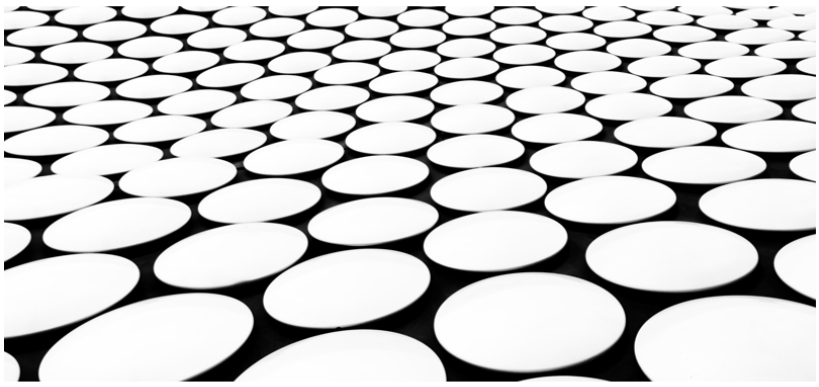


SCHOOL OF AI
ITALIA

MACHINE LEARNING AVANZATO DA ZERO

ANTONIO DI CECCO - SCHOOL OF AI

(Stochastic) Gradient Descent, Gradient Boosting

Resume: Gradient Descent

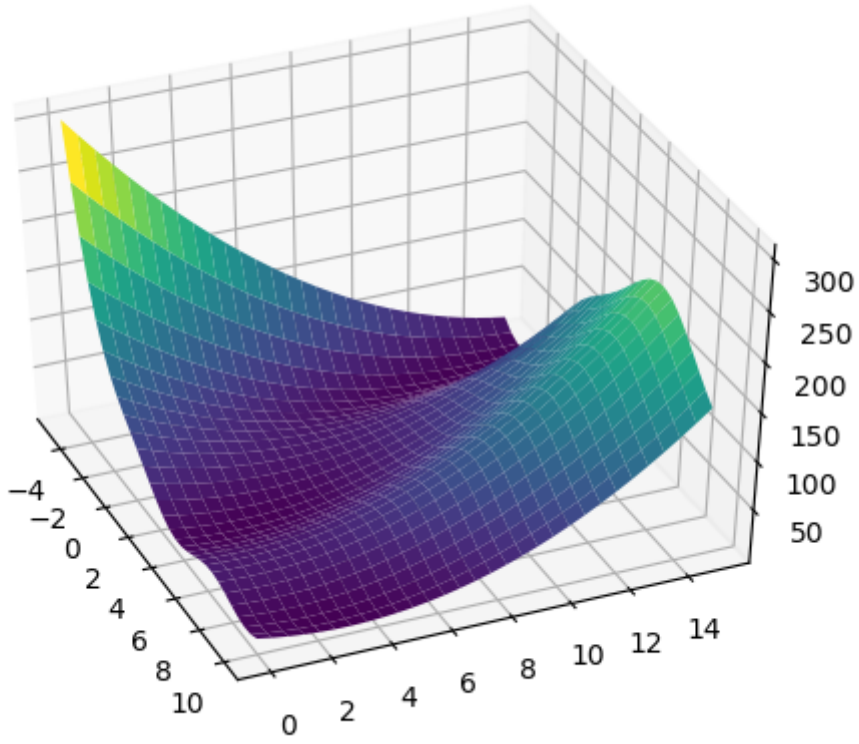
- Vogliamo

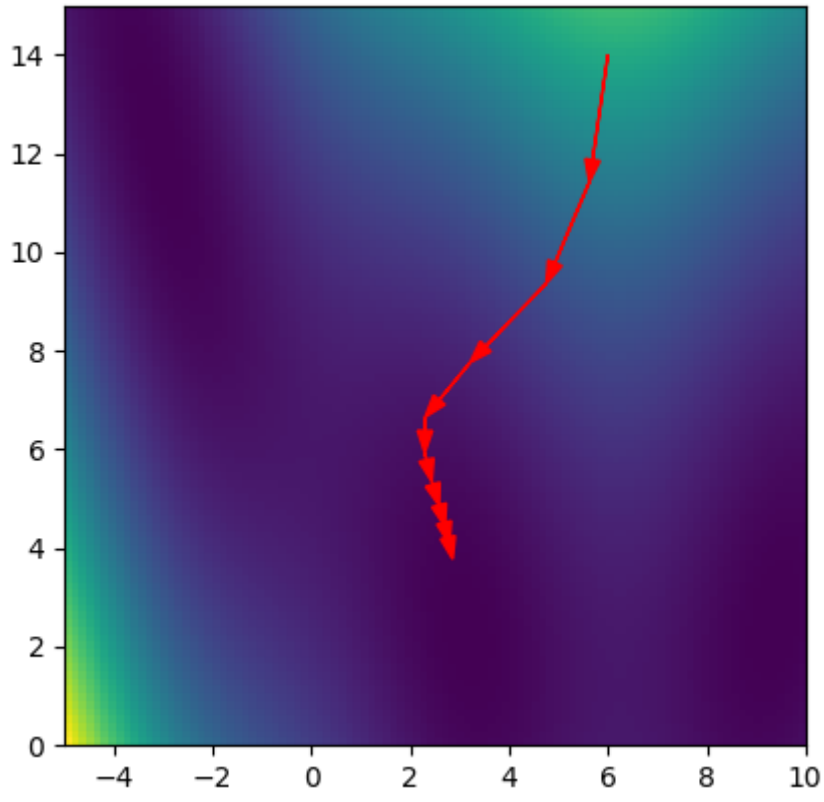
$$\arg\min_w F(w)$$

- Inizializiamo w_0

$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

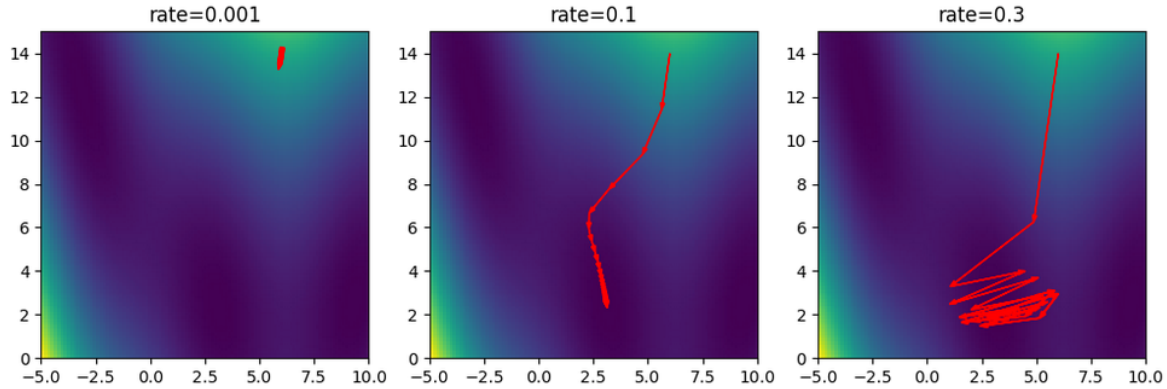
- η_i è un coefficiente fissato (spesso implementato con *decay* nel tempo)
- converge a un **minimo locale**
- se la funzione è convessa converge al **minimo globale**
- in prossimità del minimo il gradiente è piccolo in modulo quindi gli step diventano sempre più piccoli (una buona idea)





$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

“honey pot” learning rate



$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

https://www.benfrederickson.com/numerical-optimization/

(Stochastic) Gradient Descent

Logistic Regression Objective:

$$F(w, b) = -C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i i - b) + 1) + ||w||_2^2$$

Gradient:

$$\frac{d}{dw} F(w) = \frac{d}{dw} - C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i - b) + 1) + ||w||_2^2$$

Stochastic Gradient: Pick x_i randomly, then

$$\frac{d}{dw} F(w) \approx \frac{d}{dw} - C \log(\exp(-y_i w^T \mathbf{x}_i i - b) + 1) + \frac{1}{n} ||w||_2^2$$

In practice: just iterate over i.

SGDClassifier, SGDRegressor e partial_fit

```python
# Esegui fino alla convergenza
sgd = SGDClassifier().fit(X_train, y_train)

# Esegui una iterazione su un dataset diviso in (mini)batch
sgd = SGDClassifier()
for X_batch, y_batch in batches:
    sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])

# Esegui molte iterazioni su un dataset diviso in batch
for i in range(10):
    for X_batch, y_batch in batches:
        sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])
```

SGD e partial_fit

- SGDClassifier(), SGDRegressor() molto veloci su grandi dataset
- Effettuare il tuning di learning rate e batch può essere complicato
- partial_fit ci permette di lavorare con dati out-of-memory !
- **Online learning**

Boosting

$$f(x) = \sum_k g_k(x)$$

Famiglia di algoritmi che creano uno "strong" learner f da "weak" learner g_k

AdaBoost, GentleBoost, LogitBoost, ...

Ci sono moltissimi tutorial su AdaBoost (implementato in SKLearn) look it up

Idea di AdaBoost

- fitto un modello vedo l'errore che commette nel prevedere sample per sample
- i sample dove l'errore è maggiore saranno pesati di più nell'addestramento di modelli successivi
- per finire faccio una media dei modelli (*bagging like*)
- si è scoperto che questa procedura può essere generalizzata nel Gradient Boosting

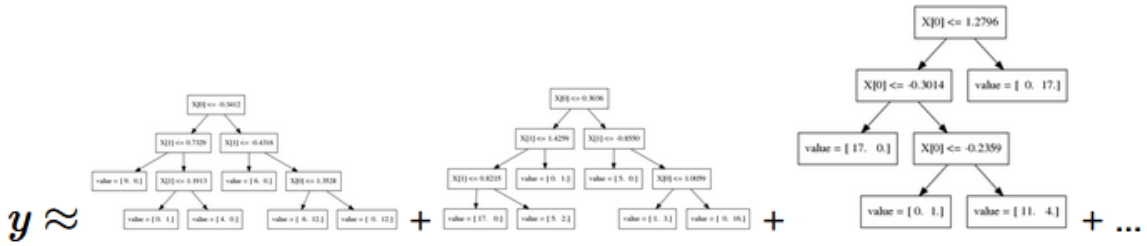
Gradient Boosting

Algoritmo di Gradient Boosting

$$f_1(x) \approx y$$

$$f_2(x) \approx y - f_1(x)$$

$$f_3(x) \approx y - f_1(x) - f_2(x)$$

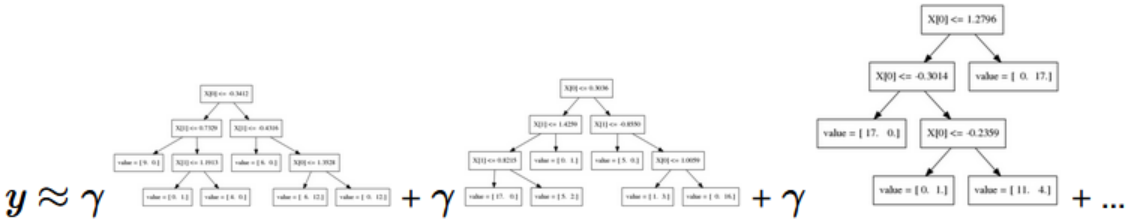


- i fit successivi sono effettuati sul residuo di quelli precedenti
- BEWARE pericolo di overfit
- procedura più gentile

$$f_1(x) \approx y$$

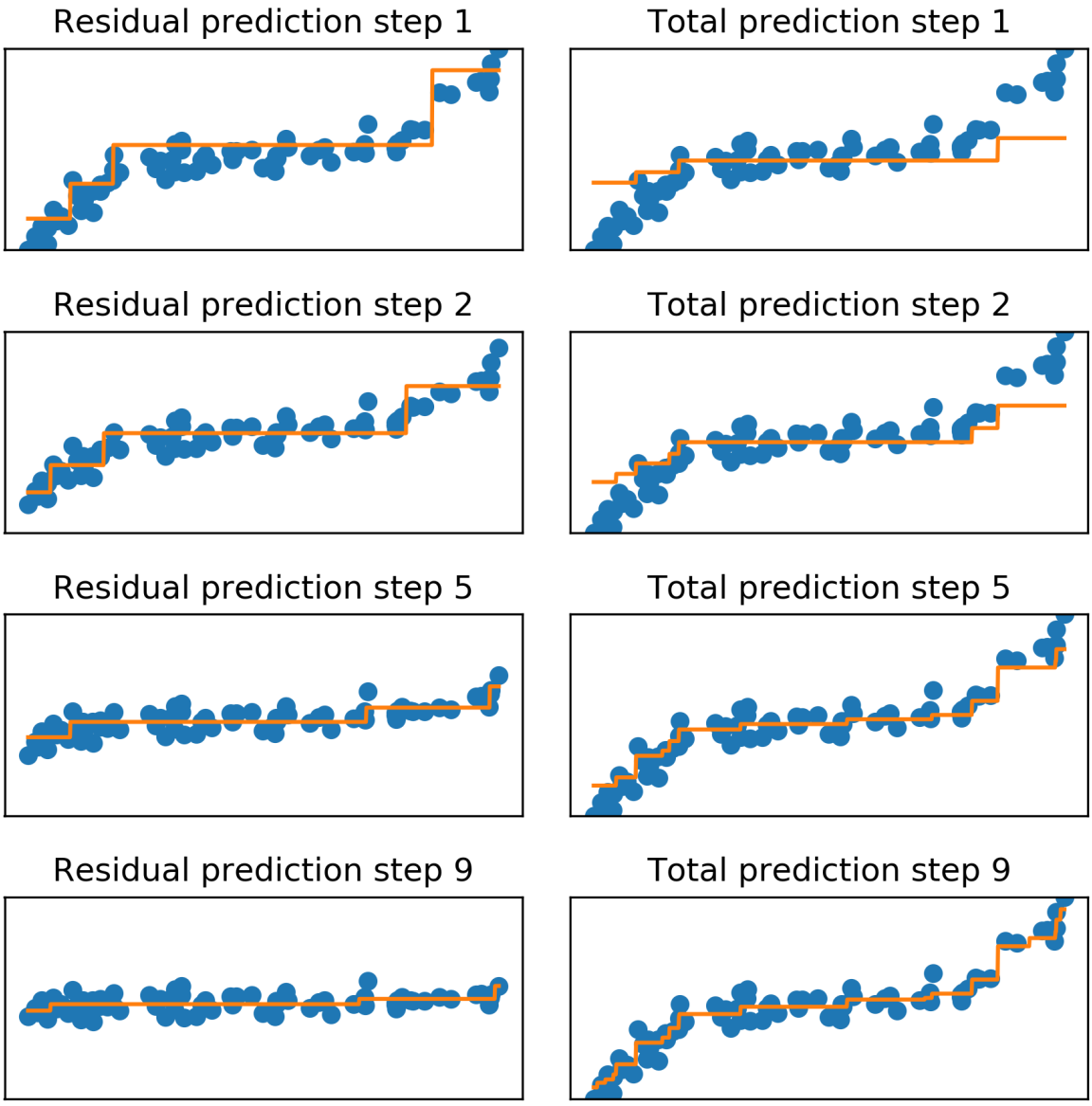
$$f_2(x) \approx y - \gamma f_1(x)$$

$$f_3(x) \approx y - \gamma f_1(x) - \gamma f_2(x)$$

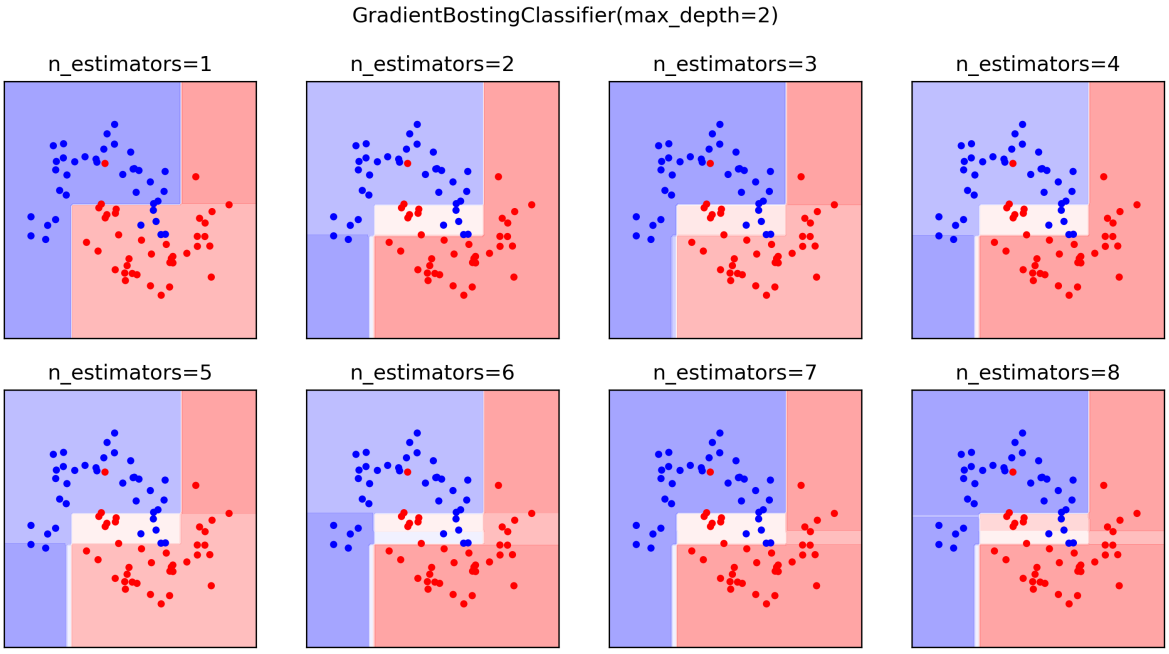


Learning rate $\gamma, i.e. 0.1$

GradientBoostingRegressor



GradientBoostingClassifier / HistGradientBoostingClassifier



Gradient Boosting vs Gradient Descent

- Linear regression vs gradient boosting

Linear regression

$$L(\mathbf{x}_i, y_i, \mathbf{w}, b) = \sum_i (y_i - \hat{y}_i)^2$$

$$= \sum_i (y_i - w^T \mathbf{x}_i - b)^2$$

optimize:

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} (y_i - w^T \mathbf{x}_i - b)^2$$

gradient descent:

$$w_{j+1} = w_j - \gamma \frac{\partial L(\mathbf{x}_i, y_i, \mathbf{w}, b)}{\partial \mathbf{w}}$$

Gradient Boosting

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

optimize:

$$\min_{\hat{y} \in \mathbb{R}^n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

gradient descent:

$$\hat{y}_{j+1} = \hat{y}_j - \gamma \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}}$$

Sorpresa: il Gradient Boosting è il gradient descent non rispetto ai parametri ma rispetto ai sample

---

Paragoniamo Logistic Regression e gradient boosting

Logistic Regression

$$\min_{w\in\mathbb{R}^p, b\in\mathbb{R}} \sum_{i=1}^{n} \log(\exp(-y_i(w^T\mathbf{x}_i + b)) + 1)$$

Gradient boosting

$$\min_{y_i\in\mathbb{R}^n} \sum_{i=1}^{n} \log(\exp(-y_i\hat{y}_i) + 1)$$

Multinomial logistic regression

$$p(y = c|x) = \frac{e^{\mathbf{w}_c^T\mathbf{x}+b_i}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^T\mathbf{x}+b_j}}$$

Multinomial Gradient Boosting

$$p(y = c|x) = \frac{e^{\hat{y}^{(c)}}}{\sum_{j=1}^{k} e^{\hat{y}^{(j)}}}$$

Un regression tree per class (per step del gradiente).

Cosa ottimizzare nelle GBM?

Early stopping (pocket algorithm)

- Aggiungere alberi può portare all'overfitting
- Smettete di aggiungere alberi quando l'accuratezza di validazione smette di crescere

due scelte:

- fissa il numero di alberi e tuni il learning rate
- fissa il learning rate, usa l'early stopping

L'early stopping è una forma di regolarizzazione perché non ti fa allontanare troppo dai valori iniziali dei parametri

---

Tuning of Gradient Boosting

- max_depth
- max_features
- column subsampling, row subsampling
- Regularizzazione esplicita l_1 Lasso o l_2 Ridge
- Scegli il learning rate e fa eearly stopping
- Scegli n_estimators (il numero di alberi), e tuna il learning rate (ninte early stopping)

  E' un modello basato sugli alberi così la predizione finale sarà una combinazione lineare di alberi e puoi guardare alla feature importance nello stesso modo fatto con DT e random forest

---

"Extreme" gradient boosting

Un miglioramento ulteriore si è avuto con

XGBoost: A Scalable Tree Boosting System, 2016

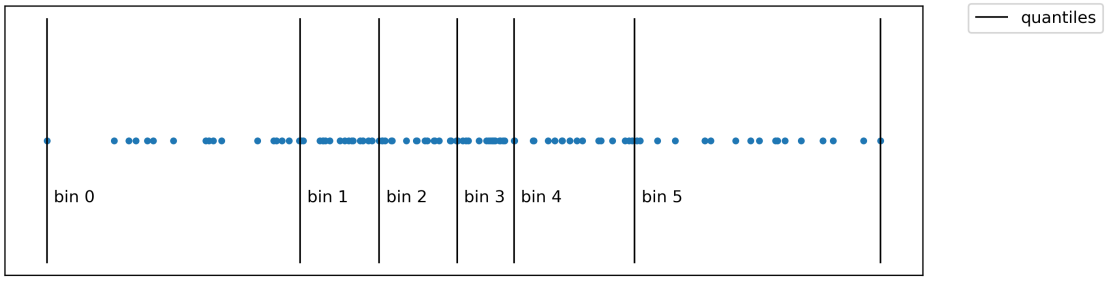
e successivi

Speeding up tree-building via binning

- trovare gli split è un'operazione lenta

```
for feature in features:
    for threshold in thresholds(f):
        gain = compute_gain(feature, threshold)
        if gain > best_gain:
            best_split = (feature, threshold)
```

- Thresholds: ricerca su tutti i valori unici della feature.
- fare la ricerca sui thresholds = ordinamento: O(n log n)
- soluzione il binning (l'istogramma)



Original

```
[[5. , 2. , 3.5, 1. ],
 [4.9, 3. , 1.4, 0.2],
 [4.4, 2.9, 1.4, 0.2],
 [5. , 2.3, 3.3, 1. ],
 [4.9, 2.5, 4.5, 1.7],
 [6.3, 2.5, 5. , 1.9],
 [6.3, 2.3, 4.4, 1.3],
 [5. , 3.5, 1.3, 0.3],
 [6.1, 2.8, 4.7, 1.2],
 [5. , 3.5, 1.6, 0.6]])
```

Binned

```
[[1, 0, 1, 1],
 [0, 2, 0, 1],
 [0, 1, 0, 1],
 [1, 0, 1, 1],
 [0, 0, 2, 3],
 [3, 0, 3, 4],
 [3, 0, 2, 2],
 [1, 4, 0, 1],
 [3, 1, 3, 2],
 [1, 4, 1, 1]])
```

(e il tempo diventa logaritmico)

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

Un criterio di split più intelligente

- di second'ordine (hessian) e un regolarizzatore

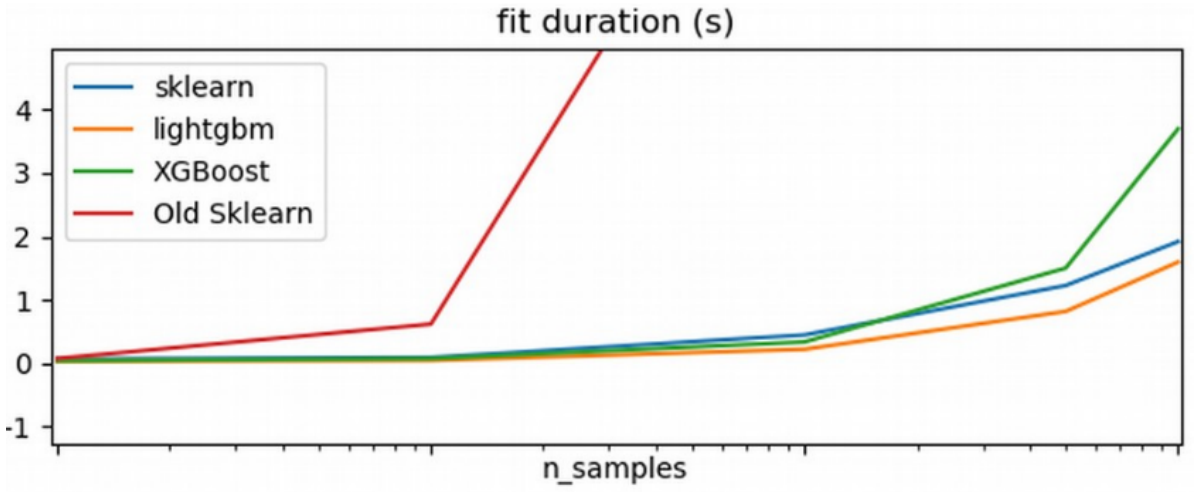
$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma$$

Aggressive sub-sampling

Fare subsampling sulle feature previene l'overfitting di più che farlo sui campioni

(rif. XGBoost: A Scalable Tree Boosting System, 2016)

Velocità di addestramento



- XGBoost e lightgbm batto il vecchio gbm

Confronti

GradientBoostingClassifier

- no binning
- single core
- sparse data support
- più *raffinato*

HistGradientBoostingClassifier

- binning
- multicore
- no sparse data support
- *missing value support*
- presto: monotonicity support
- presto: native categorical variables
- più resistente al rumore

XGBoost

```
conda install -c conda-forge xgboost
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
xgb.score(X_test, y_test))
```

- supports missing values
- GPU training
- networked parallel training
- vincoli di monotonicità (aumenta l'Explainability)
- supporta sparse data

LightGBM

```
conda install -c conda-forge lightgbm
```

```
from lightgbm.sklearn import LGBMClassifier
lgbm = LGBMClassifier()
lgbm.fit(X_train, y_train)
lgbm.score(X_test, y_test))
```

- supports missing values

- natively supports categorical variables
- GPU training
- networked parallel training
- monotonicity constraints
- supports sparse data

CatBoost

```
conda install -c conda-forge catboost
```

```
from catboost.sklearn import CatBoostClassifier
catb = CatBoostClassifier()
catb.fit(X_train, y_train)
catb.score(X_test, y_test))
```

- optimized for categorical variables
- uses one feature / threshold for all splits on a given level aka symmetric trees
- Symmetric trees are "different" but can be much faster
- supports missing value
- GPU training
- monotonicity constraints
- uses bagged and smoothed version of target encoding for categorical variables
- lots of tooling

Gradient Boosting Advantages

- Very fast using HistGradientBoosting (or XGBoost, LightGBM)
- Small model size
- Typically more accurate than Random Forests
- "old" GradientBoosting in sklearn is comparatively slow

Concluding tree-based models

When to use tree-based models?

- Model non-linear relationships
- Doesn’t care about scaling, no need for feature engineering
- *Single tree: very interpretable (if small)*
- *Random forests very robust, good benchmark*
- *Gradient boosting often best performance with careful tuning*