Ensemble Machine Learning

↳ Learning how to combine models to create stronger learners

↳ Perform strongly w/o overfitting

Outline:

→ Bias Variance tradeoff
→ Bootstrapping
→ Bagging
→ Rand. forests
→ AdaBoost (Xgboost) (Catboost)

Bias Variance Tradeoff

→ Bias
↳ error in model
High bias = simple models

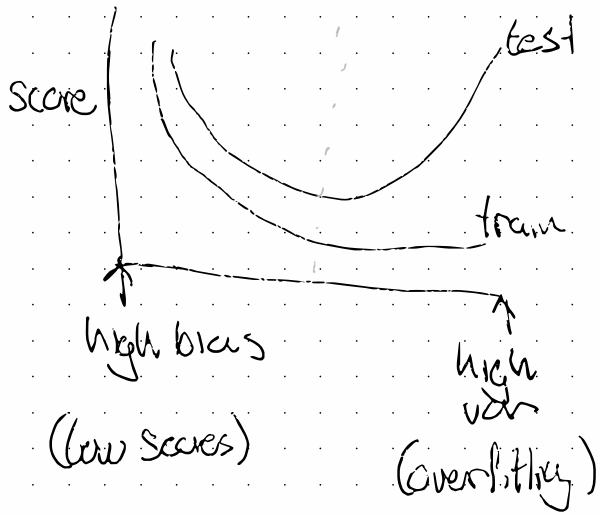
→ Variance
↳ flexibility in model
High var = complex models

→ Irreducible Error
↳ random error that cannot be accounted for

Bias variance tradeoff

As models get more complex - they don't generalise as well

↳ good, introduce just enough complexity for best score w/o overfitting



$$y = f(x) + \varepsilon, \quad \varepsilon \sim N(0, \sigma_\varepsilon^2)$$

ε: irreducible error which is normally dist.

y: data

$f(x)$: hypothetical func

$\hat{f}(x)$ = model

$$\therefore \text{err} = E\left[(y - \hat{f}(x))^2\right]$$

$$\therefore = E\left[(f(x) + \varepsilon + \hat{f}(x))^2\right]$$

let $\bar{f}(x) = E[\hat{f}(x)]$

then $err = E[(f(x) + \varepsilon - \hat{f}(x) + \bar{f}(x) - \bar{f}(x))^2]$

Add & subtract to get the same thing

next, group so that

$E[(f(x) - \bar{f}(x)) - (\hat{f}(x) - \bar{f}(x)) + \varepsilon)^2]$

↳ expand & use properties of ε to arrive at...

$= bias^2 + var + \sigma_\varepsilon^2$ ✓

How to find ideal parameters &/or models?

↳ cross validation

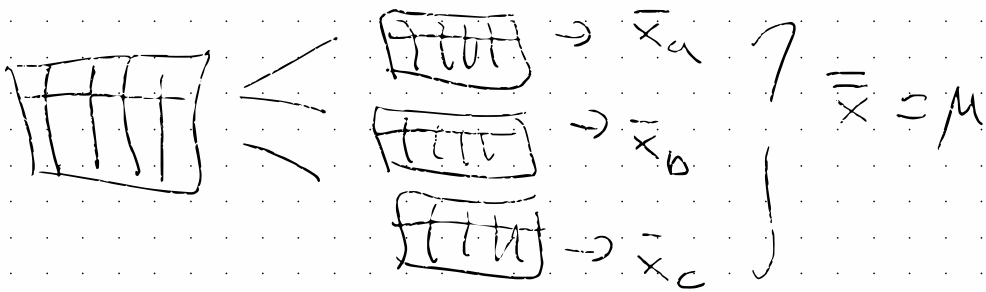
Bootstrapping (Resampling)

↳ Possible to reduce both bias AND variance

↳ Bootstrapping = tool used to achieve this

↳ sample sub-datasets from original w/ replacement

↳ possible to compute parameters from bootstrapped subsamples



→ Parameters:

$$E(\bar{\theta}_{B}) = \theta$$

$$var(\bar{\theta}_{B}) = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2$$

$$\begin{cases} \rho = corr(\hat{\theta}_i, \hat{\theta}_j) \\ var(\hat{\theta}_i) = \sigma^2 \end{cases}$$ where

↳ If $\rho = 0$ then $var(\bar{\theta}_{B}) = \frac{1}{B}\sigma^2$

→ Derivation of variance

$$E(\bar{\theta}_B) = E\left(\frac{1}{B}\sum_{i=1}^{B}\hat{\theta}_i\right) = \frac{1}{B}BE(\hat{\theta}) = \theta$$

suppose $E(\hat{\theta}) = \mu$, $\text{var}(\hat{\theta}) = E((\hat{\theta}-\mu)^2) = \sigma^2$

& rho(ρ) = correlation between bootstrapped samples

($S_B = \sum \hat{\theta}_i$, for notation's sake)

$$\text{var}(\bar{\theta}_B) = E\left(\left(\frac{1}{B}S_B - \mu\right)^2\right)$$

→ pull out $\frac{1}{B}$

→ expand sq. term

$$\text{var}(\bar{\theta}_B) = \frac{1}{B^2}E\left[S_B^2 - \overbrace{2\mu B S_B}^{\text{const}} + \overbrace{\mu^2 B^2}^{\text{const}}\right]$$

$$= \frac{1}{B^2}E\left[S_B^2\right] - \mu^2$$

$$E[S_B^2] = BE[\hat{\theta}_i^2] + B(B-1)E_{i\neq j}[\hat{\theta}_i\hat{\theta}_j]$$

using equations defined before

$$E_{xy}[\hat{\theta}_i \hat{\theta}_j] = \rho\sigma^2 + \mu^2$$

↳ insert & melt

$$\text{var}(\bar{\theta}_{RS}) = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2$$

↳ Perfect corr

then $\text{var}(\bar{\theta}_{RS}) = \sigma^2$

when using decision trees, decision boundaries don't correlate that well

↳ ends up reducing the variance

Bagging - "Bootstrap Aggregating"

↳ train multiple models & aggregate parameters (mean, etc.)

Pseudocode

```
models = [ ]
for b in [1 ... B]:
    model = model()
    Xb, yb = resample(X)
    model.fit(Xb, yb)
    models.append(model)

# predict
y_pred = np.mean(
        [mdl.pred(X) for mdl in models]
        , axis = 1
```

Stacking

↳ Aggregates so far are not weighted

add weighting

$$f(x) = \sum_{m=1}^{M} w_m \hat{f}_m(x)$$

← weight based on accuracy

$$\hat{w} = \underset{w}{\arg\min}\, E_{pop}[(y - f(x))^2]$$

$$\therefore \hat{w} = E_{pop}[(F(x)^T F(x))^{-1} F(x)^T y]$$

looks like linear regression

problem, we don't have $F(x)$ (can't use $f(x)$ since only best will be selected)

↳ solution:

train on all points except x_i, x_i

$$\hat{w}_{stack} = \underset{w}{\arg\min} \sum_{i=1}^{N} [y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x_i)]^2$$

how do you then solve this equation

↳ quadratic programming problem

↳ don't worry about solving it.

→ recall: this method is similar to LOOCV

Random Forest

↳ reduce correlation between different samples of Θ_b

↳ trees are good at capturing complex models w/ high depth, use this to create uncorrelated samples

→ don't just select random rows, also select random features

Train random forest pseudo code

```
for b in 1...B:
    Xb, yb = resample(X y)
    model = DTree()
    while not at terminal node
      and not max depth
        - select random features
        - choose best split from d features
        - add split to model
    append to models
```

Notes: Similar to bagging, but we also bag features

- This method != bagged dt
  - ↳ we've changed how splits are made

→ Method to build this class is very similar to original algorithm but splits change ✓

What happens if same columns are noise?

↳ This is why we use boosting

Rand. Forest advantage

↳ Requires little tuning (Plug & play)

↳ learners can go to arbitrary depth w/o much penalty

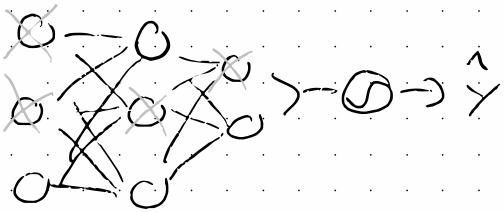
Good starting point above linear class./reg. ↳ provides good info about features.

↑ stat

"RF doesn't overfit" ← bagging/boosting powerfully prevents overfitting

Dropout: Connection w/ RF

↳ allows for uniform training over a network so as to prevent overreliance on a specific node



↳ The way how we train via bootstrapping of features & observations is

Ada boost

↳ Also good "plug & play" model

Bagging	vs	Boosting
low bias high variance learners		high bias low var. learners

↙

even after ensemble, var will remain low

Hypothesis: combo of weak learners makes a strong one.

↳ we weight each learner

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

↖ DT w/ max_depth = 1

→ Advantage: trains really fast

Notable differences:

- Labels = {-1, 1} instead of {0, 1}
- $F_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(x)\right)$
- Decision Boundary = 0

Algorithm:

- Add base model 1 at a time
  - ↳ train on all data
    - ↳ weight each obs
    - ↳ modify weight on each round
    - ↳ larger weights for wrong classifications
  - ↳ model weight is a function of error

Additive Modelling

Adaboost loss function: Exponential Loss

classification $\in \{-1, +1\}$

↳ cross entropy won't work, need smth else

$$L(y, f(x)) = e^{-y f(x)}$$

↖ same sign → 0
opp sign → inf
] never 0 or 1

re, we care abt sign more than how close it is to the real value

→ Application

$$(\alpha_m^*, f_m^*) = \underset{\alpha_m f_m}{\operatorname{argmin}} \sum_{i=1}^{M} e^{-y_i [F_{m-1}(x_i) + \alpha_m f_m(x_i)]}$$

↖ with model

note that loss can be expanded due to exp

$$J = \sum_{i=1}^{N} \underbrace{e^{-y_i f_{m-1}(x_i)}}_{\text{const}} e^{-y_i f_m(x_i)\alpha_m}$$

$$= \sum_{i=1}^{N} w_i^{(m)} e^{-y_i \alpha_m f_m(x_i)}$$

← historical vals back to root

$y_i f(x_i)$ will always be $+1$ or -1

∴ seperate into right & wrong sections

$$J = e^{-\alpha_m} \sum_{y_i = f_m(x_i)} w_i^{(m)} + e^{\alpha_m} \sum_{y_i \neq f_m(x_i)} w_i^{(m)}$$

$$\Rightarrow J = e^{-\alpha} A + e^{\alpha} B$$

→ then diff. & set to 0

$$\frac{\partial J}{\partial \alpha} = -e^{-\alpha} A + e^{\alpha} B = 0$$

$$\therefore \alpha_m = \frac{1}{2} \ln\left(\frac{A}{B}\right)$$

← weighted correct

← weighted incorrect

comparison to stacking

Stacking	Adaboost
• optimised using quadratic programming because of quad. loss func.	• can be diff easier
• CCCV	• trained w/ sample weights
• Not scalable O(NM)	• Greedy algorithm O(M) ↳ linear complexity

comparison to DL

activation func → Tanh

↳ $0 <$ range < 1

each node in network is like output of a logistic regression algorithm

Adaboost: $\hat{y} = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \text{sign}(w_m^T x_i) \right)$

DL: $\hat{y} = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \tanh(w_m^T x_i) \right)$

↑ hard vs soft outputs