infoShare ACADEMY

xgBoost

Przypomnienie

Wariancja (variance) - jak duże błędy otrzymujemy na zestawie testowym lub walidacyjnym

Obciążenie (bias) - błąd między wyjściem naszego estymatora (wartością oczekiwaną) a rzeczywistą wartością w zbiorze testowym

Niskie obciążenie a duża wariancja to **przeuczenie (overfitting)**.

Wysokie obciążenie wskazuje na niedouczenie / zbyt niską złożoność modelu.

Przypomnienie

Drzewo decyzyjne - graf uczony maksymalizacją zysku informacyjnego, decyzje podejmowane w węzłach bazują na jednej zmiennej na raz, łatwo się przeucza (duża wariancja), chyba że zostanie zregularyzowane.

Ensemble method - metoda uczenia mocnego modelu przy pomocy łączenia wyjść wielu słabszych modeli (tzw. weak learner). Ma to na celu głównie obniżenie wariancji modelu kosztem jego złożoności.

Las losowy - *ensemble method* zawierający w sobie wiele drzew decyzyjnych, uczony przez **bagging - *bootstrap aggregating*** - co ma na celu zmniejszenie wariancji modelu

Metody ensemble

Głosowanie (voting)

Uśredniamy predykcje wielu modeli. Zazwyczaj są to różne modele, ale mogą być modele o różnej konstrukcji.

Bootstrap aggregating

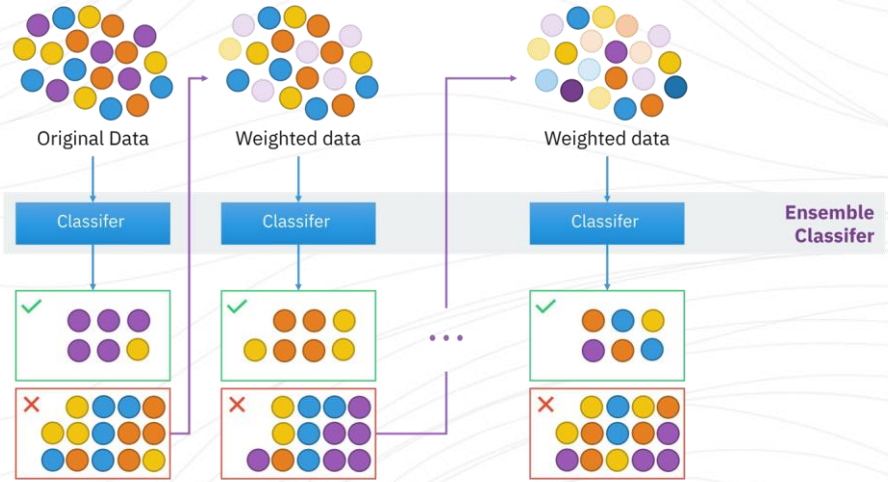
Próbkujemy zestaw danych, aby otrzymać nowy zestaw danych o innej dystrybucji punktów, czasami pomijając niektóre próbki lub cechy. Możemy uczyć pojedyncze modele w ten sposób

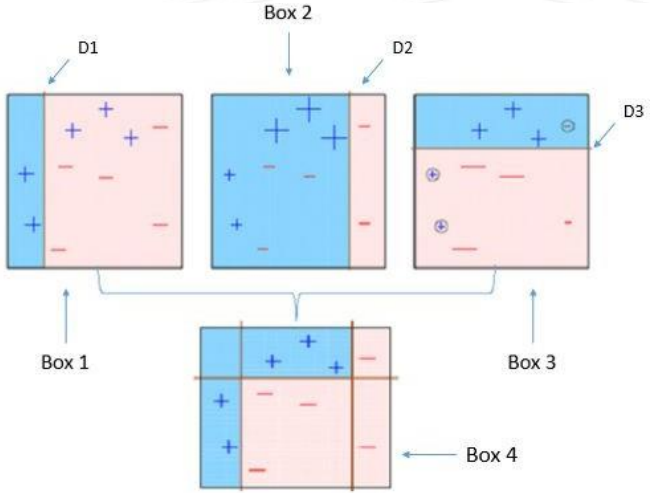
Boosting

Metody oparte o poprawianie błędów modeli poprzednio uczonych na tych samych danych. Każdy kolejny model stara się nauczyć tego, czego nie nauczył się poprzedni.

Więcej na: https://scikit-learn.org/stable/modules/ensemble.html

Boosting - intuicja





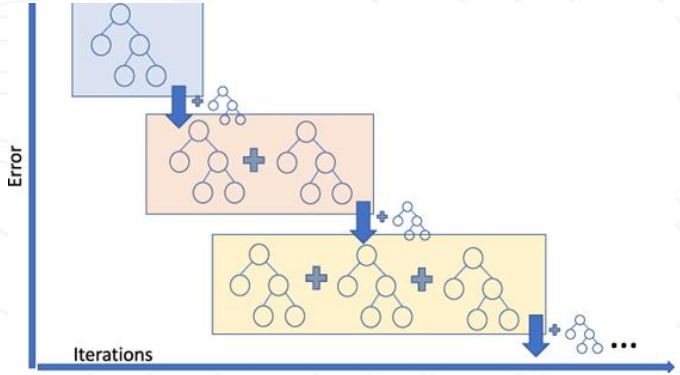
Czym jest xgBoost?

- niezależna biblioteka napisana w C++
- zaproponowana w 2016 roku przez Tiangi Chen
- wieloplatformowa
- kompatybilna z API scikit-learn
- oparta o kilka rodzajów słabych estymatorów (np. drzew decyzyjnych)
- metoda ensemble
- state-of-the-art dla danych tabelarycznych
- szybka i efektywna
- zawiera dodatkowy czynnik regularyzacyjny (poza minimalizacją błędu nakłada karę za złożoność modelu)

Gradient boosting

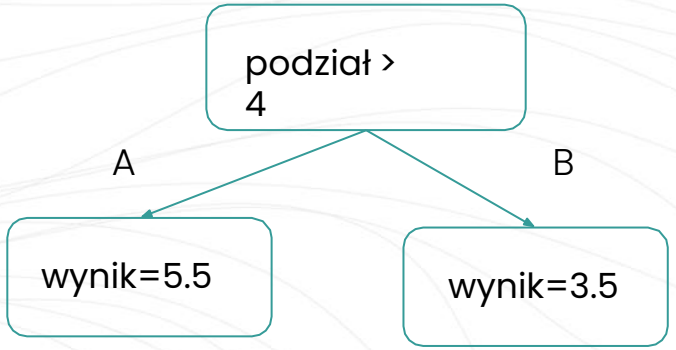
Metoda *boostingu* jest metodą sekwencyjną, opartą o zespół klasyfikatorów. Każdy kolejny słaby estymator jest uczony tak, aby uwzględniać wartość błędu poprzedniego modelu i dołączać taki estymator do zbioru modeli ze stałą wagą, co pozwala stopniowo dążyć do minimum błędu w sposób podobny do zejścia gradientu.

Próbujemy dopasować kolejny predyktor do błędu resztkowego popełnionego przez poprzedni predyktor.



Jak obliczyć wartość rezydualną?

Podział przez drzewo	Wartość prawdziwa	Wartość rezydualna
A	5	5 - 5.5 = -0.5
B	3	3 - 3.5 = -0.5
A	6	6 - 5.5 = 0.5
B	4	4 - 3.5 = 0.5



Wartość rezydualna to różnica między wartością rzeczywistą, a wartością estymowaną przez poprzedni model.

Jak wygląda model xgBoost?

Każdy kolejny model stara się przewidzieć błąd wszystkich modeli przed nim. Tzn. model 2 przewiduje błąd modelu 1 + średniej, model 1 modeluje błąd średniej.

Parametr C jest rozmiarem kroku, jaki robimy w naszej optymalizacji. Jest to tzw. *learning rate*.

Objective function: training loss + regularization

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

Training loss

MEAN SQUARED ERROR

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

LOGISTIC LOSS

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

Funkcja straty (celu)

XGboost pozwala dobrać dowolną funkcję straty poza tymi, które są podane. Parametr *objective* zawiera bardzo dużą bibliotekę już zaimplementowanych funkcji, nie tylko do klasyfikacji i regresji, ale np. do analizy przeżywalności (*time to event modelling*).

Funkcja straty musi być różniczkowalna. Dokładne wyprowadzenie (opierające się o przybliżenie funkcji błędu szeregiem Taylora) można znaleźć w oryginalnym artykule, albo na jednym z artykułów popularno-naukowych w internecie, np:

https://towardsdatascience.com/xgboost-mathematics-explained-58262530904a

Z czego się składa xgBoost?

$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

Suma błędu odpowiedzi wszystkich drzew

Suma regularyzacji wszystkich drzew

Założenie: funkcja błędu ogólnego modelu zależy od sumy funkcji błędów dla każdego kolejnego modelu estymującego błąd.

Proces uczenia - additive training

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

xgBoost w Pythonie

```
import xgboost as xgb
```

Dane wejściowe w Pythonie

Pandas DataFrame

NumPy array

xgBoost DMatrix

infoShare ACADEMY

Funkcje celu

- Wiele modeli w ramach jednej funkcji celu
- Początkowo lepsze niż randomowe (słabi uczniowie)
- Razem osiągają znacznie lepszy wynik niż świetnie wytrenowany pojedynczy model

objective: określa użytą funkcję straty (loss)

Learners

- Domyślnie ustawiony jest booster:gbtree, czy gradient boosted tree
- Modyfikacją pierwszego jest booster:dart
- Do dyspozycji jest jeszcze booster:gblinear
  - Nie jest zbyt popularny
  - Do treningu używamy funkcji .train() zamiast znanej .fit()

Regularizations

Kara za złożoność modelu.

Istotne hiperparametry modelu:

- alpha - regularyzacja L1 (lasso)
- lambda - regularyzacja L2 (ridge)

Użycie tych parametrów ma na celu zmniejszenie złożoności modelu, aby zmniejszyć wariancję kosztem obciążenia. Używa się tych parametrów, aby unikać overfittingu. L1 i L2 regularyzują wagi poszczególnych przykładów w kalkulacji funkcji celu.

Gamma - kontroluje, czy nastąpi podział węzła na podstawie oczekiwanej redukcji funkcji straty po podziale, im wyższe wartości tym mniej podziałów.

Parametry

- **booster** (gbtree/ gblinear/ dart) - rodzaj użytych słabych klasyfikatorów
- **learning_rate** (eta) - spadek learning_rate w trakcie treningu, szybkość uczenia (domyślnie 0.3)
- **gamma** - minimalna zmiana funkcji straty potrzebna do wykonania kolejnego podziału w drzewie (analogiczne do parametru `min_impurity_decrease` w drzewach) (default=0)
- **max_depth** - jak w drzewach, maksymalna głębokość drzewa
- **n_estimators** - liczba drzew wzmocnionych gradientem
- **subsample** - podpróbka zestawu szkoleniowego - domyślnie 0.5 zestawu danych
- **objective** - określa użytą funkcję straty
- **colsample_bytree** - podpróbka kolumn użytych do konstrukcji każdego drzewa
- **n_jobs** - zrównoleglenie wątków
- **verbosity** - wartości: 0 (silent), 1 (warning), 2 (info), and 3 (debug)
- **eval_metric** - pozwala zmienić domyślnie używaną miarę oceny modelu (domyślnie dla regresji RMSE a dla klasyfikacji accuracy)
- **missing** - wartość, która ma być traktowana jako brak danych

https://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters
https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn
https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html

Visualization

- Rysunek drzewa
  - xgb.plot_tree()
  - 1 parametrem jest wytrenowany model
  - 2 parametrem jest głębokość rysunku

- Wykres istotności cech
  - xgb.plot_importance()
  - Parametrem jest model

Zalety

- Szybki i wydajny
- Algorytm można zrównoleglić
- Bardzo dokładny dla danych o ilości próbek większej niż liczba cech
- Odpowiedni dla danych numerycznych oraz kategorycznych
- Używany w problemach regresji i klasyfikacji
- Kompatybilny z API scikit-learn
- Szeroki wybór wbudowanych parametrów strojenia modelu

Wady

Nie jest odpowiedni do problemów:

- Lepiej rozwiązywalnych przez algorytmy uczenia głębokiego:
  - rozpoznawania obrazów
  - przetwarzania języka naturalnego
- Z małą liczbą przypadków uczących
- Danych, gdzie ilość featurów jest porównywalna lub większa od ilości próbek

Przydatne linki

dokumentacja xgboost

wprowadzenie do matematyki i procesu estymacji w xgboost

implementacje oparte o gradient boosting:

- XGBoost – algorytm napisany przez Tianqi Chen. Chyba najbardziej znana i najczęściej używana implementacja: https://arxiv.org/pdf/1603.02754.pdf
- LightGBM – algorytm Microsoftu: https://lightgbm.readthedocs.io

Lasy losowe vs gradient boosting

xgboost in Python

Dzięki!