

Een introductie tot Gradient Boosting

Pipple Lecture 5 - 15 mei 2019

Programma

- › Wat is gradient boosting?
- › Gradient boosting met XGBoost
- › Pre-processing en training tips
- › Hands-on ML competitie
- › Bekendmaking winnaars

Deze lecture is voornamelijk gericht op de intuïtie en de toepassing

Wat is gradient boosting?

Is een effectief machine learning algoritme. Het is toe te passen op:

- › Regressie
- › Classificatie
- › Ranking

Wordt veel gebruikt in machine learning competities

Wat is gradient boosting?

Ensemble algoritmes

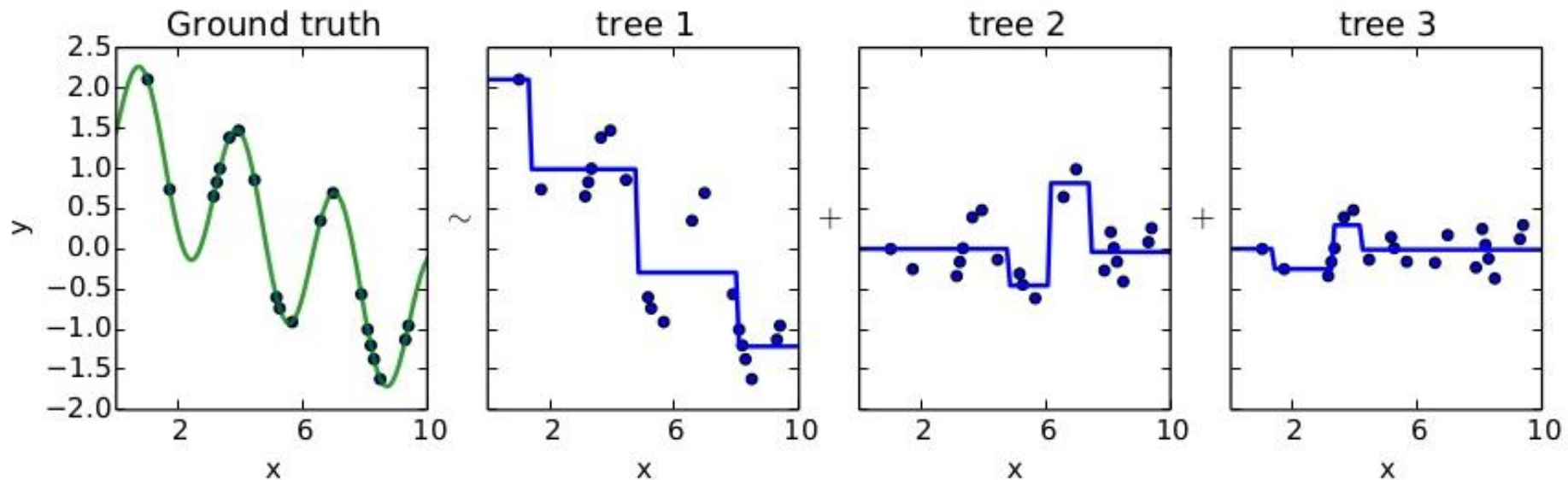
- › Bagging
- › Boosting
- › Random Forest

Een ensemble is een samenvoeging van verschillende learners, bijvoorbeeld:

- › Decision Tree
- › Regression Tree

Intuïtie gradient boosting

Intuïtie: train een nieuw model op de fouten (residuals) van voorgaande modellen, en voeg dit nieuwe model toe aan de voorgaande modellen



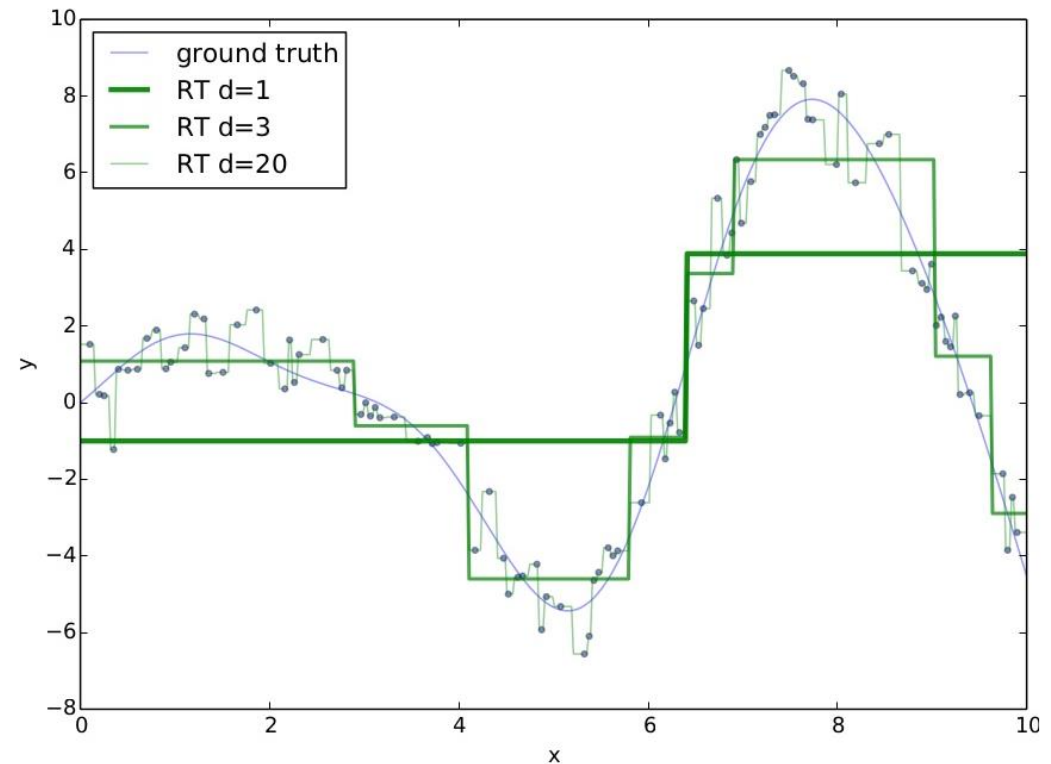
Wat is gradient boosting?

Gradient boosting bestaat uit verschillende onderdelen:

- › Weak learners
 - › Regression Tree
 - › Classification Tree
- › Loss function
 - › Wordt geoptimaliseerd
- › Additive model
 - › Methode waarop de weak learners gecombineerd worden

Regression tree

- › Verbeteren door meer splits
- › In gradient boosting beperkingen
 - › Weinig splits (vroeger stumps)
 - › Learners blijven weak



Loss function

Residuals:

- › Een functie meestal op basis van errors / residuals

Veel gebruikte functies zijn:

- › **RMSE** : square Root of Mean Squared Error
- › **MAE** : Mean Absolute Error

Additive model

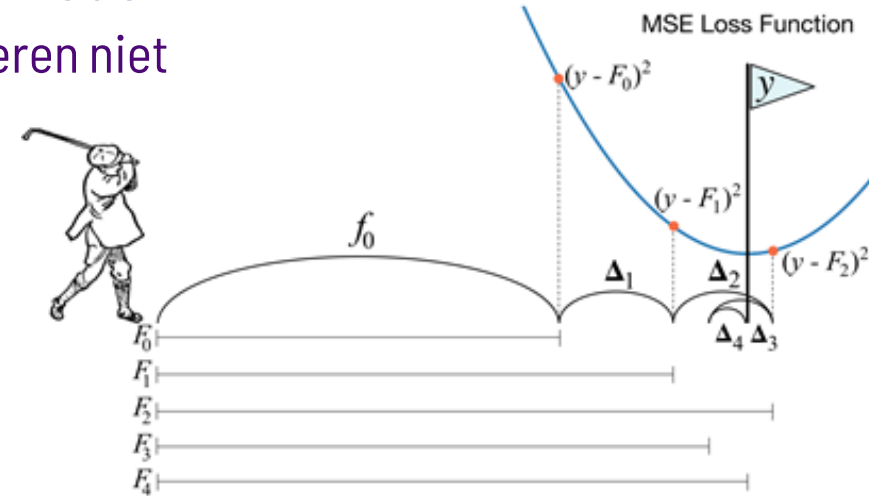
Stapsgewijs toevoegen en verbeteren van model

- › Bij elke stap wordt gekeken welke tree toegevoegd dient te worden
- › Minimaliseren van loss function d.m.v. gradient descent op residuals

Decision tree wordt toegevoegd aan huidig model

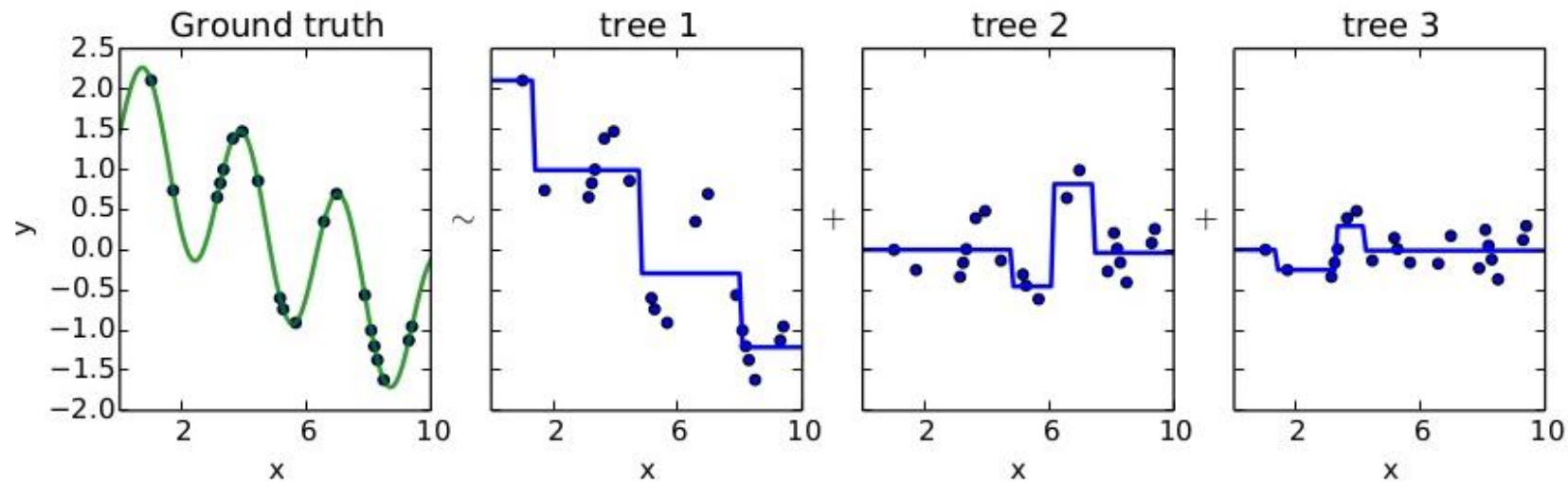
- › Huidige elementen (trees) van model veranderen niet
- › Model is som van elementen
- › Toevoeging hoeft niet lineair te zijn

Gebruikt gradient descent om loss functie te minimaliseren



Additive model

Meer verklaren door extra trees toe te voegen:



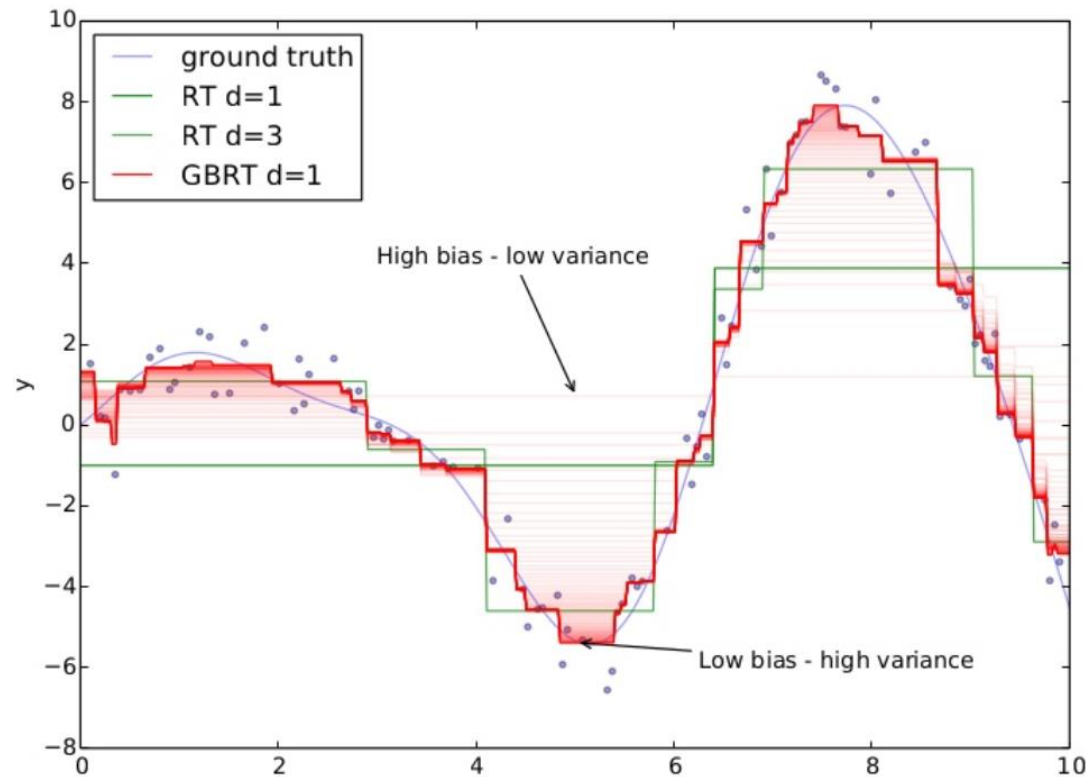
Gradient Boosting vs. Regression Tree

Regression Tree

- › Aantal splits

Gradient Boosting

- › Aantal trees



Gradient Boosting vs. Random Fores

RF berekenen van meerdere trees tegelijkertijd

- › Voordeel voor distribueren

GB berekenen van meerdere trees na elkaar

- › Kleine trees kosten minder tijd

Over het algemeen:

- › GB > RF > Bagging > RT

GB meer tuning nodig

- › Kans op overfitting

Gradient boosting met XGBoost

De XGBoost library

- › eXtreme Gradient Boosting
- › Beschikbaar in C++, Java, Python, R, en Julia
- › Veel documentatie beschikbaar

dmlc
XGBoost



CatBoost



Microsoft

Gradient boosting met XGBoost

Basis model

```
import pandas as pd

# Maak train en validatie matrixen aan
dtrain = xgb.DMatrix(data=train_X.values, label=train_y.values, feature_names=train_X.columns)
dvalid = xgb.DMatrix(data=valid_X.values, label=valid_y.values, feature_names=valid_X.columns)

# Stel de verschillende parameters in
param = {'eta': 0.01, 'max_depth': 10, 'silent': 1, 'objective': 'reg:linear', 'eval_metric': 'mae'}

num_round = 100

bst = xgb.train(param, dtrain, num_round)

# Maak nieuwe voorspellingen op validatie set
preds = bst.predict(dvalid)
```

XGBoost parameters (1/3)

- › **eta** staat voor hoeveel elke decision tree bijdraagt aan de uiteindelijke voorspelling [default=0.3]
- › **gamma** is de minimale toegevoegde waarde van een nieuwe split [default=0]
- › **max_depth** is de maximale diepte van elke decision tree [default=6]
- › **subsample** is de ratio van random training samples waarop elke decision tree wordt getraind [default=1.0]
- › **colsample_bytree** en **colsample_bylevel** geven aan welke ratios van de kolommen worden gebruikt per tree en per level [default=1.0]

XGBoost parameters (2/3)

- › **objective** is het type loss functie die we minimalizeren, bijv. binary:logistic, multi:softmax, multi:softprob [default=reg:linear]
- › **eval_metric** is de metric waarmee we evalueren hoe goed onze voorspellingen zijn op de validatie data [default according to objective]
- › **seed** staat voor welke random seed we gebruiken [default=0]
- › **num_round** geeft aan hoeveel decision trees we trainen

XGBoost parameters (3/3)

```
# De verschillende parameters
param = {'eta':0.01,
        'max_depth': 10,
        'min_child_weight': 1,
        'gamma': 0,
        'subsample': 1.0,
        'colsample_bytree': 1.0,
        'colsample_bylevel': 1.0,
        'silent': 1,
        'objective': 'reg:linear',
        'eval_metric': 'mae',
        'seed': 42}

# Hoeveel trees we trainen
num_round = 100
```

Gradient boosting met XGBoost

Overfitting

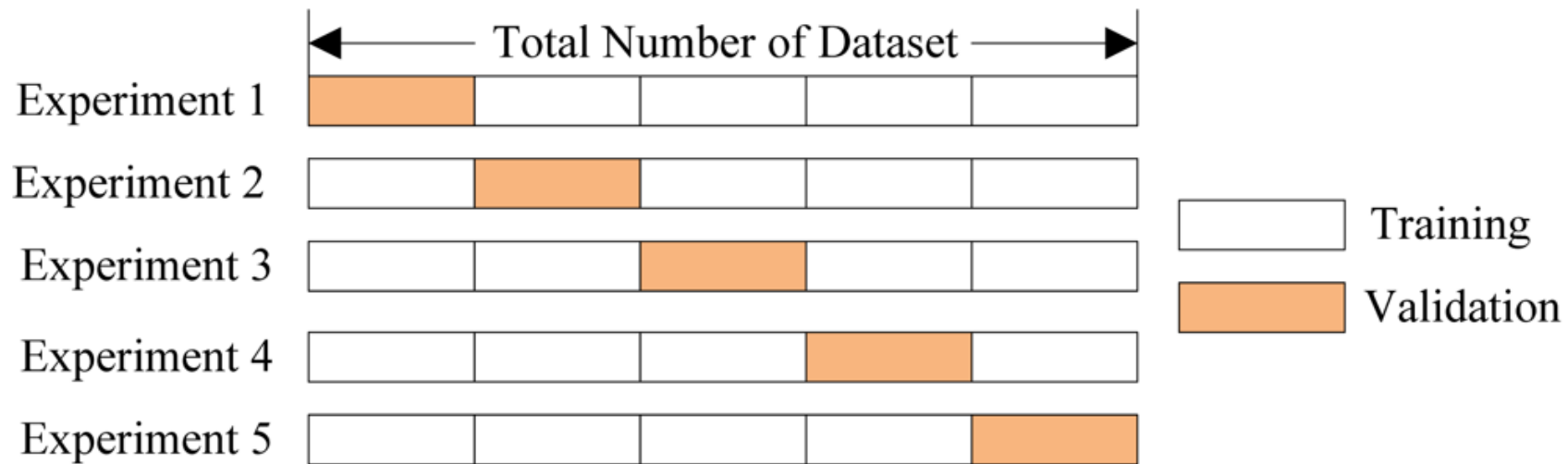
- › Hoge training accuracy, lage validatie accuracy

Drie manieren om overfitting te voorkomen:

- › Model complexiteit verminderen met **max_depth** en **gamma**
- › Randomness toevoegen met **subsample**, **colsample_bytree** en **colsample_bylevel**
- › De **eta** verminderen en **num_round** verhogen

Gradient boosting met XGBoost

Cross-validation



Gradient boosting met XGBoost

Cross-validation

```
# Gebruik CV om overfitting te voorkomen
cv_results = xgb.cv(param, dtrain, num_boost_round=num_round,
                    nfold=5, metrics={'mae'}, early_stopping_rounds=10)
```

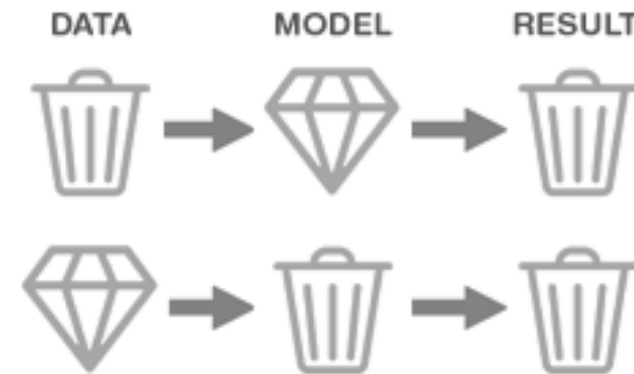
cv_results

	train-mae-mean	train-mae-std	test-mae-mean	test-mae-std
0	6.256053	0.005370	6.256020	0.021399
1	6.193581	0.005316	6.193493	0.021403
2	6.131712	0.005190	6.131599	0.021508
3	6.070492	0.005191	6.070339	0.021497

Gradient boosting kan niet alles...

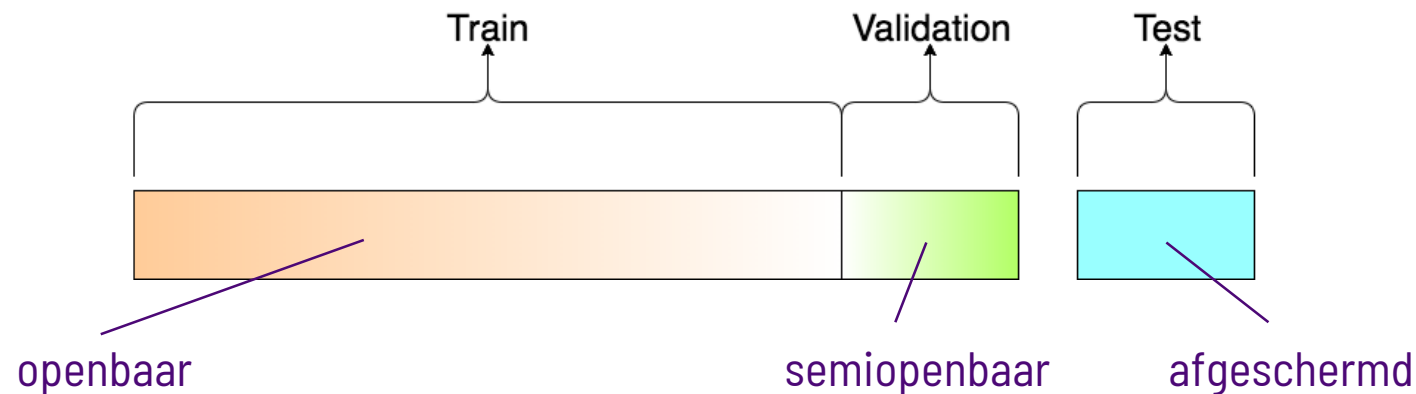
- › 'Garbage in, garbage out'
- › Data-inzicht en pre-processing zijn net zo belangrijk
- › Tips van Kaggle winnaars

The 'garbage in, garbage out' principle:



Kaggle competities

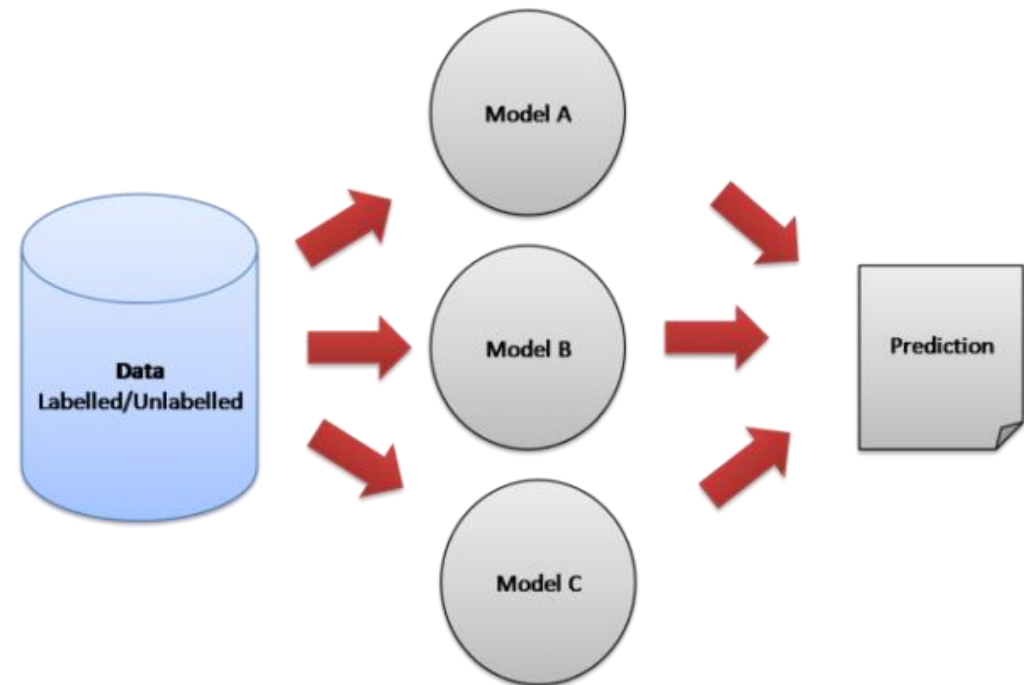
- › Bouw model op openbare training data
- › Maak voorspellingen op semiopenbare validatie data
- › Validatie data score staat op public leaderboard
- › Test data score blijft geheim tot einde competitie



Tip 1: stacken van modellen

Op basis van:

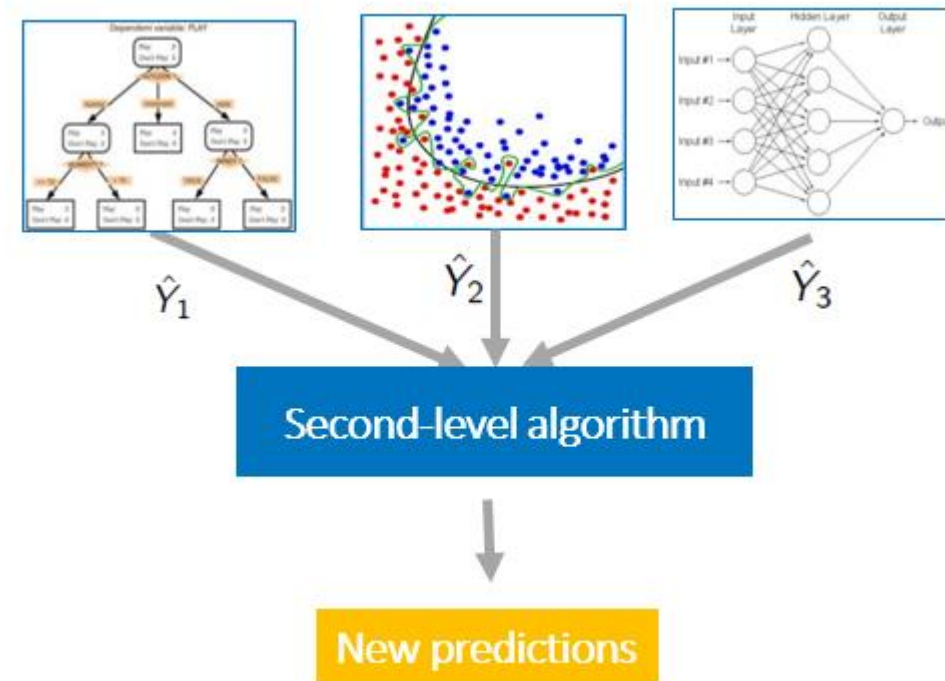
- › Verschillende type modellen
- › Verschillende parameters
- › Verschillende features
- › Verschillende training sets



Tip 2: combineren van voorspellingen

Meest gebruikt:

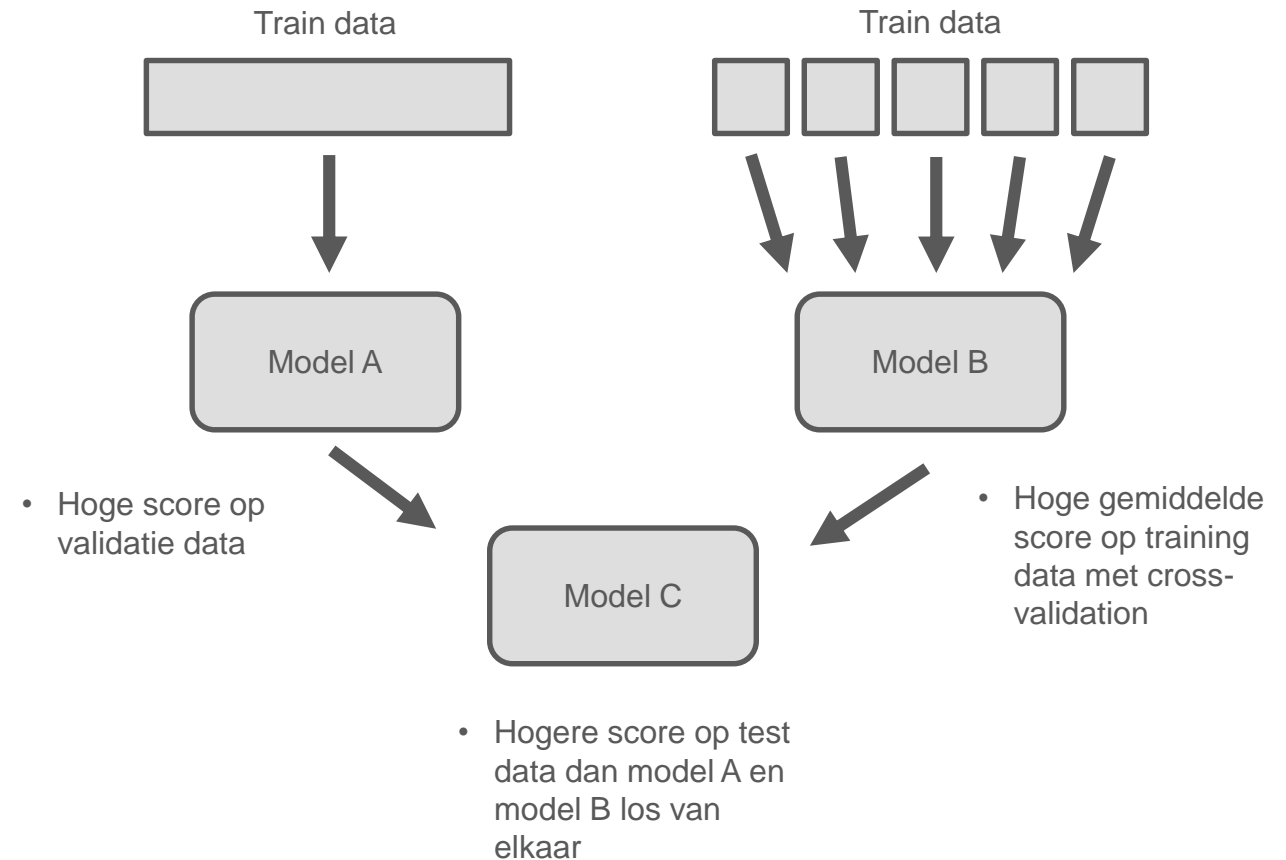
- › (Un)weighted average
- › Majority vote
- › Trainen van nieuw model op de voorspellingen van voorgaande modellen



Tip 3: cross-validation

Als test data en validatie data verschillend verdeeld zijn:

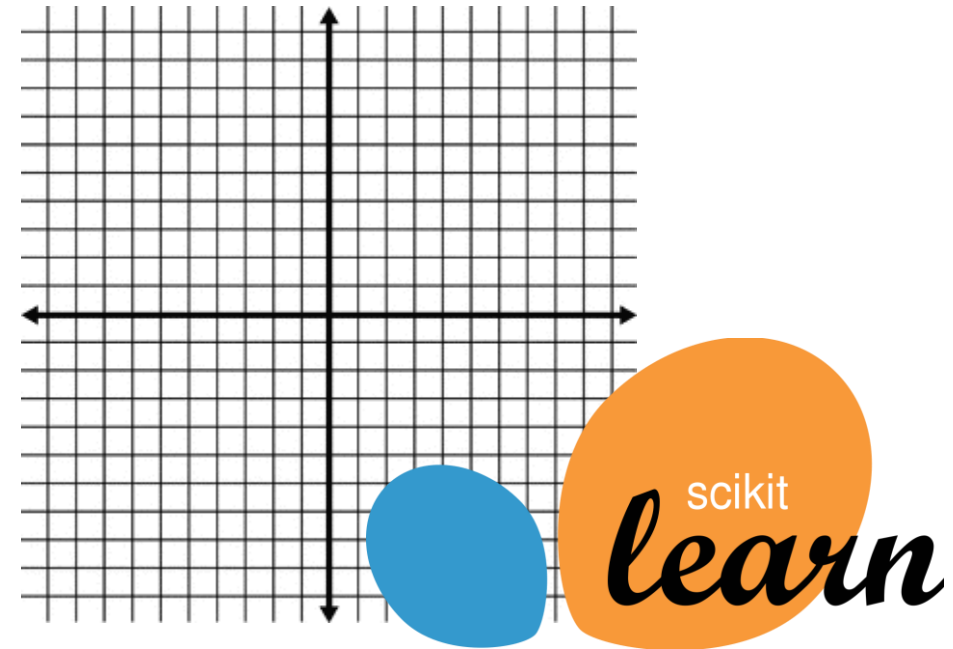
- › Train model met hoge score op de validatie data
- › Train model met beste gemiddelde score op cross-validation over training set
- › De combinatie geeft beter resultaat op test data



Tip 4: parameter tuning

Grid search

- › Automatisch zoeken naar de beste hyperparameters voor een model
- › Verschillende parameter combinaties worden getest
- › Kan met scikit-learn



Tip 4: parameter tuning

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

cv_params = {'eta':[0.2, 0.3, 0.4],
             'n_estimators': [10, 50, 200],
             'max_depth':[3, 6, 9],
             'subsample':[0.5, 0.75, 1.0],
             'colsample_bytree':[0.5, 0.75, 1.0],
             'colsample_bylevel':[0.5, 0.75, 1.0]}

ind_params = {'seed':42,
              'objective':'reg:linear',
              'eval_metric':'mae',
              'silent':1}

optimized_GBM = GridSearchCV(xgb.XGBRegressor(**ind_params),
                             cv_params,
                             scoring = 'neg_mean_absolute_error',
                             cv = 5,
                             n_jobs = -1)

optimized_GBM.fit(valid_X, valid_y)

optimized_GBM.best_params_
```

xgboost aanpasbare parameters

xgboost vaste parameters

grid search parameters

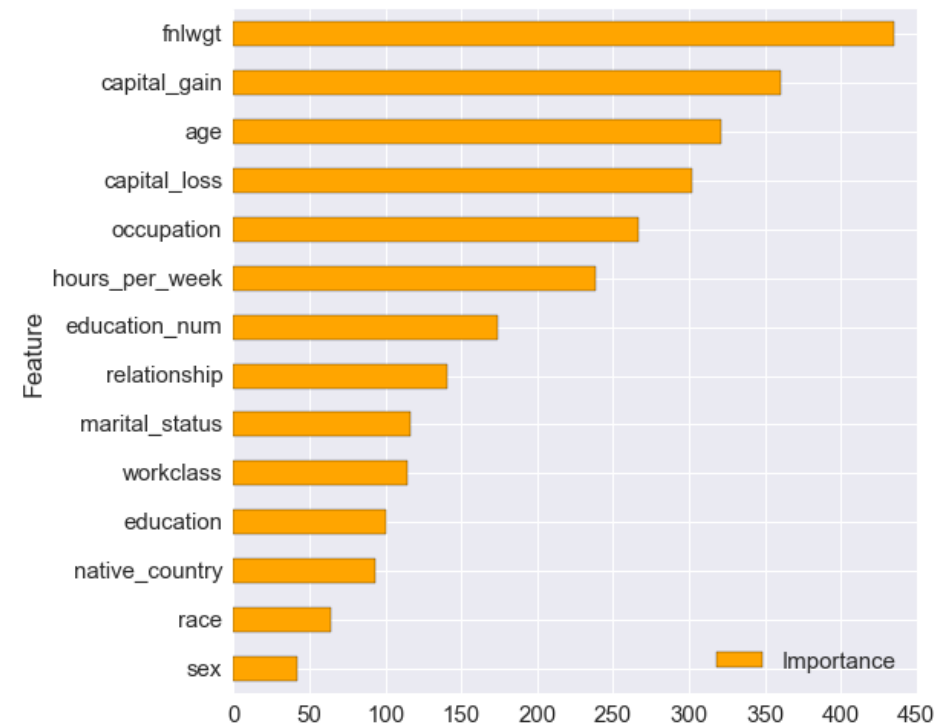
Tip 5: feature engineering & selection

Feature engineering

- › Toevoegen van nieuwe features
- › Kan relaties in kaart brengen die nog niet in de data worden meegenomen

Feature selection

- › Verwijderen van onbelangrijke variabelen
- › Kan makkelijk in XGBoost



Tip 5: feature engineering & selection

```
import xgboost as xgb
import pandas as pd

# Maak train en validatie matrixen aan
dtrain = xgb.DMatrix(data=train_X.values, label=train_y.values, feature_names=train_X.columns)
dvalid = xgb.DMatrix(data=valid_X.values, label=valid_y.values, feature_names=valid_X.columns)

# Stel de verschillende parameters in
param = {'eta': 0.01, 'max_depth': 10, 'silent':1, 'objective': 'reg:linear', 'eval_metric': 'mae'}

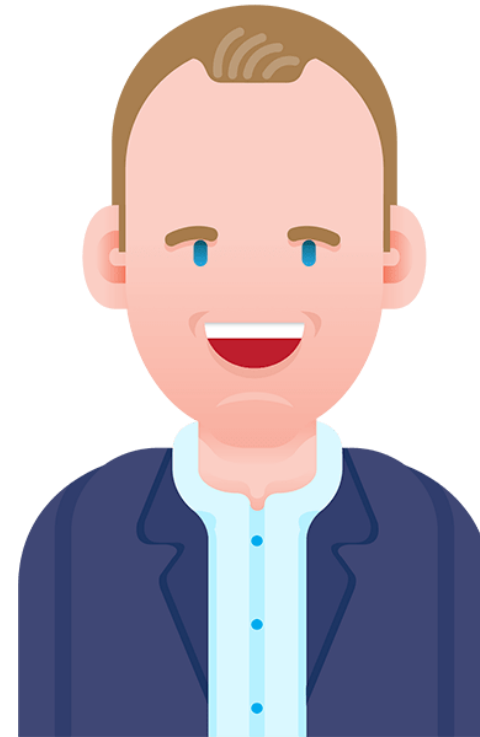
num_round = 100

bst = xgb.train(param, dtrain, num_round)

xgb.plot_importance(bst)
```

Speciaal bier competitie

- › Bart houdt van speciaal bier
- › Bart heeft een bier app
- › Bart wilt weten welke (nieuwe) bieren hij wel of niet lekker zal vinden
- › Bart vraagt jullie hulp



Speciaal bier competitie

- › Maak een XGBoost model dat voorspelt welke score Bart geeft aan een bier
- › Gebruik één of meerdere modellen
- › Winnaar krijgt een speciaalbier pakket



Info over de bier data (1/3)

- › **calorien:** Totaal aantal calorieën in het bier
- › **dichtheid:** De dichtheid van het bier (kg/dm^3)
- › **gedronken:** Aantal keren dat Bart het bier gedronken heeft
- › **is_belgisch:** Of het bier uit België komt (0=nee, 1=ja)
- › **is_speciaal:** Of het bier een speciaalbier is (0=nee, 1=ja)
- › **pct_alcohol:** Percentage alcohol in het bier (%)
- › **pct_eiwitten:** Percentage eiwitten in het bier (%)

Info over de bier data (2/3)

- › **pct_gist:** Percentage gist in het bier (%)
- › **pct_hop:** Percentage hop in het bier (%)
- › **recensie:** Gemiddelde score in de bier app (0.0-10.0)
- › **suiker:** Hoeveelheid suiker in het bier (g/100ml)
- › **water_kw:** Score voor de kwaliteit van het water (0.0-2.0)
- › **zuur:** Score voor hoe zuur het bier is (0.0-2.0)
- › **beoordeling:** Score voor hoe lekker Bart het bier zal vinden (1 t/m 10)

Info over de bier data (3/3)

- › **Training set:** 3898 rijen (60%)
- › **Validatie set:** 1299 rijen (20%)
- › **Test set:** 1300 rijen (20%)

	calorien	dichtheid	gedronken	is_belgisch	is_speciaal	pct_alcohol	pct_eiwitten	pct_gist	pct_hop	recensie	suiker	water_kw	zuur	beoordeling
0	152.0	0.9935	26.0	1.0	1.0	6.7	10.00	0.98	4.4	6.0	1.20	0.47	0.39	7.0
1	126.0	0.9943	17.0	1.0	1.0	5.8	9.10	1.06	11.1	4.0	4.50	0.46	0.26	6.0
2	26.0	0.9965	13.0	0.0	0.0	6.0	10.00	1.84	10.0	5.7	2.20	0.47	0.04	6.0
3	197.0	0.9980	41.0	1.0	1.0	8.5	9.80	1.48	5.2	2.3	16.20	0.50	0.21	4.0
4	45.0	0.9914	9.0	1.0	1.0	6.7	10.50	0.68	5.4	2.8	3.60	0.40	0.33	7.0
5	140.0	0.9914	44.0	1.0	1.0	6.4	10.70	0.70	2.1	3.5	1.10	0.55	0.29	8.0

Meten van de resultaten

Train, validatie, test

- › Openbare training set
- › Openbare validatie set
- › Test set features beschikbaar
- › Test set labels niet beschikbaar
- › Volgorde van voorspellingen moet hetzelfde blijven

Metric

- › Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

- › Absolute error per voorspelling
- › Elke error telt even zwaar mee

Stuur je voorspellingen (als csv) en team naam naar gijs@pipple.nl!

Hulpmiddelen

Er is een Jupyter en Colab notebook beschikbaar (inclusief de data):

- › Te downloaden via <https://github.com/PippleNL/gradient-boosting-lecture>

EXAMPLES

RECENT

GOOGLE DRIVE

GITHUB

UPLOAD


Enter a GitHub URL or search by organization or user



☐ Include private repos


PippleNL



Repository: [PippleNL/gradient-boosting-lecture](#) Branch: [master](#)

Path

 gradient-boosting-colab.ipynb

 gradient-boosting-notebook.ipynb

Deadline

Deadline voor inzendingen:

- › Om 20:30
- › Mailen naar gijs@pipple.nl



Let's Pipple!

pipple
DATA SCIENCE
WITH PURPOSE

