued.

Thanks!