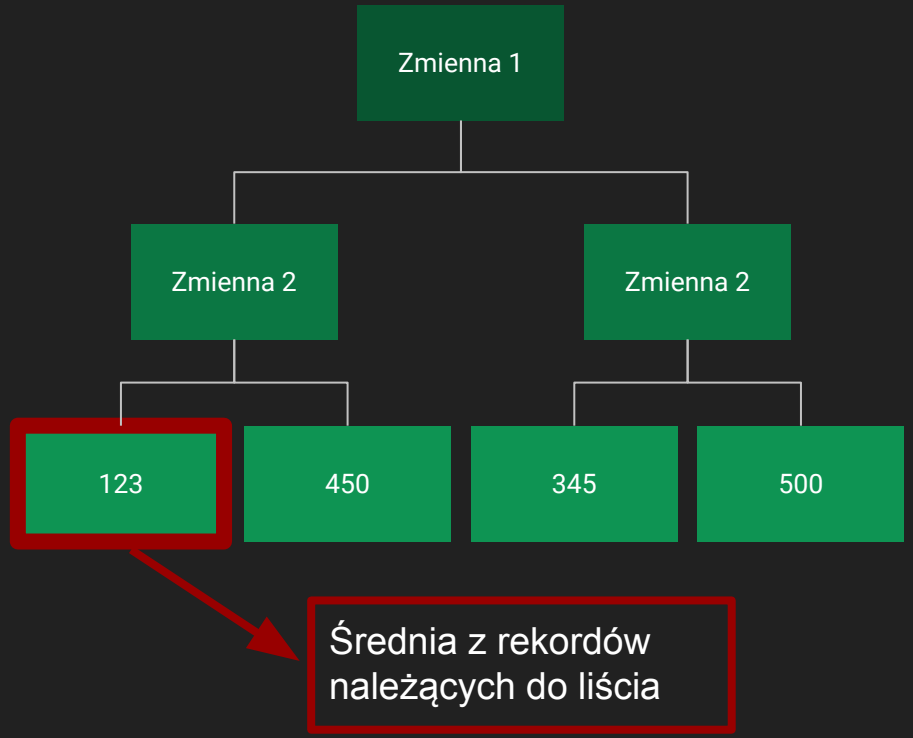
Random forest

Powtórka z drzew decyzyjnych

Drzewa regresyjne

Chcemy poznać wartość zmiennej ciągłej



Drzewa regresyjne

W każdym rozgałęzieniu zbiór danych jest dzielony przez taką zmienną i stałą, by zminimalizować SSE:

$$\texttt{minimize}\left\{ SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2 \right\}$$

Dzielenie jest tak długo powtarzane aż pewne kryteria zostaną spełnione.

Wynikiem jest drzewo, które ma duże prawdopodobieństwo do overfittingu i złej predykcyjności na danych, których jeszcze nie widziało.

Wybieramy rozgałęzienie

Chcemy wybrać zmienną, która najlepiej rozdziela dane, liczymy zatem najmniejsze SSE dla każdej.

1. Dla zmiennych kategorycznych to poszczególne możliwe do przyjęcia wartości rozdzielają różne gałęzie. Wybieramy ich podzbiór, który najlepiej minimalizuje SSE dla tej zmiennej.
2. Dla zmiennych ciągłych:
    i. sortujemy zmienne
    ii. obliczamy średnią dla sąsiadujących rekordów
    iii. obliczamy SSE dla wszystkich uzyskanych wartości [uz]. Do lewego dziecka trafiają wartości ‘<’ uz a do prawego ‘>=’ uz
    iv. wybieramy najlepszy stałą najlepiej minimalizującą SSE.
3. Wybieramy zmienną i stałą/ podzbiór wartości dające najmniejsze SSE

Cost complexity criterion (α)

W ten sposób możemy znaleźć najoptymalniejsze drzewo i polepszyć wyniki dla nowych danych.

$$\texttt{minimize}\{SSE + \alpha|T|\}$$

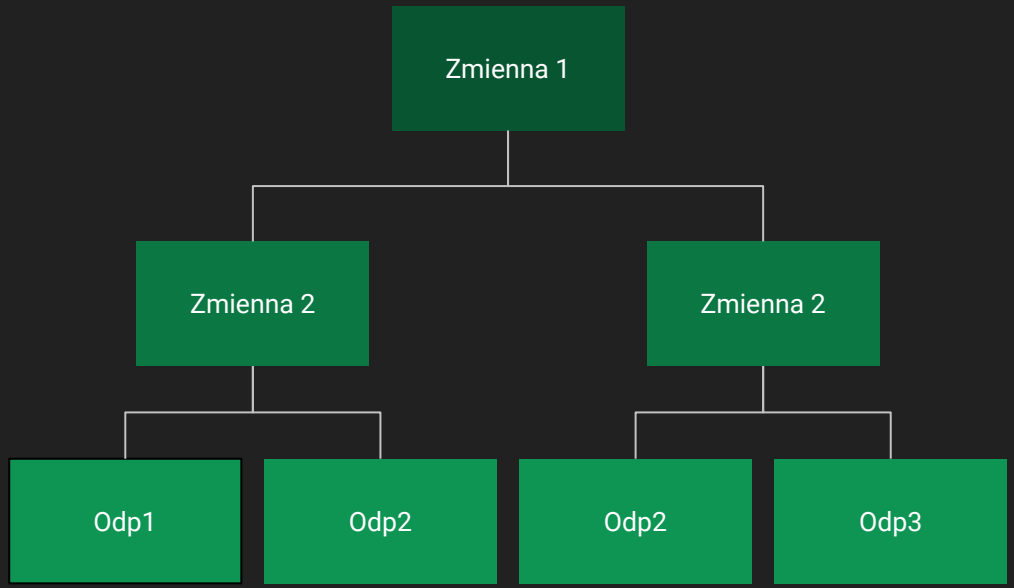
|T| - liczba liści w drzewie

Drzewa klasyfikacyjne

$$Gini = 1 - \sum_j p_j^2$$

p_j = prawdopodobieństwo, że próbka należy do danej klasy w wybranym węźle

Zupełny indeks Gini- przy wybieraniu rozgałęzienia jest to średnia ważona z tymczasowych lub już ostatecznych liści.
Wybieramy takie rozgałęzienia, by miał on jak najmniejszą wartość

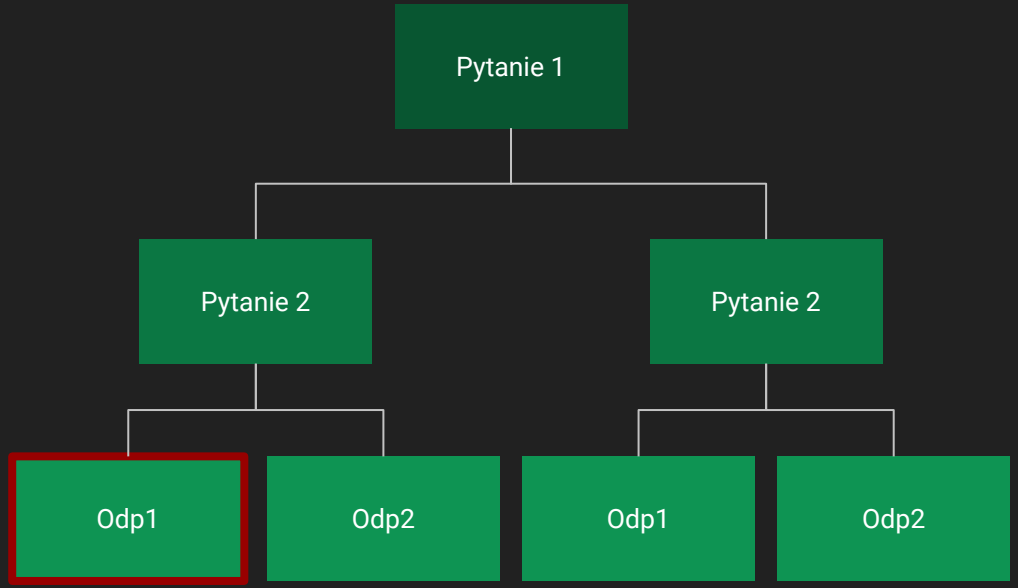


Wybieramy rozgałęzienie.

Chcemy wybrać zmienną, która najlepiej rozdziela dane, liczymy zatem najmniejsze GINI dla każdej.

1. Dla zmiennych kategorycznych to poszczególne możliwe do przyjęcia wartości rozdzielają różne gałęzie. Wybieramy najlepszy ich podzbiór, który najlepiej minimalizuję GINI.
2. Dla zmiennych ciągłych:
    i. sortujemy zmienne malejąco
    ii. obliczamy średnią dla sąsiadujących rekordów
    iii. obliczamy Gini dla wszystkich uzyskanych wartości [uz]. Do lewego dziecka trafiają wartości ‘<’ uz a do prawego ‘>=’ uz
    iv. wybieramy najlepszy podział, zmienną i Gini dla nich
3. Wybieramy zmienną, która najlepiej rozdziela, czyli taką która jest “najczystsza” (o tym właśnie informuję nas Gini)

Przykład z życia

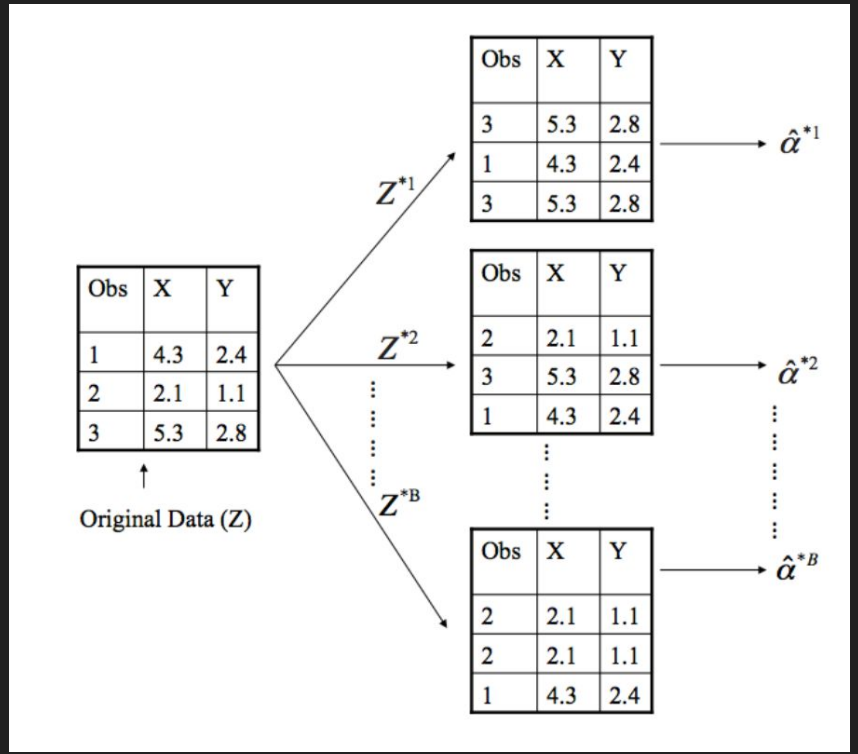


Bootstrapping

Jest to utworzenie próbki z źródłowego zbioru danych o:

- tej samej wielkości
- z możliwością powtarzania się rekordów

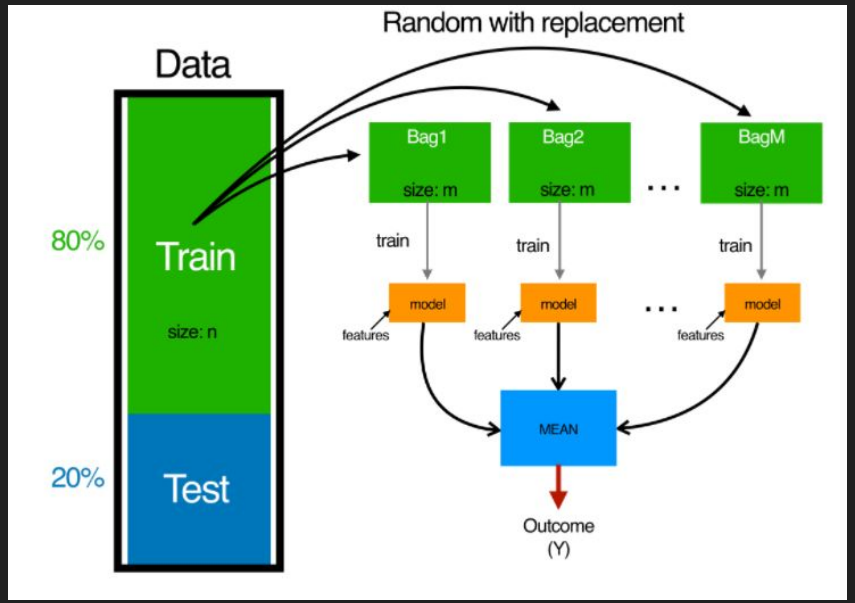
Umożliwia to stworzenie różnych zbiorów danych o podobnych rozkładach.



Bagging

1. Jest to stworzenie m “bootstrapowych” próbek
2. Dla każdej z próbek tworzymy drzewo regresyjne
3. Agregujemy wyniki ze wszystkich drzew, aby uzyskać ogólny wynik

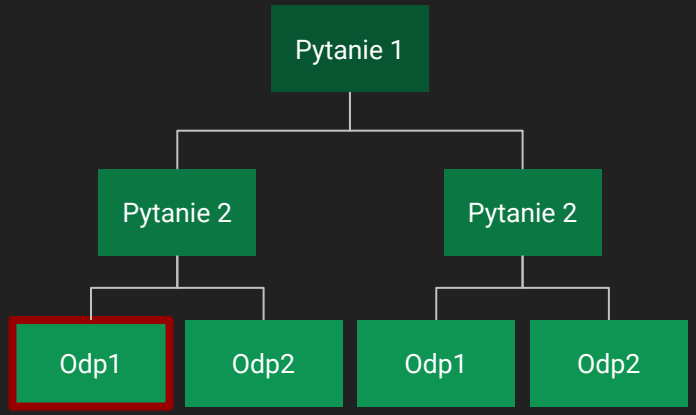
Metoda ta zmniejsza różnorodność i tendencję do nadmiernego dopasowywania się

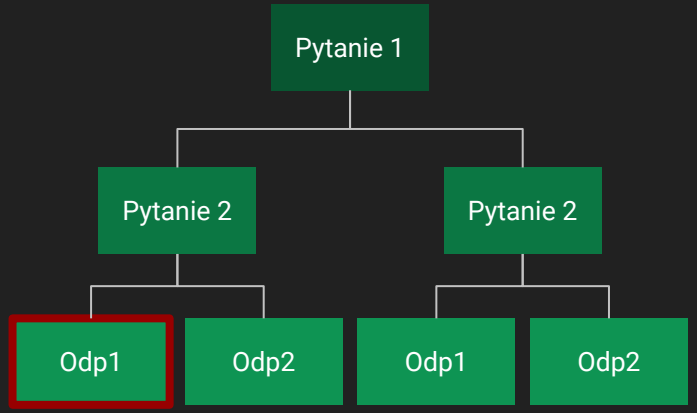


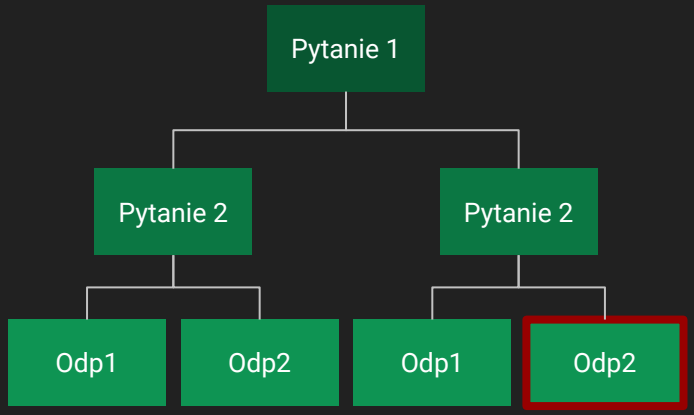
Historia algorytmu

- 1995- zaprezentowany przez Tim Kam Ho
- 2006- Leo Breim and Adele Cutler rozszerzyli go o połączenie “bagging”-u i losowego wyboru zmiennych.

Przykład z życia







Po co nam cały las?

- Pojedyncze proste drzewa decyzyjne nie są zbyt skuteczne.
- Drzewa z zastosowanym baggingiem nie są od siebie kompletnie niezależne, bo wszystkie zmienne objaśniające są rozważane w każdym rozgałęzieniu każdego drzewa. Z tego powodu drzewa z różnych “bootstrap” próbek mają podobne struktury. Widać to szczególnie w górnych warstwach.

Las losowy łączy prostotę drzew z polepszeniem dokładności nawet dla danych nie pochodzących ze zbioru.

Zbudujmy las losowy

Krok 1: Tworzenie “bootstrapped” zbioru danych

Zbiór danych

Piętrowy	Bliźniak	Basen	m^2	Drogi
Nie	Nie	Nie	125	nie
Tak	Tak	Tak	180	tak
Tak	Tak	Nie	210	nie
Tak	Tak	Tak	167	tak

“Bootstrapped” zbiór danych

Piętrowy	Bliźniak	Basen	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie

Krok 2:
Split-variable randomization

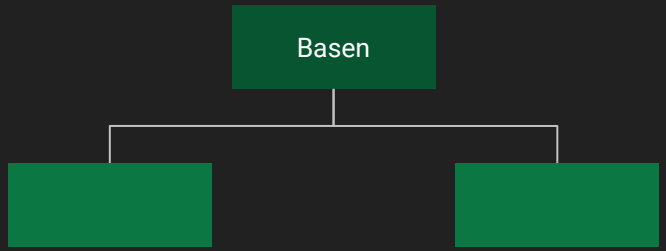
Stwórz drzewo decyzyjne na podstawie utworzonego zbioru danych, ale używając tylko losowego podzbioru zmiennych w każdym kroku.

W przykładzie będziemy rozważać 2 kolumny w każdym z kroków

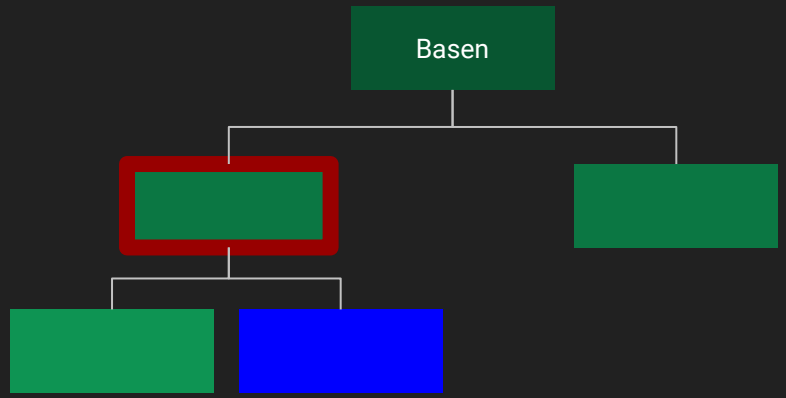
Wybieramy je losowo:

Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie

Załóżmy, że basen najlepiej oddziela próbki:



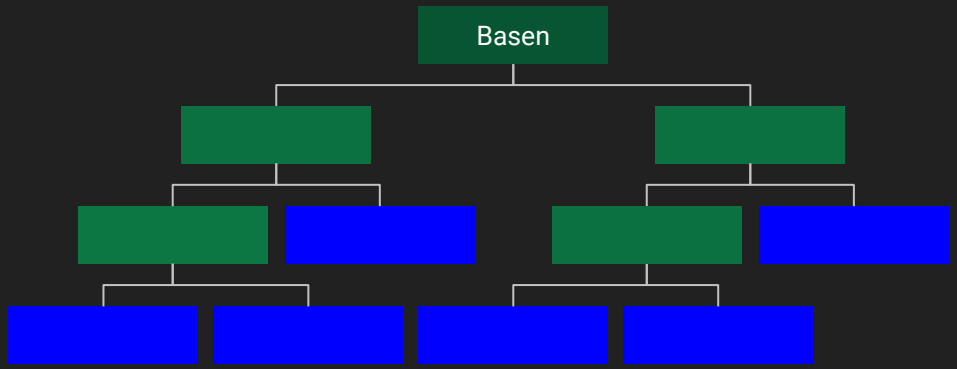
Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie



Wybieramy losowo kolumny do kolejnego rozgałęzienia

Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie

Dalej budujemy drzewo jak zwykle, rozważając tylko losowe podzbioru kolumn przy rozgałęzieniu



Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie

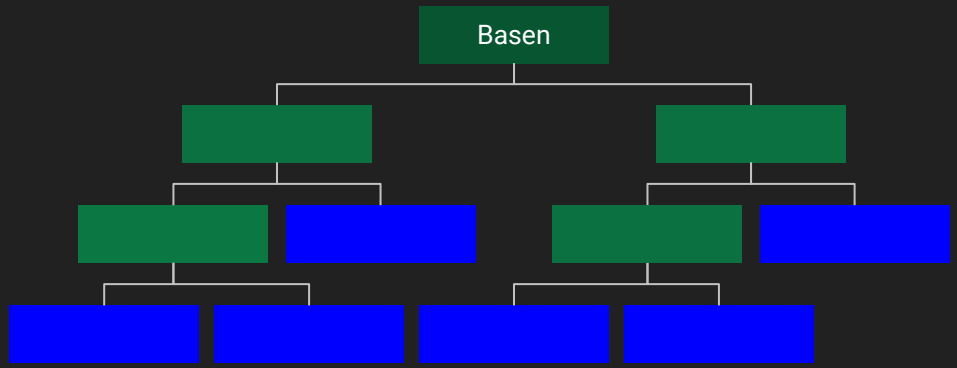
Teraz powtórzmy krok 1 i 2 n razy

Dzięki temu uzyskamy wiele różnorodnych drzew, co prowadzi do polepszenia efektywności względem pojedynczych drzew.
W rozważaniach przyjmujemy n=500

Jak tego używać

Piętrowy	Basen	Bliźniak	m^2	Drogi
Yes	No	No	160	???

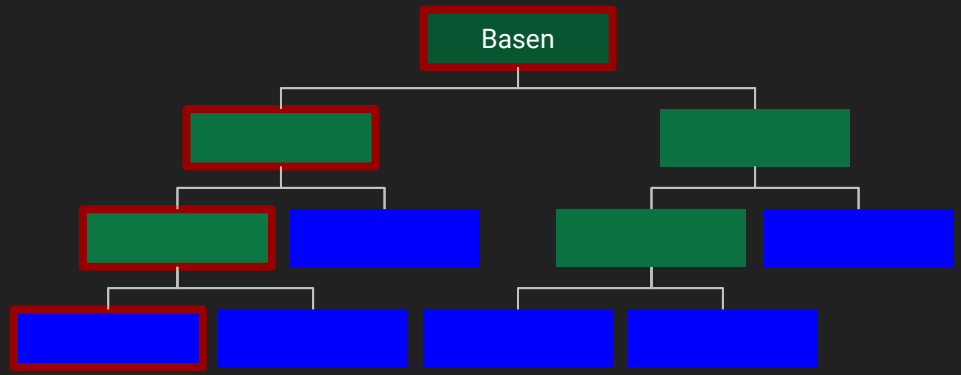
Z takimi danymi chcemy przewidzieć czy dom jest drogi…



Piętrowy	Basen	Bliźniak	m^2	Drogi
Yes	No	No	160	???

Z takimi danymi chcemy przewidzieć cenę…

Bierzemy dane i przechodzimy po pierwszym drzewie



Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Nie	160	???

Wynika, że dom jest drogi.

Zapamiętujemy wynik

Drogi?	
tak	nie
1	0

Teraz powtarzamy ten proces z każdym drzewem

Na koniec patrzymy, którą odpowiedź uzyskaliśmy najczęściej.
To właśnie ona jest tą finalną na podstawie wrzuconych danych

Jak sprawdzić czy to działa?

Zwykle ⅓ rekordów nie znajduje się w “bootstrapped” zbiorze.

Z naszego zbioru 4 rekord się w nim nie znalazł

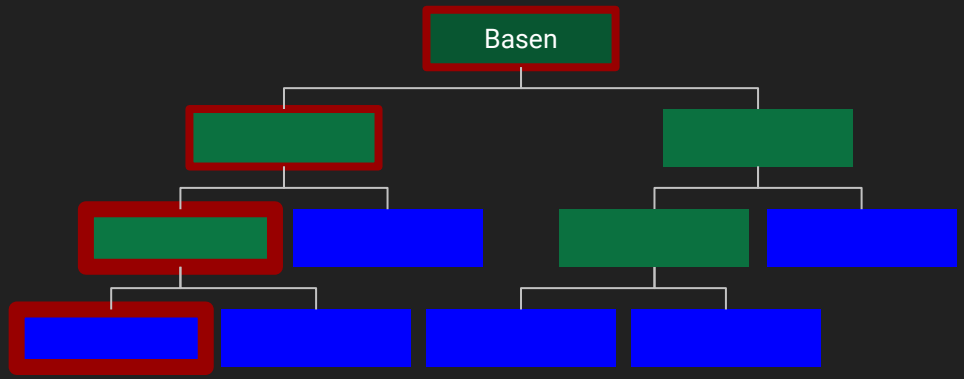
Zbiór danych

Piętrowy	Bliźniak	Basen	m^2	Drogi
Nie	Nie	Nie	125	nie
Tak	Tak	Tak	180	tak
Tak	Tak	Nie	210	nie
Tak	Tak	Tak	167	tak

“out-of-bag” zbiór danych

Piętrowy	Bliźniak	Basen	m^2	Drogi
Tak	Tak	Tak	167	tak

Możemy go wykorzystać, żeby sprawdzić czy nasz algorytm sprawdza się.



Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	167	tak

Dla pierwszego drzewa algorytm sprawdził się.

Powtarzamy dla wszystkich drzew niewykorzystujących tego rekordu:

Drogi?	
tak	nie
350	100

Uzyskana wartość to ta z największą liczbą wystąpień.
W tym przypadku algorytm dał prawidłową odpowiedź.

Powtarzamy dla wszystkich “out-of-bag” rekordów ze wszystkich “bootstrapped” zbiorów danych.

Miarą efektywności jest proporcja tych próbek które zostały poprawnie zaklasyfikowane

“Out-of-bag error” udział próbek, które zostały zaklasyfikowane błędnie

W przykładzie będziemy rozważać 2 kolumny w każdym z kroków

Wybieramy je losowo:

Piętrowy	Basen	Bliźniak	m^2	Drogi
Tak	Tak	Tak	180	tak
Nie	Nie	Nie	125	nie
Tak	Tak	Nie	210	nie
Tak	Tak	Nie	210	nie

Wiedząc o “out-of-bag error” możemy go porównywać dla różnych m (tu równego 2) i wybrać najlepszy model lasu losowego.

Zwykle m=p/3, gdzie p to ilość zmiennych objaśniających.
Jeśli m=p to algorytm sprowadza się do wykonywania kroku 1., czyli baggingu.

Czy coś się zmieni dla lasu losowego dla regresji?

Zbiór danych

Piętrowy	Bliźniak	Basen	m^2	Cena
Nie	Nie	Nie	125	300K
Tak	Tak	Tak	180	600K
Tak	Tak	Nie	210	320K
Tak	Tak	Tak	167	640K

Po prostu budujemy drzewa regresyjne zamiast klasyfikacyjnych i agregujemy uśredniając odpowiedzi.

Zalety i Wady

Zalety:

- Zazwyczaj dają bardzo dobre rezultaty
- Dają dość dobre rezultaty bez zmieniania domyślnych parametrów
- Wewnętrzny zbiór walidacyjny- nie trzeba poświęcać dodatkowych danych
- Nie wymagają zbyt dużego pre-processingu
- Są odporne na outliery

Wady:

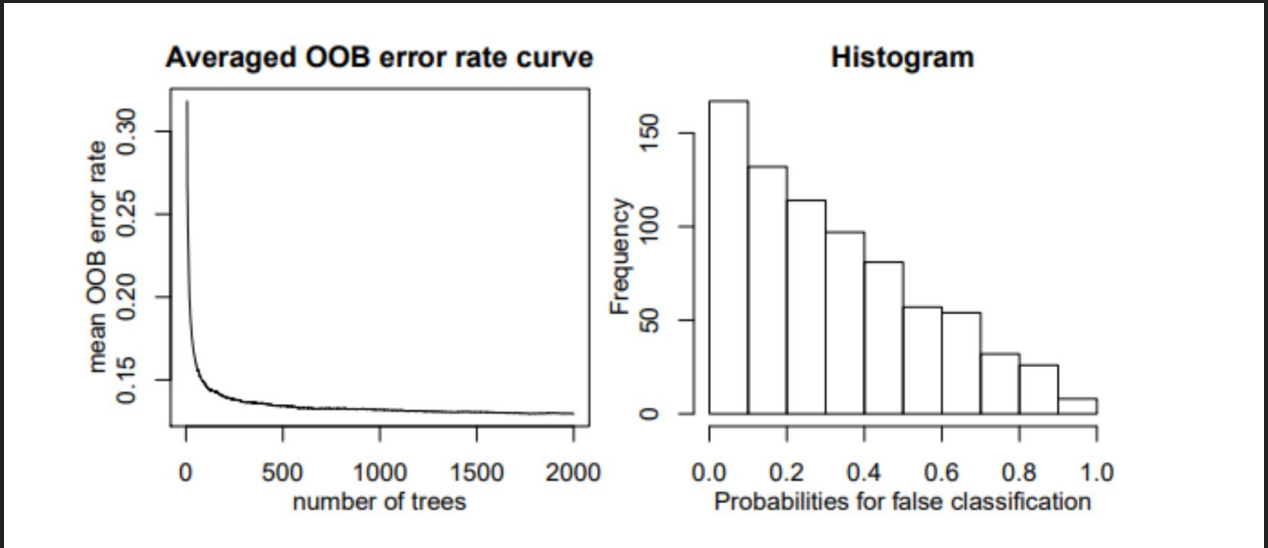
- Wolne dla dużych zbiorów danych
- Trudniej interpretowalne od drzew
- Często wypadają gorzej od zaawansowanych algorytmów “boostingowych”

Parametry i ich wpływ

- Liczba drzew decyzyjnych w lesie
- Maksymalna wysokość każdego z drzew
- Minimalne próbki do podziału w wewnętrznym węźle
- Maksymalna liczba “liści”
- Liczba losowych obiektów
- Rozmiar początkowego zestawu danych
- Kryteria podziału w węzłach

Liczba drzew decyzyjnych

- Zwykle im wyższa liczba drzew, tym lepiej działa model
- Duża liczba drzew może spowolnić obliczenia
- Konieczne jest znalezienie optymalnej wartości



Maksymalna wysokość drzew

- Zwiększenie głębokości poszczególnych drzew zwiększa możliwą liczbę kombinacji cech, które są brane pod uwagę.
- Jeśli wartość jest wyższa niż wartość optymalna, model będzie miał tendencję do nadmiernego dopasowania (overfitting).
- Mała głębokość może utrudnić proces treningu.

Kryteria podziału

$$Gini = 1 - \sum_{j=1}^{c} p_j^2$$

$$Entropy = -\sum_{j=1}^{c} p_j \log p_j$$

Drzewa podejmują optymalne lokalnie decyzje w każdym węźle, obliczając, która cecha i która wartość tej cechy najlepiej rozdziela obserwacje do tego punktu.

- Regresja: ogólną zasadą jest przyjmowanie MSE - jeśli w danych nie ma wielu wartości odstających, w przeciwnym wypadku - MAE
- Klasyfikacja: Musimy obliczyć miarę “zanieczyszczenia” za pomocą Gini lub Entropii, co czasami może skutkować innym podziałem.

Liczba losowych cech przy każdym podziale

Najlepsza wartość tego parametru jest trudna do wybrania bez eksperymentowania.

Weź pod uwagę:

- Mała wartość (mniej cech branych pod uwagę w każdym węźle) zmniejszy wariancję, kosztem wyższego błędu indywidualnego drzewa.
- Zwiększenie maksymalnej liczby cech losowych uwzględnionych w podziale zwykle zmniejsza obciążenie modelu, ponieważ istnieje większa szansa na uwzględnienie dobrych cech, jednak z ryzykiem zwiększonej wariancji oraz czasu wykonywania

Rozmiar zestawu danych Bootstrapped

Procent danych treningowych wykorzystany do trenowania każdego drzewa z osobna.

Ponieważ obserwacje są próbkowane z zastąpieniem, nawet jeśli rozmiar początkowego zestawu danych jest taki sam jak całego zestawu uczącego, oba zestawy danych będą różne, więc wiele razy ten parametr pozostaje nienaruszony, a każde drzewo jest trenowane za pomocą losowego zestawu obserwacji z tym samym rozmiarem początkowych danych treningowych.

Nie zaszkodzi użyć pełnego rozmiaru danych treningowych, najlepiej pozostawić wartość domyślną.

Pakiety w R

Lista pakietów z implementacją RF

Nazwa pakietu	Miesięczne pobrania z CRAN
ranger	~85 000
randomForest	~74 000
xgboost	~76 000
h20	~25 000
randomForestSRC	~8 000
Rborist	~1 000

ranger

- Implementacja lasu losowego na podstawie Breiman (2001), wyspecyfikowana pod wielowymiarowe dane.
- Poza regresją i klasyfikacją wspierane są także survival forests (opisane w Ishwaran et al. 2008), extremely randomized trees (Geurts et al. 2006) oraz quantile regression forests (Meinhausen 2006)
- Pakiet napisany jest w C++, dzięki czemu proces treningu jest bardzo szybki

```
## Classification forest with default settings
ranger(Species ~ ., data = iris)

## Prediction
train.idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
rg.iris <- ranger(Species ~ ., data = iris.train)
pred.iris <- predict(rg.iris, data = iris.test)
table(iris.test$Species, pred.iris$predictions)

## Quantile regression forest
rf <- ranger(mpg ~ ., mtcars[1:26, ], quantreg = TRUE)
pred <- predict(rf, mtcars[27:32, ], type = "quantiles")
pred$predictions
```

Random Forest

- Do niedawna najpopularniejszy pakiet implementujący las losowy
- Dostosowany zarówno dla klasyfikacji jak i regresji
- Stworzony na podstawie artykułu Breiman (2001)
- Poza tworzeniem lasu losowego pozwala także na rysowanie lasu, ekstrakcje ważności zmiennych, ekstrakcje pojedynczych drzew oraz ich ensembling

```
# S3 method for formula
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
# S3 method for default
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             weights=NULL,
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
```

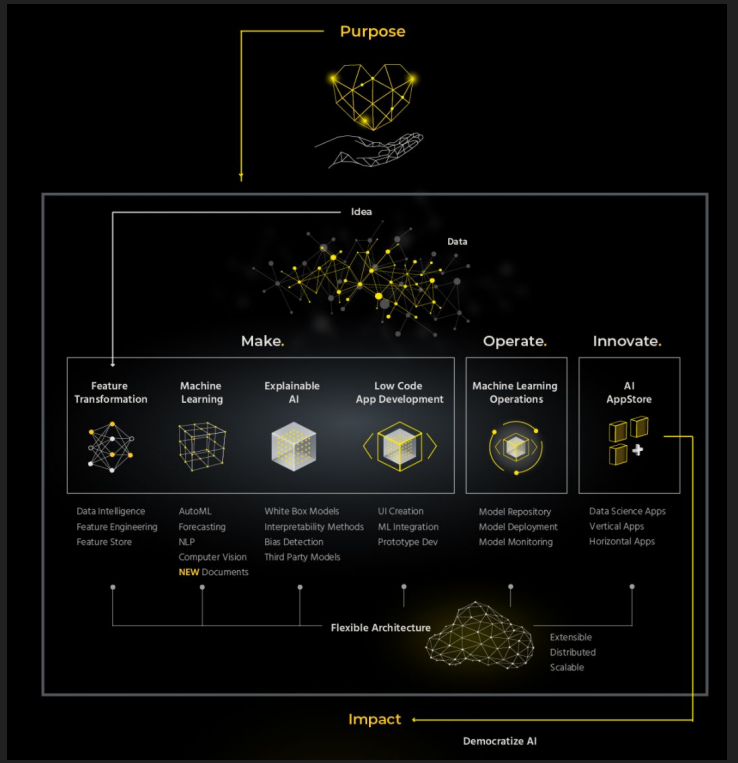
xgboost

- Pakiet implementujący Extreme Gradient Boosting na podstawie Chen & Guestrin (2016)
- Tradycyjny las losowy można uzyskać poprzez odpowiednie ustawianie hiperparametrów
- Za pomocą pakietu można narysować otrzymane drzewa, zapisać i wczytać model, otrzymać ważność zmiennych, dokonać crossvalidacji, czy też znormalizować dane

```
## A simple xgb.train example:
param <- list(max_depth = 2, eta = 1, verbose = 0, nthread = 2,
              objective = "binary:logistic", eval_metric = "auc")
bst <- xgb.train(param, dtrain, nrounds = 2, watchlist)
```

h2o

- Pakiet H2O to otwarta, rozproszona, szybka i skalowalna platforma uczenia maszynowego i analiz predykcyjnych.
- W paczce znaleźć można implementacje dotyczących Deep Learningu, Gradient Boostingu, XGBoosta, Random Forest i wiele więcej
- Pakiet pod spodem napisany został w Javie
- H2O wspiera także obliczenia na serwerach zewnętrznych, pozwala na pracę z Hadoopem oraz zajmuje się AutoML'em



randomForestSRC

- Pełna nazwa: Fast Unified Random Forest for Survival, Regression and Classification (RF-SRC)
- Implementacja nie tylko tradycyjnego lasu losowego lecz także lasów służących do analizy ryzyka, uczących się nie nadzorowanie, regresji kwantylowej i wiele więcej.
- Posiada wiele zbiorów danych, np. wines, hd, wlhs

```
## veteran data
## randomized trial of two treatment regimens for lung cancer
data(veteran, package = "randomForestSRC")
v.obj <- rfsrc(Surv(time, status) ~ ., data = veteran,
                    ntree = 100, block.size = 1)
```

Rborist

- Implementacja klasycznego lasu losowego zoptymalizowanego pod wielowątkowość i obliczenia na GPU
- Napisany w C++
- Planowana implementacja pakietu pod Pythona

ParallelForest

- Zrównoleglona implementacja klasycznego lasu losowego
- Napisana w Fortranie

```
fforest = grow.forest(
  Y~.,
  data=low_high_earners,
  min_node_obs = 1000,       # min obs to split node
  max_depth = 15,            # max tree depth
  numvars = 3,               # num vars to draw for each tree
  numboots = 50              # number of trees in the forest
)
```

Rdocumentation

Ranger: https://www.rdocumentation.org/packages/ranger/versions/0.13.1/topics/ranger

randomForest: https://www.rdocumentation.org/packages/randomForest/versions/4.7-1

xgboost: https://www.rdocumentation.org/packages/xgboost/versions/1.5.2.1

h20 : https://www.rdocumentation.org/packages/h2o/versions/3.36.0.3

randomForestSRC: https://www.rdocumentation.org/packages/randomForestSRC/versions/3.0.2

Rborist: https://www.rdocumentation.org/packages/Rborist/versions/0.2-3

Preprocessing

Preprocessing pipeline

- Transformacja ramki do typu Data Frame
- Imputacja brakujących wartości (NA):
  - brak targetu oznacza usunięcie obserwacji
  - brakujące wartości kategoryczne uzupełniamy modą
  - brakujące wartości numeryczne medianą
- Skalowanie wartości kolumn ciągłych
- Binaryzacja kolumn kategorycznych, które mają 2 kategorie
- One-hot-encoding kolumn o wielu kategoriach
- Podział na zbiór treningowy i walidacyjny

References

Articles

Breiman (2001) https://link.springer.com/article/10.1023/A:1010933404324

Chen & Guestrin (2016) https://dl.acm.org/doi/10.1145/2939672.2939785

Ishwaran et al. 2008 https://arxiv.org/pdf/0811.1645.pdf

Geurts et al. 2006 https://link.springer.com/article/10.1007/s10994-006-6226-1

Meinhausen 2006
https://www.jmlr.org/papers/volume7/meinshausen06a/meinshausen06a.pdf

Blogs

https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

https://towardsdatascience.com/random-forest-hyperparameters-and-how-to-fine-tune-them-17aee785ee0d