

Maskininlärning

Mätsättning: Numeriskt representerad kunskap om en mängd observationer. Givet kunskapsvill vi förtäga framtidens observationer.

Tex: koefficienterna i en linjär ekvation.

Sannolikheten att två musikstücken är skrivna av samma kompositör

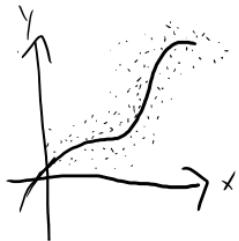
För att ha någon chans måste vi göra vissa antaganden; viktigt att komma ihåg i komplexa fall. Antaganden kan visa sig vara felaktiga / inte uppfyllda.

Grundantagande : Dataen representerar varians över

en distribution genererad av en

grundsaning. Om det finns

systematiska fel i dataen kan den förlovas.



För att kunna göra förutsägelser behöver vi något sätt att utvärdera kvalitén på kunskapen vi samlar.

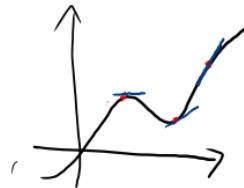
Dela därför stickprovet i en träningsmängd och en test-mängd.

Grundsaningen är okänd, så hitta linjära approximationer vidra punkterna i träningsdatan. Om vi får samma resultat

på testdatan är de linjära approximationerna vi lärt

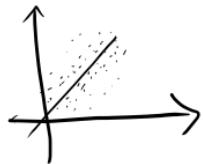
oss av träningsdatan också en god uppskattning av

grundsaningen. (Ärigen; under förutsättning att data till verkligen är distribuerad kring en grundsanning)



Differentialkalkyl (derivator) ger oss sådana linjära approximationer.

Enklast är förstas linjen (alltså ett enkelt linjärt beroende).



Enkel linjär regression:

$$Y = \beta_0 + \beta_1 X,$$

$\begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$ -matris = vektor

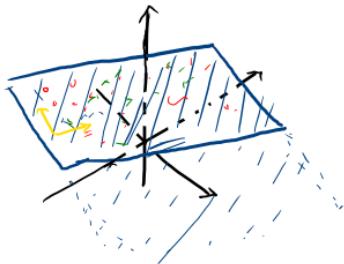
Koefficienterna är en matris med vektorer som beskriver skärningen med y -axeln och linjens lutning.

Statistiskt uttrycker vi detta

$$E(Y | X) = X\beta$$

"Väntevärde av Y , givet X , är linjärt beroende till X "

Multipel linjär regression



Datan i 3D runt \hat{Y} , fördelad kring planet.

Vi uppskattar "lutningen på planeten",
gradienten:

$$\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right) = \nabla f(x_1, x_2) \quad (p=2)$$

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

i p dimensioner.

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad \frac{1}{n-p-1} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = MSE = \text{Bias}^2 + \text{Varians} + bryns$$

$$RMSE = \sqrt{MSE} \approx \text{avvikelse (endast standard om centrerat)}$$

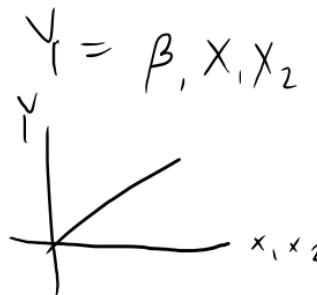
Allmän linjär regression

Vi kan linjärisera andra förhållanden mellan X och Y.

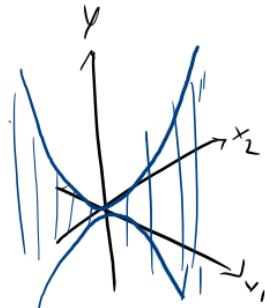
Ett rätt framt sätt är interaktionseffekter mellan parametrar:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$$

Redan en 4D-graf. Låt oss förenkla.



men i termor av
 X_1, X_2 separata.



Sedelgta, icke-linjär så
det förslår!

polynom-expansion: $y = f(x_1, x_2)$:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2$$

Fran 3D-graf till 6D-graf....

Genom att räkna i högre dimensioner kan vi hitta linjära modeller av icke-linjära relationer. En sorts linjärisering.

Men, matrismetoder kostar i allmänhet $\mathcal{O}(n p^2)$ där n är antalet träningspunkter och p är antalet dimensioner.

"Därför att lägga till parametrar, billigt med mer data"

Nu är

$$E(y|x) = f(x)$$

f här är en feature-map. Vi omvandlar alltså x först.



För att beräkna koefficienterna - alltså lära oss vete -
optimerar vi en kostnadsfunktion.

$$\text{OLS} \begin{cases} C(\beta) = \sum_{i=1}^n (y_i - \hat{y})^2 & (\text{RSS}) \\ C(\beta) = \frac{1}{n-p-1} \sum_{i=1}^n (y_i - \hat{y})^2 & (\text{MSE}) \end{cases} \quad \left[\text{även } \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 \right]$$

MSE är oft föredra av statistiska skäl. MSE kan teoretiskt delas
upp i tre delar:

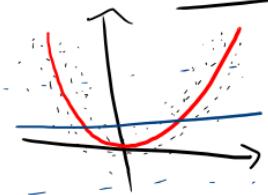
från fördelningen kring grundsättningen

$$\text{MSE} = \text{Bias}^2 + \text{Varians} + \text{Brus}$$

Vi vill ha (tillräckligt) låg varians (hög precision) och låg bias (hög noggrannhet)
Dvs 9.33 ± 0.11

Om Bias dominerar över bruset men vi har låg varians \rightarrow underfit

underfit

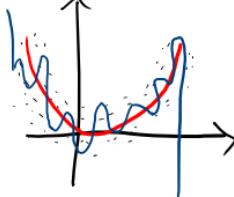


Hög bias = låg noggrannhet = stort konfidensintervall

Låg varians = hög precision

$$9.312789113 \pm 8.2$$

Overfit



Låg bias = hög noggrannhet = litet konfidensinterval

Hög varians = låg precision

$$9.1 \pm 0.00134$$

Alla punkterna är verkligen nära linjen, men bara just de punkterna är nära linjen.

Varians är den som i de flesta fall skall minskas:

- reguljärisering \rightarrow kostnadsfunktionen

- dimensionsreducering \rightarrow feature-mängden

Reguljärisering:

$$C(\beta) = \text{MSE} + \lambda \sum_{i=1}^p \beta_i^2$$

"Ridge-regression" använder ℓ_2 -normen, $|u| = \sqrt{u_1^2 + \dots + u_p^2}$, euklidiskt

Tendrar att hålla β liten, vilket motverkar varians (sma värden \rightarrow sma förändringar). Fördelar vikterna mellan alla parametrar, dvs "jämnar ut" deras relevans. λ styr tendensen.

$$C(\beta) = \text{MSE} + \lambda \sum_{i=1}^p |\beta_i|$$

Lasso regression använder ℓ_1 -normen, $|u| = \sum_{i=1}^p |u_i|$, taxicabs

Tenderar att sätta koeficienter till 0. λ styr denna tendens.

Om λ är för stort tenderar modellen snabbt mot underfit.

Dimensionsreduktion:

Feature-engineering

- Forward Selection

Testa alla p-parametrar individuellt, välj den med högst förklaringsgrad.

Lägg till varje p-1 återstående var för sig, —||—

Repetera tills R^2 slutar öka. (eller MSE slutar minnas)

- Backward Elimination

Testa alla p-parametrar i en modell.

Testa alla modeller med p-1 parametrar.

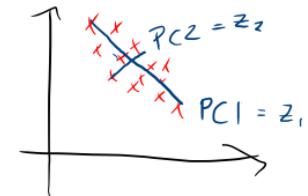
Om någon har bättre R^2 /MSE, välj den.

Upprepa tills R^2 /MSE slutat förbättras.

- Orthogonalisering (PCA) (kan minska dim; frikopplar features)

Omvandla $\mathbb{X} \rightarrow \mathbb{Z}$ där $Z_i = \phi_{1i}x_1 + \dots + \phi_{pi}x_p$

Alltså varje Z är linjär komb. av X



Mer om kostnadsfunktionen:

$$\min C(\beta) = \text{MSE} + r(w)$$



Kan ändra denna också förstås!

Tex Maximal margin, Maximum Likelihood

Istället för att hitta kortaste avståndet till alla punkter
kan vi hitta största avståndet till de närmsta punktarna.

$$\max C(\beta) = \sum_{i=1}^n \hat{\alpha}_i \langle x, x_i \rangle + r'(\beta) \quad \begin{matrix} \text{bi-viktor} \\ (\text{rentetsnormal}) \end{matrix} \quad \sum_{i=1}^n \beta^2 = 1$$

SVM med linjär kernel = linjär klassifikator ($\begin{cases} y > 0 : 1 \\ y \leq 0 : 0 \end{cases}$)

Istället för att expandera feature-mängden så expanderas en kernel datamängden, dvs beräkningarna ske i högre dimensioner.

polynom kernel:

RBF:

Andra kostnadsfunktioner ändrar regressionstyp.

Maximum likelihood kan uppsätta Bayes Sats direkt.

$$P(Y=1|X) = \frac{P(X|Y=1) \cdot P(Y=1)}{\text{relativ frek i träningsdata}}$$

Naive Bayes

Antag dist $P(X)$ \leftarrow konstant (X har redan antagit sina värden)
 P_i denna, räkna väntevärde.

Bernoulli NB

ok klassificerare, usel uppsättare

Gaussian NB

Skalning nödvändigt pga alla antaganden. std-normal rit företräda.

Slutligen kan vi ge upp antagandet om oberoende normalfördelning kring hyperplanet. Detta leder till andra regressionstyper igen:

Om $X_y \sim \text{Binomial}$ så har vi logistisk regression:

X räknar då antalet lyckanden för n stycken Bernoulli försök med sannolikhet p .

Varietrad i indata η är alltså ett Bernoulli försök:

$$\hat{p}(x) = \frac{e^{x\beta}}{1 + e^{x\beta}} = \frac{1}{1 + e^{-x\beta}} = \mu$$

För hela X : $\ln\left(\frac{\mu}{n-\mu}\right)$ \leftarrow länk funktion, söger hur vi skall transformera X till ett optimeringsvillef.

Odds att lyckas (fölhör klassen):

$$\frac{P(X)}{1-P(X)} = \frac{e^{XB}}{1+e^{XB}} = \frac{e^{XB}}{\left(1+e^{XB}\right) - e^{XB}} = e^{XB}$$

$$\ln\left(\frac{P(X)}{1-P(X)}\right) = XB \quad \text{"logit-odds"}$$

Vi har: Allmän Linjär Regression

a) $\ln\left(\frac{P(X)}{1-P(X)}\right) = XB$ en linjär modell

b) $n = \ln\left(\frac{\mu}{n-\mu}\right)$ en linkfunktion som omvandlar X, Y till optimeringsvillkor

c) en kostnadsfunktion $C(B) = \min \sum_{i=1}^n (-y_i \ln P(x_i)) - (1-y_i) \ln(1-P(x_i)) + r(w)$
i någon mening "MSE" för Log. reg. / sannolikheter

notera:

logistisk regression
behöver skalning.

sannolikheter och medelvärden
→ om X är dimensionerade
viktade enligt de skalorna)

Stochastic Gradient Descent

Iterativ metod för att hitta närmevärde.



Istället för ett djelet optimeringsproblem som vi löser analytiskt med matrismetoder så itererar vi en funktion över vikterna:

$$\beta_{i+1} = \beta_i - \eta \nabla C(\beta)$$

För stokastisk gradient descent beräknas uppdateringarna för en datapunkt i taget, och parametrarna justeras lite i taget.

Mini-batch väljer en slumpmässig batch av punkter.

Batch kör över alla datapunkter på en gång.

Krävs fortfarande differentierbara funktioner, men optimeringsproblemet behöver inte ha analytiska lösningar.

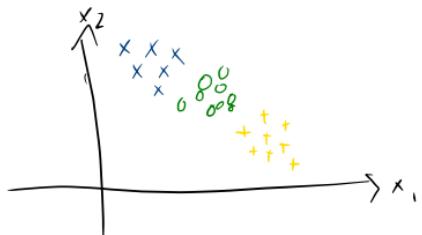
k - Means Inte längre en GLM (Allmän Linjär Modell)

optimerings villkor: $\min_{c_1, \dots, c_k} \left[\sum_{k=1}^K \frac{1}{|C_k|} \sum_{i: i \in C_k} \sum_{j=1}^p (x_{ij} - \bar{x}_{\cdot j})^2 \right]$

Notera att y inte förekommer. Alltså ett exempel på

Oövervakad inlärning. Genererar istället ett Y !

Minimerar avståndet mellan punkterna i k kluster, givet X .



k-NN

En annan sorts ML algoritm. Denna gör i princip alltid en "overfit" och läser sig hela träningsmängden. "Similarity measure", dvs beräkningen av "närmst" är avgörande.

Kallas en icke-generalisering algoritm.

En del av många andra tekniker.

"brute force" fortfarande giltigt sätt att beräkna -

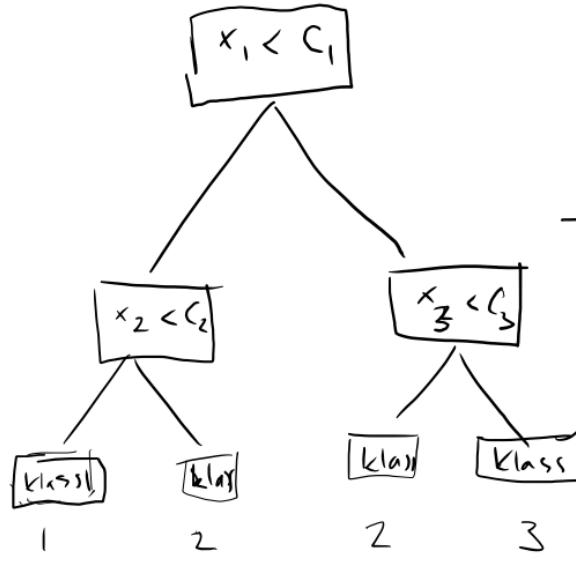
dvs räkna avstånd/likhet över hela indata.

I princip ett sorter-problem $O(n \log n)$.

Beslutsträd

Klassisk AI (GOFAI, Good Old Fashioned AI)

Konstruerar ett binärt träd över \mathbb{X} :



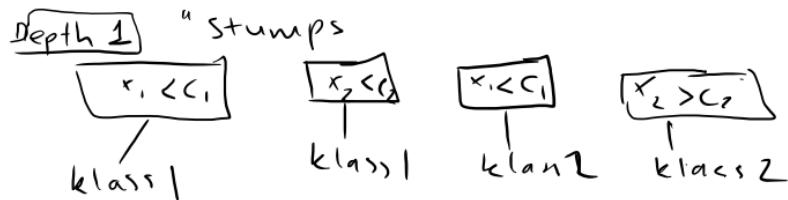
Träden kan styckas upp (de är linjära).

klass 1: $x_1 < c_1, x_2 < c_2$

klass 2: $x < c_1, x_2 > c_2$

$x_1 > c_1, x_3 < c_3$

klass 3: $x_1 > c_1, x_3 > c_3$



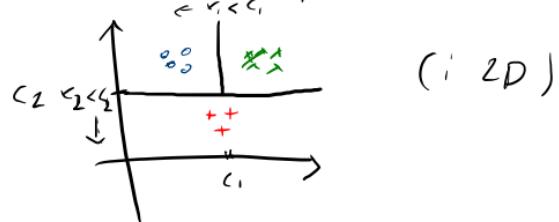
Random forest:

Väljer slumpmässigt $x \in \mathbb{X}$ att göra splits. Man väljer träd-alla röstar. I praktiken slår detta de mest avancerade splits (entropi, GINI) genom att undvika bias och utnyttja datans σ (sprider sina features uniformt).

Måste hyperparametrar optimeras.

Okänslig för skalning.

Ger rätvinkeliga beslutsgränser:

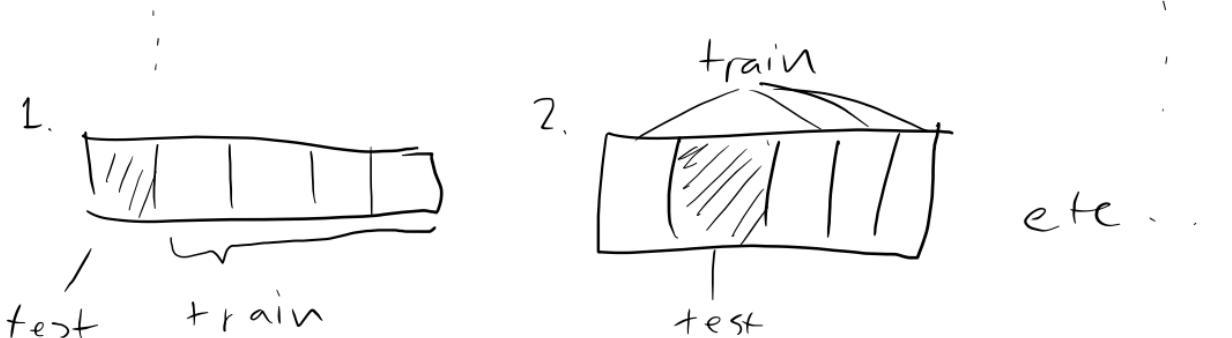


Träning, tuning, utvärdering:

k-fold cross-validation:

dela datan i k delar.

utesluta en, k_1 , träna på övriga, testa på k_1
välj nytt k_2 , — || — — || — k_n



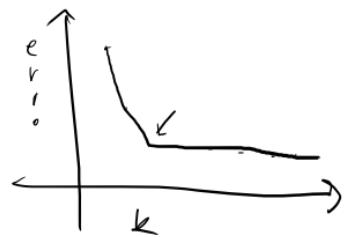
Tydlig gör varians och känslighet för outliers etc.

Utilnyttjar hela datamängden för train och test.

hyperparameter optimering:

testa parametrar, välj de som presterar bäst.

elbow-plots



silhouette (k -means)



gridsearchCV gör både k-fold cross validation

och hyperparameter optimering!

RMSE, MSE, confusion Matrix, classification report

MSE och RMSE mitt på hur bra en värdesregression är.
 RMSE lättare att tolka, har samma enheter som datan.
 För klassifikation:

| | precision | recall | f1-score |
|---|--------------------|---|--|
| 0 | $\frac{TN}{TN+FN}$ | $\frac{TN}{TN+FP}$ | $2 \times \frac{Pr \cdot Re}{Pr + Re}$ |
| 1 | $\frac{TP}{TP+FP}$ | $\frac{TP}{TP+FN}$ | för varje klass |
| andelen av klassific. som var rätt (förmängn $\rightarrow < 1$) | | andelen av klassen som täcktes (för fä $\rightarrow < 1$) | |
| accuracy: $\frac{TP+TN}{ALL}$ | | | |

| | | |
|---|----|----|
| 0 | TN | FP |
| 1 | FN | TP |

hög precision: undvikar FN på 0
 hög recall: undvikar FP på 1
 FP på 0
 FN på 1