Machine Learning In Finance

SUPERVISED LEARNING (Chapter 2)

Supervised learning: we have pairs $(x_1, y_1), \dots, (x_n, y_n)$ where x_i = feature vectors and y_i = label / response. We want to construct a function $Y = F(X)$.

Regression: when we have continios output (ex. pricing problem)

Classification: categorical output (ex. which firms will go out of business).

Linear regression: in linear regression we want to approximate the model with a dipendent random variable Y that satisfies:

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \varepsilon$$

where $Y, \beta_0, \beta_i, X_i, \varepsilon \in \mathbb{R}$ and ε is the error.

Obs: Implicitely we're saying that $\mathbb{E}[Y \mid X_1, \dots, X_p]$ is approximately linear.

Obs: X_i can also be dummy (fittizie (es maschio/femmina $\rightsquigarrow$ 1/0)) variables (they are binary) that encode the levels of qualitative inputs; an input with K levels would require $K-1$ dummy variables. → (ex. red, green, blue: dummy_1: 1 if red, 0 if no; dummy_2: 1 if green, 0 if no; dummy_3: 1 if blue, 0 if no; but $\sum_{i=1}^{3}$ dummy_i = 1)

↳ **Minimizing the residual sum of squares**: we are given the training data $(y_1, x_1), \dots, (y_N, x_N)$ (x_i: vectors).

Set $\beta := \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix}$, $y := \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$, $X := \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix}$ ← features vectors ($= x_1$)

We want to minimize respect to β:

$$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 = \boxed{\min_{\beta} \| y - X\beta \|_2^2}$$

If we set $\hat{y}_i = \beta_0 + \sum_{j=1}^{p} x_{ij} \beta_j$ we want to minimize the residual sum of squares:

$$\boxed{RSS := \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 =: \hat{\varepsilon}^T \hat{\varepsilon}}$$

(this is a general definition where $\hat{y}_i$ doesn't need to be linear)

↳ **Computation to find the minimum:**

We want $\nabla \| y - X\beta \|^2 = 0$. So:

$$\frac{\partial}{\partial \beta_k} \left(\sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 \right) = -\sum_{i=1}^{N} 2 \left(y_i - \underbrace{\beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j}_{(X\beta)_i \text{ scalar}} \right) \cdot x_{ik}$$

$$= -2 \, X^{k} \cdot (y - X\beta)$$ (X^k: k^{Th}-column). And so:

$$\nabla \|y - X\beta\|^2 = -2 X^T (y - X\beta) = 0 \implies \boxed{\hat{\beta} := (X^T X)^{-1} X^T y}$$ (when $(X^T X)^{-1}$ exists)

$X^T X$ is a square-matrix but X is not

So $\boxed{\hat{y} = X\hat{\beta} = \underbrace{X (X^T X^{-1}) X^T}_{:= H \text{ projection matrix}} y}$ and $\boxed{\hat{\varepsilon} := y - \hat{y} = (Id - H) y}$.

→ Normal errors: If $\varepsilon \sim N(0, \sigma^2 I)$:

$$\hat{\beta} = (X^T X)^{-1} X^T y \underset{y = X\beta + \varepsilon}{=} \beta + (X^T X)^{-1} X^T y = \beta + (X^T X^{-1}) X^T \varepsilon \sim N(\beta, (X^T X)^{-1} \sigma^2)$$

$(Var(Ab) = A\, Var(b)\, A^T)$.

Furthermore, σ^2 can be estimated with the sample variance: $\hat{\sigma}^2 = \frac{1}{N-p-1} \sum_{i=1}^{N} (y_i - \hat{y}_i^2) = \frac{RSS}{N-p-1}$

Obs.: with normal distributed errors, OLS = MLE

Obs.: Weighted least squares if variance is heteroskedastic (→ sui dati non sulle features), $\varepsilon \sim N(0, \Lambda^{-1})$ $\Lambda := diag(\sigma^2(x_n))$

we have $\hat{\beta} = (X^T \Lambda X)^{-1} X^T \Lambda y$ ← we use ΛX instead of X in the definition

$\Lambda^{-1} = diag\left(\frac{1}{\sigma^2 x_n}\right)$

→ Numerical issues: — If $X^T X$ is not invertible we drop the redundant columns of X.

→ QR decomposition: If $N >> p$ (→ it is convenient for computational cost), inverting $X^T X$ can be avoided using $X = QR$ where $Q \in \mathbb{R}^{N \times (p+1)}$ has orthonormal columns and $R \in \mathbb{R}^{(p+1)\times(p+1)}$ is upper-triangular:

$$\hat{\beta} = (X^T X)^{-1} X^T y = ((QR)^T QR)^{-1} (QR)^T y = (R^T \underbrace{Q^T Q}_{Id} R)^{-1} R^T Q^T y = R^{-1} \underbrace{R^{-T} R^T}_{Id} Q^T y = R^{-1} Q^T y$$

$$\Downarrow$$

$$\boxed{\hat{\beta} = R^{-1} Q^T y}$$

→ Singular Value decomposition (SVD): $X \in \mathbb{R}^{N \times (p+1)}$, $X = UDV^T$ with $U \in O(N)$, $V \in O(p+1)$ (↑ is a change of basis)

and $D \in \Pi(N, p+1)$ diagonal in this sense:

$N = p+1$: $\begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_N \end{pmatrix}$ $N > p+1$: $\begin{pmatrix} \sigma_1 & & \\ & \ddots & \sigma_n \\ & & \end{pmatrix}$ $N < p+1$: $\begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}$

(σ_i are the singular values)

$$\hat{\beta} = (X^T X)^{-1} X^T y = (V D \underbrace{U^T U}_{Id} D V^T)^{-1} V D U^T y = V^{-T} D^{-2} \underbrace{V^{-1} V}_{Id} D U^T y = V^{-T} D^{-1} U^T y \underset{V^{-T} = V}{=} \boxed{V D^{-1} U^T y}$$

Inverting D is much easier.

→ **Model selection**: If $p+1 << N$ and we want to drop q columns we have two hypothesis:

$H_0: \beta_{p-q+1} = \dots = \beta_{p-1} = \beta_p = 0$ vs H_1: at least one $i \geq p-q+1$ s.t. $\beta_i \neq 0$

we use F-statistic: $F = \dfrac{(RSS_0 - RSS)/q}{RSS/(N-p-1)}$ with RSS_0 is the RSS without the last q variables

(F is always ≥ 0 because $RSS_0 \leq RSS$ and $N-p-1 < 0$)

→ **Potential problem with linear regression**: ① $\mathbb{E}[Y|X]$ is not linear (so use non linear trasf. for predictors)

② σ has not zero correlation, i.e. $\sigma \neq \sigma^2 Id$

③ Heteroskedasticity of error terms

④ Outliers: y_i far from the predicted value of $\hat{\beta}^T x_i$: could be genuine or data error

⑤ High-leverage points: these are points whose presence has a large impact on the fitted model (we can find them computing $h_i := x_i (X^T X)^{-1} x_i^T$ where x_i is the i-th observation).

⑥ Collinearity and multicollinearity: 2 or more predictors are highly correlated and it is difficult to disentangle the individual effects (solution is to drop some variables or combine them in one predictor).

Log-model / linear regression: Usually when we want to predict a price we have:

cambio sistema di riferimento

$\begin{cases} X \\ Y \end{cases} \rightsquigarrow \begin{cases} X \\ \ln Y \end{cases}$ $\ln Y = \ln X + c$ $\rightsquigarrow$ $e^{\ln Y} = e^{\ln X} e^c$

$Y = e^c \cdot X$

Linear regression with basis function:

$$Y = \beta_0 + \sum_{i=1}^{M} \beta_i \varphi_i(X) + \varepsilon$$

with $\varphi_i : \mathbb{R}^p \longrightarrow \mathbb{R}$ the i-th basis function.

$X = (x_1, \dots, x_p)$

To have the linear model we set $M = p$ and

$\varphi_i : \mathbb{R}^p \longrightarrow \mathbb{R}$ $(x_1, \dots, x_p) \longmapsto x_i$

Ex $\varphi(x) = \dfrac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} e^{-\frac{1}{2}\|x-\mu_i\|_2^2}$

Parameter estimate is: $\hat{\beta} = (\Psi^T \Psi)^{-1} \Psi^T y$ $\quad \Psi = \begin{bmatrix} 1 & \psi_1(x_1) & \cdots & \psi_n(x_1) \\ \vdots & \vdots & & \vdots \\ 1 & \psi_1(x_N) & \cdots & \psi_n(x_N) \end{bmatrix}$

<u>Bias - Variance decomposition</u>: Suppose $y = f(X) + \varepsilon$ the true model e $\hat{f}(X, D)$ our estimate where D is the dataset from which $\hat{f}$ learns. Furthermore, soppose $\mathbb{E}_D[\varepsilon] = 0$ and $Var_D(\varepsilon) = \sigma_\varepsilon^2$; ε and $\hat{f}$ indipendent (we are computing $\mathbb{E}$ and Var with respect to the distribution we have by the dataset D). It holds:

$$\boxed{MSE = Bias(\hat{f})^2 + Var(\hat{f}) + \sigma_\varepsilon^2}$$ where $MSE := \mathbb{E}_D[(y - \hat{f})^2]$ and $Bias(\hat{f}) := f - \mathbb{E}_D[\hat{f}]$

<u>Proof</u>: $\mathbb{E}_D[(y - \hat{f})^2] = \mathbb{E}_D[y^2 + \hat{f}^2 - 2y\hat{f}] = \mathbb{E}_D[y^2] + \mathbb{E}_D[\hat{f}^2] - 2\,\mathbb{E}_D[y\hat{f}]$.

Now:

$\mathbb{E}_D[y^2] = \mathbb{E}_D[f^2] + 2\mathbb{E}_D[f\varepsilon] + \mathbb{E}_D[\varepsilon^2] \overset{f \text{ is } D\text{-indipendent}}{=} f^2 + 2f \cdot 0 + \sigma_\varepsilon^2 = f^2 + \sigma_\varepsilon^2$;

$\mathbb{E}_D[\hat{f}^2] = Var_D[\hat{f}] + \mathbb{E}_D[\hat{f}]^2$;

$\mathbb{E}_D[y\hat{f}] = f\,\mathbb{E}_D[\hat{f}] + \mathbb{E}_D[\varepsilon\hat{f}] \overset{\varepsilon \text{ ind. } \hat{f}}{=} f\,\mathbb{E}_D[\hat{f}] + 0 \cdot \mathbb{E}_D[\hat{f}] = f\,\mathbb{E}_D[\hat{f}]$

So:

$$\mathbb{E}_D[(y - \hat{f})^2] = f^2 + \sigma_\varepsilon^2 + Var_D(\hat{f}) + \mathbb{E}_D[\hat{f}]^2 - 2f\,\mathbb{E}_D[\hat{f}] =$$

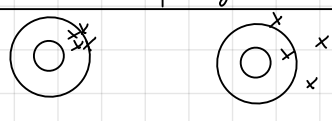
$$= (f - \mathbb{E}_D[\hat{f}])^2 + Var_D(\hat{f}) + \sigma_\varepsilon^2 .$$

□

<u>Bias</u>: tendency to consistently learn wrong thing ($f - \mathbb{E}[\hat{f}]$: on average I miss the true model)

<u>Variance</u>: tendency to learn random things irrespective of the real signal ($\mathbb{E}[\hat{f} - \mathbb{E}[\hat{f}]^2]$)

↳ We can't do anything for the irreducible error. We can control or bias (decreases with model complexity) or variance (increases with model complexity).

	Low variance	High variance
High bias		
Low bias		

Choosing the right set of predictors :

Subset selection : Keep subset of indipendent variables

Regularization / shrinkage : try to avoid overfitting using penalization methods

↗ Next Chapter 3

We will discuss them in a bit, first of all let's introduce TSS:

$$TSS := \sum_{i=1}^{n} (y_i - \bar{y})^2$$ where y_i in the dataset and $\bar{y}$ the mean

$$R^2 := 1 - \frac{RSS}{TSS}$$ (it follows immediatly as $\frac{RSS}{TSS} < 1$ because RSS is constructed to minime $\sum_{i=1}^{n} (y_i - z_i)^2$ as $(z_i)_i$ vary, so in particular when $z_i = \bar{y}$ $\forall i$. So, $0 < R^2 < 1$)

$$MSE := \frac{1}{N} RSS$$ $$RMSE := \sqrt{MSE}$$ $$MAE := \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$ ← punishes less a large error (obviously)

External validity : divide dataset in training set and test, obviously the training set MSE will understimate the test set MSE. So what can we do?

→ **Comparing model's performance**:

- $C_p := \frac{1}{N}(RSS + 2p\hat{\sigma}^2)$ with $\hat{\sigma}^2$ unbiased estimate of σ^2 (C_p became an unbiased estimate of the test MSE) (least squa. method)
- $AIC := \frac{1}{N\hat{\sigma}^2}(RSS + 2p\hat{\sigma}^2)$
- $BIC := \frac{1}{N}(RSS + \log(N)\, p\hat{\sigma}^2)$ (penalizes models with many param.)
- $\text{Adj } R^2 = 1 - \frac{RSS/(N-p-1)}{TSS/(N-1)}$

→ **K - cross validation** :

hp.: the underlying data contains indipendent sample and is identically distributed (this hp. don't fit with time series)

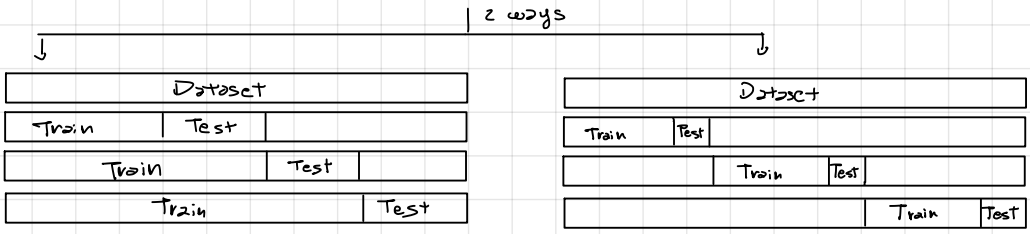
Algorithm :
- Separate data set in K sub-dataset
- For $j = 1 : K$: use all subsets except j as training set. Use j-subset as test.
- The final statistics is the average of the K statistics

Remark : when $K = N$ (# datase) we call it **Leave - one - out cross validation**. Better with small data.

Remark : • Its focus his on predictive data rather then on estimation of parameters

• Use out-sample comparisons ⟹ unbiased comparison.

With time series : make no sense to predict frome the future something in the past, se:



Choosing set of predictors (Chapter 3)

→ Subset selection : Step:

① $\mathcal{M}_0$ be the null model which contains no predictor. This model simply gives the mean of the data to each observation

② For $k = 1, \dots, p$, we train all the $\binom{n}{k}$ models that contain k predictors; pick the best w.r.t. RSS or R^2

③ Select a single best model $\mathcal{M}_0, \dots, \mathcal{M}_p$ using cross-validated pred. error, AIC, BIC or adj. R^2.

Problem: for p too large is too complex.

→ Forward stepwise selection : Step:

① $\mathcal{M}_0$ be the null model, as above.

② For $k = 0, \dots, p-1$ we define $\mathcal{M}_{k+1}$ as the model with the same predictors of $\mathcal{M}_k$ + the predictor $\bar{p}$ s.t $\mathcal{M}_k \cup \{\bar{p}\}$ is the best model, i.e. the smallest RSS or highest R^2. In this step we have to fit $p-k$ model.

③ Same as subset selection

→ Backward stepwise selection : Step

① Same as sub selection;

② For $k = p, p-1, \dots, 1$: we define $\mathcal{M}_{k-1}$ as the model with same predictors of $\mathcal{M}_k$ - the predictor $\bar{p}$ s.c. $\mathcal{M}_k - \bar{p}$ is the best model.

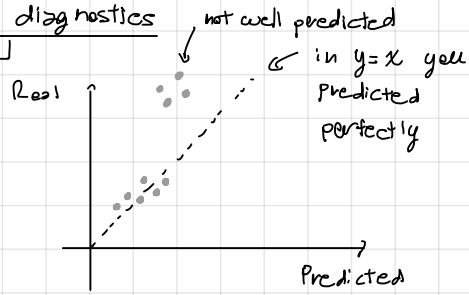
③ Same as sub. selection.

→ Hybrid methods : • forward and backward at each step

• Once a model has been chosen, run model diagnostics

compare prediction to data in test set: example →

So we can try to conditioning on some predictors value



Regularization : technique used to prevent overfitting and improve the generalization of a model. Regularization methods add a penalty term to model's loss function penalyzing large parameter values.

→ Ridge regression : $\hat{\beta}^R := \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \| y - X\beta \|^2 + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right\}$ (*) (increasing λ increases shrink; loss function) or equivalently

$\hat{\beta}^R = \operatorname{argmin} \left\{ \frac{1}{2} \| y - X\beta \|^2 \right\}$ subject to $\sum_{j=1}^{p} \beta_j^2 \leq s$. Also called L2 regularization

→ Step: (1) β_0 is not shrunk, $\hat{\beta}_0 := \bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$

(2) Scale predictors so $x_{ij} \longrightarrow x_{ij} - \overline{x_j}$ (mean = 0) (also we can put variance=1).

So we solve (*) with β where β_0 is dropped. The solution is:

$$\hat{\beta}^R = \left(X^T X + \lambda I \right)^{-1} X^T y$$

↳ it is always invertible! ($\forall \lambda > 0$) (Using QR or SVD also)

Remark : Ridge regression outperforms least squares because is capable of trading off a small increase in bias for a larger decrease of variance (since penalizes complex models).

→ Bayesian view of ridge : suppose $\mathbb{P}[y \mid X, \beta] \sim N(X\beta, \sigma^2 I)$ and $\beta \sim N(0, \tau^2 I)$ with $\tau > 0$. So :

$$\mathbb{P}[\beta \mid y, X] = \frac{\mathbb{P}[\beta, y, X]}{\mathbb{P}[y, X]} = \mathbb{P}[y \mid X, \beta] \cdot \frac{\mathbb{P}[X, \beta]}{\mathbb{P}[y, X]} =$$

$$= \mathbb{P}[y \mid X, \beta] \cdot \mathbb{P}[\beta] \cdot \frac{\mathbb{P}[X \mid \beta]}{\mathbb{P}[y, X]} \propto \mathbb{P}[y \mid X, \beta] \cdot \mathbb{P}[\beta] \propto$$

$$\propto \exp\left(-\frac{1}{2\tau^2} \| \beta \|_2^2 \right) \exp\left(-\frac{1}{2\sigma^2} \| y - X\beta \|_2^2 \right) (*)$$

So, maximizing $\mathbb{P}(\beta \mid y, X]$ (maximum a priori - MAP) is $\max_{\beta}$ (*) $\iff$

$\min_{\beta} \left\{ \frac{1}{2} \| y - X\beta \|^2 + \frac{\sigma^2}{2\tau^2} \| \beta \|_2^2 \right\}$ $\Rightarrow$ is the ridge regr. with $\lambda = \frac{\sigma^2}{2\tau^2}$. Note that in the Bayesian approach λ is a priori ($\notin \pi|_-$)

→ Lasso : $\min_{\beta} \left\{ \frac{1}{2} \| y - X\beta \|^2 + \lambda \| \beta \|_1 \right\}$, $\| \beta \|_1 = \sum_{j=1}^{p} |\beta_j|$ (Ridge with $\| \cdot \|_1$)

also used for subset selection

that is equivalent to $\underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \|y - X\beta\|_2^2 \right\}$ subject to $\sum_{j=1}^{p} |\beta_j| \leq s$

Remark : • No closed-form to solve

• Easy to solve, not computationally more expensive than ridge

• $\|\cdot\|_1$ shrinks more than $\|\cdot\|_2$ so Lasso encourages the model to have more zeros parameters $\Rightarrow$ sparse models.

→ Ridge VS Lasso :

• Example : $N = p$, $X = Id$, no intercept.

Least square : $\min_{\beta} \sum_{j=1}^{N} (y_j - \beta_j)^2 \Rightarrow \hat{\beta}_j = y_j$

Ridge regression : $\min_{\beta} \sum_{j=1}^{N} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \Rightarrow$ use the formula $\beta_j^R = \frac{y_j}{1+\lambda}$

Lasso : $\min_{\beta} \sum_{j=1}^{N} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$: $\frac{\partial}{\partial \beta_j} \Rightarrow = -2(y_j - \beta_j) + \lambda \operatorname{sgn}(\beta_j) = 0$

$y_j = \beta_j + \frac{\lambda}{2} \operatorname{sgn}(\beta_j)$,

if $y_j \leq -\frac{\lambda}{2} \Rightarrow \beta_j < 0$ (by contraddiction) $\Rightarrow \beta_j = y_j + \frac{\lambda}{2}$

if $y_j \geq \frac{\lambda}{2} \Rightarrow \beta_j > 0$ (//) $\Rightarrow \beta_j = y_j - \frac{\lambda}{2}$

if $-\frac{\lambda}{2} < y_j < \frac{\lambda}{2} \Rightarrow \beta_j = 0$ (by contraddiction)

• Lasso can be biased easier (tend to set to zero more parameters then ridge $\Rightarrow$ simpler model $\Rightarrow$ higher bias).

• Lasso not satisfy the oracle property (oracle property: when $N \to +\infty$ can identify the right zero parameters with prob = 1).

↓

Adaptive Lasso : $\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|^2 + \lambda \sum_{k=1}^{n} \omega_k \|\beta_k\| \right\}$ where ω_k are the weights and are obtained as $\omega_k := \frac{1}{|\tilde{\beta}_k|}$ where $\tilde{\beta}_k$ are the coefficients from ols or Bridge. It satisfy the oracle property.

→ Group Lasso : we define a partition of the indexes $\{1, \dots, p\}$ of the features (i.e. we do a partition of the features), so we suppose to have $\beta_1, \dots, \beta_m$ index :

$$\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|^2 + \lambda \sum_{k=1}^{m} \|\beta_k\|_2 \right\} \quad (\text{when } m = p \Rightarrow \text{Lasso})$$

→ Other shrinkage methods: • Composite norm methods: $\min_{\beta} \left\{ \|y - X\beta\| + \lambda \sum_{k=1}^{m} \|\beta_k\|_2 \right\}$

• Elastic nets: $\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|^2 + \lambda \left((1-\alpha)\frac{1}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right) \right\}$

Note that all the methods we did preserve linearity (if X is solution, $\alpha X + \beta Y$ is solution (module the constraints)).

↓

Classification and regression trees (CART): decisions are modelled as a series of binary decisions (splits). The output of CART is a set of prediction rules and predicted values.

→ How it works with a single predictor: Step:

① | x | y |
|---|---|
| x_1 | y_1 |
| ⋮ | ⋮ |
| x_n | y_n |

← 1° cutoff

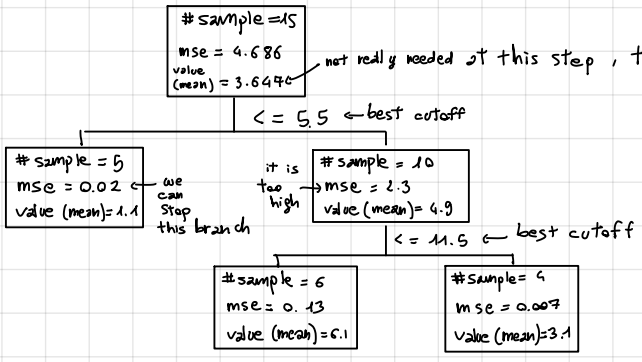
Form two groups creating a cutoff: start with the cutoff between x_1 and x_2 (the cutoff is $\leq x_1$, $> x_1$).

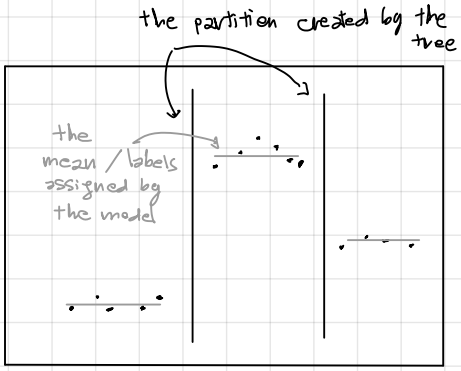
② Take the mean of each groups and assign the mean to each groups (the mean because minimizes $MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$)

③ Now we have labelled all the data with new labels and we compute the total MSE of the data.

④ Repeat this for all the $n-1$ possible cutoff and choose the cutoff that minimizes the total MSE.

⑤ Go down as long as MSE's are high (set a fixed MSE's benchmark). Ex: if the benchmark is 0.15 (≤ 0.15 you stop), these could be a tree for predicting a price car given the age:



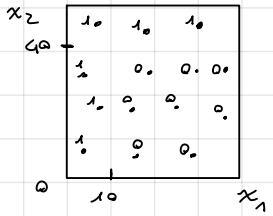


So the output is an indicator function $f(X) = \sum_{m=1}^{M} c_m \cdot \mathbb{1}_{\{X \in R_m\}}$ ← the mean of each group R_m

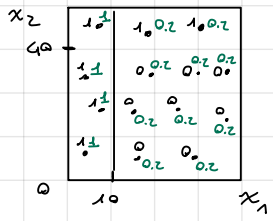
↑ can approximate any function

→ How it works with multiple predictors: for every terminal node we repeat points ① to ④ for all predictors and then we choose the couple (predictor, cutoff) that minimizes RMSE.

Example with figure: suppose we have a dataset where we want to predict YES/NO (1,0) with two parameters x_1, x_2. Suppose we have this data:

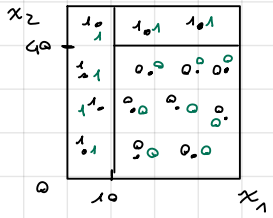


The first couple will be (x_1, <10) (we can see from the figure we make only two mistakes):

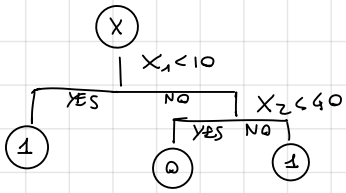


In green the values (mean) assigned at this step

The second step will be (x_2, <40) (no mistakes):



It's perfect!



→ Strenghts and weaknesses:

- When to stop? Each splits improves fit but we risk overfitting. So we can use stopping rule (as minimum number of obs. in each split; number of obs. in any terminal node; minimum fit improvement)
- Splitting is not robust (change few variable could change everything because the tree don't see in the future when is constructed)

→ we can try to fix this smaller tree (although will be worse in-sample fit)

we can put stopping rule more restrictive: the problem is that we stop at a not important split but the split after was important

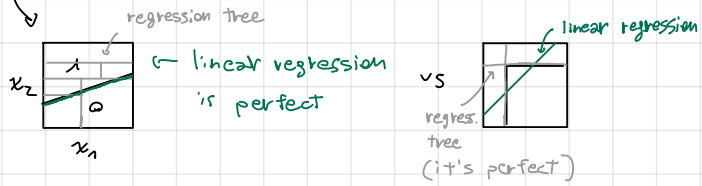
Pruning: grow a tree as large as possible and then cut back (usually better)

- Greedy algorithm: the best split is made at a particular step is a local optimum (not the best possible tree) so is myopic but very fast. Not so great as prediction method

- Variable importance: we don't have coefficient to estimate and a variable appear at many splits. So we can use this to understand the variable importance: measures how much fit improved when a given predictor is used for splitting and we sum these improvements across all splits that use this predictor. (We can also compare that number with a prediction that not use this variable).

→ Linear regression vs Regression tree:

Feature	Linear regression	Regression tree
• Solution method	• formula	• Algorith no formula
• Solution goal	• Minimize RSE	• Minimize RSE
• Solution optimality	• Best possible lin.regr. with the x variables	• Greedy algorithm: no find the best tree
• Variable selection	• Predefined list	• No predefined list
• Main results	• Coefficients and formula for prediction	• Terminal nodes with the tree for predic.
• Predicted $\hat{y}$	• $\hat{y}$ different for different x	• Assign same $\hat{y}$ to different x
• Linear relationships	• Captures perfectly	• Approximate by indicator
• Nonlinear relationships	• Need to specify functional form to approximate	• Approximate by indicator



Although pruning, overfitting is a problem in regression tree so:

Ensemble methods: introduce randomness into data: construct many imperfect models that together produce a better model

→ Bagging: Start from a dataset, split in training set and test set. Step:

① Bootstrapping: draw a random sample of data from the training set and we substitute that with random data from the dataset. Same sized sample, some observations included

multiple times, same eliminated.]

Do K bootstrapped samples

(2) Construct K trees, one for each bootstrapped sample with a pre-defined stopping rule. Run the model for each tree.

(3) Average the K predicted values and this mean will be the final prediction (can use cross validation at the end).

→ Random forest: same as bagging but at each node we run the algorithm choosing randomly m features (a subset of the initial features. That because if a predictor is very strong, most of bagging trees will start with the same feature ⇒ all the trees will be similar. So the strenghts are that is fast, few hyperparameters (T = number of trees; m = number of variables checked at each split, typically $\sqrt{p}$; depth of trees; often selected by cross validation). On the other hand is not interpretable.

→ Random forest vs bagging: bagging don't reduce so much the variance for the previous reason, random forest "decorrelate" the bagged trees.

→ Partial dependence plot (PDP): this plot tells us hat the value of the feature x_i influences the model prediction. Steps:

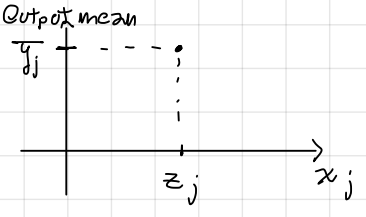
(1) Suppose to have $(y_1, x^1), \dots, (y_n, x^n)$ data points where $x^i = (x_1^i, \dots, x_p^i)$ is the features vector. Fix a feature j.

(2) Let feature j varying in an arbitrary range. Suppose the j-th feature is equal to z_j. We fix all the other variables and so we have n points

$\overline{x}^i = (x_1^i, \dots, x_{j-1}^i, z_j, x_{j+1}^i, \dots, x_p)$ $\forall i = 1, \dots, n$.

(3) Run the trained model with $\overline{x}^i$ and let $\overline{y}_j$ the mean of these n outputs.

A point of the plot will be $(z_j, \overline{y}_j)$



Remark: for linear regression the PDP of a variable x_i is a straight line with coefficient β_i.

→ **Boosting**: another ensemble method based on trees. Intuitively, improves upon regression trees but sequentially (each tree is built based on the mistakes made by the previous one). At the end boosting combines all the trees to make a prediction.

→ **Weak tree**: is a tree with 1 to 3 nodes. In contrast, a strong tree is a tree with a large number of nodes.
(↓ max depth is 1)

→ we have two methods based on boosting

→ **Adaboost**: train a set of weak learner trees iteratively on a weighted version of the data (data points with large prediction error have their weights increased to focus more on them).

→ **Gradient Boosting Machine (GBM)**: to understand how it works we do an example:

① We have the following dataset

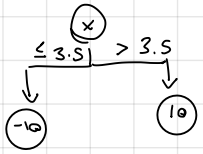
Feature	Target
2	10
3	20
4	30
5	40

② Start with a base prediction: $\hat{f}(x) = \bar{y}$ (mean)

and define $r = y - \bar{y}$ the residuals (base step)

Target	Pred.	Residuals
10	25	-15
20	25	-5
30	25	5
40	25	15

③ Let $\hat{f}^1(x)$ be a weak tree to the residuals (i.e. choose the local optimum node for minimize residuals):



Use it to update the initial prediction using the formula $\hat{f}(x) + \lambda \hat{f}^1(x)$:

Update model prediction $\left(\begin{pmatrix}2\\3\\4\\5\end{pmatrix}\right)$ $\underset{\lambda := 0,1}{=}$ $25 + 0,1\left(\begin{pmatrix}-10\\-10\\10\\10\end{pmatrix}\right) = \begin{pmatrix}24\\24\\26\\26\end{pmatrix}$

And for residuals $r - \lambda \hat{f}^1(x)$:

Update residuals $\left(\begin{pmatrix}-15\\-5\\5\\15\end{pmatrix}\right) = \begin{pmatrix}-15\\-5\\5\\15\end{pmatrix} - 0.1\begin{pmatrix}-10\\-10\\10\\10\end{pmatrix} = \begin{pmatrix}-14\\-4\\4\\14\end{pmatrix}$

④ Iterate, building a new $f^2(x)$ weak tree to locally optimize the new vector residuals.

⑤ Suppose we have obtained $\hat{f}^1(x), \dots \hat{f}^B(x)$ prediction, we take as final prediction:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda f^b(x)$$

(More robust in overfitting but more iterations).

Classification (Chapter 4)

→ Goal: predict a categorical output from a vetor of inputs (example: email spam or not spam or sentiment analysis: good or bad news for stock prices). Classification algoritms learn a classifier using training data, then used to predict category for new inputs.

→ 1-of-K Ecoding scheme: uses linear regression to do classification when you have a labelled dataset: ex. we have a dataset of vehicles with features (ex. weights, engine power ...) and each vehicles is labelled Car (G_1), Truck (G_2) or Motorcycle (G_3) (K = number of classe = 3). We define $Y_i(x) = \begin{cases} 1, & x \in G_i \\ 0, & x \notin G_i \end{cases}$ $\forall i = 1, \dots, K(=3)$. Then, fixed the category G_i, we have n vectors where n is the number of elements in the dataset:

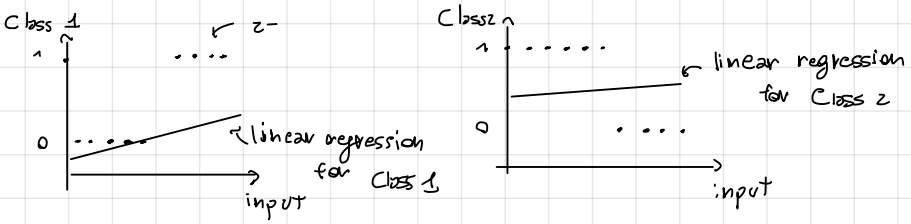
$$\{(\text{features}(x), Y_i(x))\}_{x \in \text{Dataset}}.$$

So I can do a linear regression problem on this dataset obtaing a linear regression model for each category G_i. So, for a new input we say that.

$$\hat{G}(\bar{x}) := \underset{G_i}{\operatorname{argmax}} \hat{f}_i(\bar{x})$$

($\hat{G}(\bar{x})$: the category of $\bar{x}$; $\hat{f}_i(\bar{x})$: linear regression model for G_i.)

- Problems: a class can be masked: 2 classes



every data is classificnd in Class 2.

→ Logistic regression: we use it when we have only two possible outputs (binary outputs).

Given feature input x and a vetor of parameters $w = (1, w_1, \dots, w_n)$ we define:

$$P(y=1 \mid x, w) := \frac{\exp(w^T x)}{1+\exp(w^T x)} \quad \text{and} \quad P(y=0 \mid x, w) := 1 - P(y=1 \mid x, w) = \frac{1}{1+\exp(w^T x)}$$

So, the question is: how we find w?

We note that if we define $\text{Odds} := \frac{p}{1-p}$ and $\text{Logit} := \log\left(\frac{p}{1-p}\right)$ we have

$\text{Logit}\left(P(y=1 \mid x, w)\right) = w^T x$ so this quantity is linear in the features $\Rightarrow$ we would like to use least-squares but the log transforms the values in something very negative or positive $\Rightarrow$ we have the problem of outliers. So we can use MLE (maximum likelihood estimation) : we have a dataset of N independently distributed points, the likelihood of w is: $L(w) = \prod_{i=1}^{N} p_i(w)^{y_i} \left(1 - p_i(w)\right)^{1-y_i}$ (y_i: label of x_i)

where $p_i(w) = P(y_i = 1 \mid x_i, w)$.

Now we use log to have a sum:

$$l(w) := \sum_{i=1}^{N} y_i \log\left(p_i(w)\right) + (1-y_i)\left(\log\left(1 - p_i(w)\right)\right) =$$

$$= \sum_{i=1}^{N} y_i \log\left(p_i(w)\right) + \log\left(1 - p_i(w)\right) - y_i \log\left(1 - p_i(w)\right) =$$

$$= \sum_{i=1}^{N} y_i \log\left(\frac{p_i(w)}{1 - p_i(w)}\right) + \log\left(1 - p_i(w)\right) =$$

$$= \sum_{i=1}^{N} y_i w^T x_i - \log\left(1 + \exp(w^T x_i)\right)$$

So, maximizing this function:

$$\nabla_w l(w) = \sum_{i=1}^{N} \left(y_i x_i - \frac{\exp(w^T x_i)}{1 + \exp(w^T x_i)} \cdot x_i \right) = \sum_{i=1}^{N} x_i \left(y_i - p_i(w) \right) = 0$$

(vectore eq.)

$\Rightarrow n+1$ non linear equations in w (n: number of features) (can be solved numerically with Iteratively Reweighted Least Squares (IRLS).) Advantages: manages outliers.

(Multinomial logistic regression : $K > 2$, $G_1, \dots, G_K$ classes, now we have:

$$P(G_K \mid x, w) = \frac{\exp(w^T x)}{\sum_j \exp(w_j^T x)}$$

, use MLE to estimate w

→ Probit regression : the same as logistic but $P(y=1 \mid x, w) = \Phi(w^T x)$ where $\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$ (cumulative distribution function of $N(0,1)$). (Multinomial probit as well)

Generative vs discriminative algorithms : the discriminative algorithms directly model the decision boundary capturing the differences between classes based on the dataset. (It's like a child who learns only the differences between things and recognizes them by these differences). On the other hand, generative algorithms learn model the joint distribution of the

data (it's like a child who memorizes things not by their differences but in depth).

It is not always true that the second one is better.

Examples of algorithms

Discriminative: Least squares, regression based classifiers, logistic regression, Bayesian logistic regression, probit, classification trees, K-nearest neighbors, support vector machine, (deep) neural networks.

Generative: LDA, QDA, Naive Bayes, Bayesian network, Markov random fields, Hidden Markov models.

Generative classification algorithms: aim: we model the joint distribution $P(X,G) \overset{\text{Bayes}}{=} P(X|G)P(G)$

It makes sense that a predictor $\tilde{G}$ wants:

$$\tilde{G}(X) = \underset{G \in \mathcal{G}}{\operatorname{argmax}}\ \hat{P}(G|X)$$

, so pointwise for a point $x \in X$

$\mathcal{G}$: possible classes; $\hat{P}$: the probability we have from the data

$$\boxed{\tilde{G}(x) = \underset{G \in \mathcal{G}}{\operatorname{argmax}}\ \hat{P}(G \mid X = x)}$$

For Bayes: $\tilde{G}(X) = \underset{G \in \mathcal{G}}{\operatorname{argmax}}\ \tilde{P}(G|X) = \underset{G \in \mathcal{G}}{\operatorname{argmax}}\ \frac{\tilde{P}(X|G)\hat{P}(G)}{P(X)} = \underset{G \in \mathcal{G}}{\operatorname{argmax}}\ \tilde{P}(X|G)\tilde{P}(G)$

($P(X)$ does not depend on G)

Another point of view

EPE (expected prediction error): suppose we have a loss function $L(G, \tilde{G}(X))$ that take a couple $(G_K, G) \xrightarrow{L}$ (loss of the event "the true label is G_K and the model predicted G"). We define:

$$EPE := \mathbb{E}_{X,G}\left[L(G, \tilde{G}(X))\right] \overset{\text{def. of conditional density } f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}}{=} \mathbb{E}_X\left[\mathbb{E}_G\left[L(G, \tilde{G}(X)) \mid X\right]\right] \overset{\text{def. cond. expec.}}{=}$$

$$= \mathbb{E}_X\left[\sum_{K=1}^{K} L(G_K, \tilde{G}(X)) P(G_K \mid X)\right]$$

If we want to minimize the EPE we can minimize it pointwise as function of x, defining a new classifier:

$$\tilde{G}(x) = \underset{G \in \mathcal{G}}{\operatorname{argmin}} \sum_{K=1}^{K} \underbrace{L(G_K, G)}_{\text{does not depend on } x!} P(G_K \mid X = x)$$

In case we have as $L(G, \hat{G}(X))$ the 0-1 loss function ($L(G_K, G)$ = 1 if $G \neq G_K$, 0 if $G = G_K$ (the pred. is correct, the predic. is true)) we have that:

$$\hat{G}(x) = \underset{G \in \mathcal{G}}{\operatorname{argmin}} \sum_{k=1}^{K} L(G_k, G)\, P(G_k \mid X = x) =$$

$$= \underset{G \in \mathcal{G}}{\operatorname{argmin}} \sum_{G_k \neq G} P(G_k \mid X = x) \overset{\leftarrow}{=} \sum_{k=1}^{K} P(G_k \mid X = x) = 1$$

$$= \underset{G \in \mathcal{G}}{\operatorname{argmin}} \left(1 - P(G \mid X = x)\right) = \underset{G \in \mathcal{G}}{\operatorname{argmax}} P(G \mid X = x)$$ so we have found the same classifier

Also called Maximum a Posteriori (MAP) classifier

→ Linear Discriminant Analysis (LDA): generative model which assumes that data are normally distributed within each class. So the classifier is:

$$\hat{G}(x) = \underset{G \in \mathcal{G}}{\operatorname{argmax}} \hat{P}(x \mid g = k)\, \hat{P}(G = k) = \underset{G \in \mathcal{G}}{\operatorname{argmax}} \hat{f}_k(x)\, \hat{P}(G = k)$$

where: • $\hat{f}_k(x) = \frac{1}{(2\pi)^{p/2} |\hat{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \hat{\mu}_k)^T \hat{\Sigma}^{-1} (x - \hat{\mu}_k)\right)$ with

($|\hat{\Sigma}|$: determinant; $\hat{\Sigma}$: it's the same $\forall k$)

$\hat{\mu}_k := \frac{1}{N_k} \sum_{x_i \in G_k} x_i$ ($N_k = \#G_k$, x_i elements in G_k) (centroid of class k)

$\hat{\Sigma} := \frac{1}{N - K} \sum_{k=1}^{K} \sum_{x_i \in G_k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$ (covariance matrix) ↓ obtained thanks to MLE

• $\hat{P}(G = k) = \frac{N_k}{N}$

→ Linear discriminant function: note that the log-odds are linear in x:

$$\log\left(\frac{\hat{P}(G = k \mid X = x)}{\hat{P}(G = I \mid X = x)}\right) = \log\left(\frac{\hat{P}(G = k) \cancel{\frac{1}{(2\pi)^{p/2}}} |\hat{\Sigma}|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \hat{\mu}_k)^T \hat{\Sigma}^{-1} (x - \hat{\mu}_k)\right)}{\hat{P}(G = I) \cancel{\frac{1}{(2\pi)^{p/2}}} |\hat{\Sigma}|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \hat{\mu}_I)^T \hat{\Sigma}^{-1} (x - \hat{\mu}_I)\right)}\right) =$$

$$-\tfrac{1}{2}(a-b)^T M (a-b) - \left(-\tfrac{1}{2}\right)(a-c)^T M (a-c) = \cancel{-\tfrac{1}{2} a^T M a} + \tfrac{1}{2} a^T M b + \tfrac{1}{2} b^T M a - \tfrac{1}{2} b^T M b + \cancel{\tfrac{1}{2} a^T M a} - \tfrac{1}{2} a^T M c - \tfrac{1}{2} c^T M a + \tfrac{1}{2} c^T M c$$

$$= a^T M b - \tfrac{1}{2} b^T M b - a^T M c + \tfrac{1}{2} c^T M c$$

$$= \left(\log \hat{P}(g = k) + x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \tfrac{1}{2} \hat{\mu}_k^T M \mu_k\right) - \left(\log \hat{P}(g = I) + x^T \hat{\Sigma}^{-1} \hat{\mu}_I - \tfrac{1}{2} \hat{\mu}_I^T \hat{\Sigma}^{-1} \hat{\mu}_I\right)$$

$$= \delta_k(x) - \delta_I(x)$$

where $\boxed{\delta_k(x) := \log \hat{P}(g = k) + x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k}$ is a linear function in x with intercept $\log \hat{P}(g = k) - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k$ and $x^T \hat{\Sigma}^{-1} \hat{\mu}_k = \hat{\mu}_k^T \hat{\Sigma}^{-T} x =$

$= \hat{\mu}_k^T \hat{\Sigma}^{-1} x$ ($\hat{\Sigma}$ symm.). $\delta_k(x)$ is called linear discriminant functions.

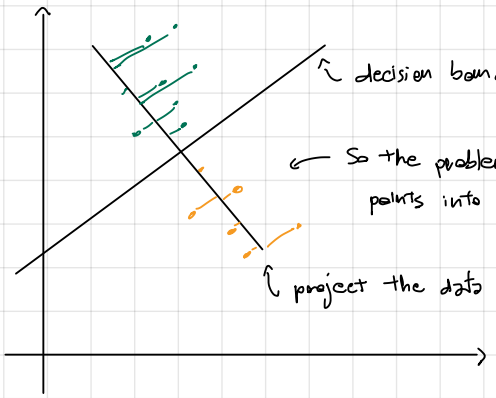
Since the log is an increasing function (and Σ is the same so thanks (*) doesn't change argmax)

$$\hat{G}(x) = \arg\max_K \delta_K(x) \qquad (*)\ \frac{1}{(2\pi)^{n/2}|\Sigma|^{\frac{1}{2}}}$$

Decision boundary: fix two classes K, I. The decision boundary is the set of all x s.t. $\delta_K(x) = \delta_I(x)$. Note that is an affine space: $\delta_K(\lambda x + \mu y) = \lambda \delta_K(x) + \mu \delta_K(y) =$ (δ_K linear)

$= \lambda \delta_I(x) + \mu \delta_I(y) = \delta_I(\lambda x + \mu y)$.

Geometric interpretation: two classes:



Advantages: • simple: for classify a new point x_0, compute $\delta_K(x_0)$ and take the max.

Disadvantages

- log-odds are linear as in logistic regression
- logistic is somewhat more general (no assumption about $P(X)$ in logistic)
- In practice very similar results

→ LDA example: classification rule for default (D) / non-default (ND) on the basis of credit card balance (i) and (ii) student status. Dataset of 10.000 training samples and the overall default rate is 3,33 % $\left(\frac{3,33}{100} \cdot 10'000 = 333 \text{ default}\right)$.

Two types of error: False positives / False negatives

Confusion matrix:

		True status D	True status ND
Pred. status	D	81	23
	ND	252	9644

Accuracy = % right pred. = $\frac{9644+81}{10000} = 97,25\%$ Error rate LDA = $100\% - 97,25\% = 2,75\%$

Precision = $\% \left(\frac{\text{True positive}}{\text{True positive + False positive}}\right) = \% \left(\frac{81}{81+23}\right) = 77,9\ \%$

Recall = $\% \left(\frac{\text{True positive}}{\text{true positive + false negative}}\right) = \% \left(\frac{81}{81+252}\right) = 25\ \%$ ← very bad

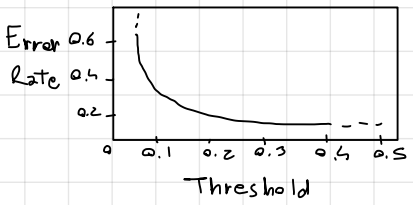
To adjust the recall we can adjust the threshold: for example we say to the model that if the posterior default probability $(P(\text{Default} \mid X = x))$ exceeds 20% it has to predict default. The confusion matrix becomes:

		True status D	True status ND
Pred. Status	D	195	235
	ND	138	9432

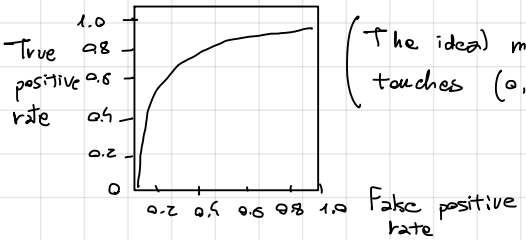
$$\text{Accuracy} = \frac{195 + 9432}{10000} = 96,27\ \% \qquad \text{Precision} = \frac{195}{195 + 235} \simeq 45,3\ \%$$

$$\text{Recall} = \frac{195}{195 + 138} \simeq 58,6\ \%$$

. So varying the threshold we will have:



ROC Curve : a point in the curve is obtained fixing a threshold and finding the TP rate and FP rate



→ Naive Bayes : LDA is incovenient with many predictors.

Naive Bayes assumes: $P(X \mid G) = \prod_{j=1}^{p} P(X_j \mid G)$, i.e. the features conditional on G are indipendent (very strong and unrealistic but it works, for example sentiment analysis).

The predictor is:

$$\hat{G}(X) = \underset{G}{\operatorname{argmax}}\ \hat{P}(G) \prod_{j=1}^{p} \hat{P}(X_j \mid G)$$

where $\hat{P}(g)$ e $\hat{P}(X_j \mid g)$ are estimated via MLE.

Remarks: • If Z contains the indipendent features (conditional on G) and we assumed that are Gaussian, assume $Z \sim N(\mu, \Sigma)$ with μ mean vector and Σ covariance matrix that is diagonal. So we can apply both LDA and Naive Bayes.

• If Z contains features that are indipendent on G but $Z \sim N(\underbrace{\Sigma^{-\frac{1}{2}}}_{\text{cov. mat.}} \underbrace{\mu}_{\text{mean vector}}, I)$ I can apply Naive Bayes but not LDA because $\Sigma^{-\frac{1}{2}}\mu$ is not the mean vector (μ is the mean vetor). But I can apply LDA to $X := \underbrace{\Sigma^{\frac{1}{2}}}_{\text{linear trasformation}} Z \sim N(\mu, \Sigma)$

- Suppose the class conditional probabilities are Gaussian with same covariance Σ so

$$\hat{P}(X_j \mid G) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_K)^T \Sigma^{-1} (x-\mu_K)\right)$$ as in LDA $\Rightarrow$ we know

that the decision boundary is a linear space (it means that in this case Naive Bayes is a linear classifier). Furthermore the region of probability $\geq K\%$ are ellipsoid (I find at least with this prob.) full-dimensional. Indeed:

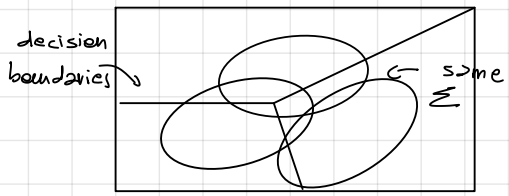
$$\hat{P}(X_j \mid G) \geq K \quad \text{(fixed percentage)}$$

$$\frac{1}{(2\pi)^{d/2} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_K)^T \Sigma^{-1} (x-\mu_K)\right) \geq K$$

$$\log\left(\frac{1}{(2\pi)^{d/2} |\Sigma|^{\frac{1}{2}}}\right) - \frac{1}{2}(x-\mu_K)^T \Sigma^{-1} (x-\mu_K) \geq K$$

$$\underbrace{(x-\mu_K)^T \Sigma^{-1} (x-\mu_K)}_{\text{"ellipsoid"}} \leq \underbrace{2\log\left(\frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}}\right) - 2K}_{\text{constant}}$$

Note that, here, Σ can be change varying K, we will obtain different ellipsoids (when Σ is the same only the center changes, if Σ is different the "shape" changes as well).



- In general, the naive Bayes boundary can be of higher degree, for example quadratic (for example a paraboloid)

→ Quadratic discriminant Analysis (QDA): is LDA where we drop the hyp. that Σ is equal $\forall K$. So $P(X_K \mid G) = \frac{1}{(2\pi)^{M/2} |\hat{\Sigma}_K|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\hat{\mu}_K)^T \hat{\Sigma}_K^{-1} (x-\hat{\mu}_K)\right)$.

Using log:

$$\log\left(P(X_K \mid G)\right) = -\frac{M}{2}\log(2\pi) - \frac{1}{2}\log|\hat{\Sigma}_K| - \frac{1}{2}(x-\hat{\mu}_K)^T \hat{\Sigma}_K^{-1} (x-\hat{\mu}_K)$$

So:

$$\hat{G}(x) = \underset{G \in \mathcal{G}}{\arg\max}\ \hat{P}(x \mid G=K)\ \hat{P}(G=K) = \underset{K}{\arg\max} -\frac{1}{2}\log|\hat{\Sigma}_K| - \frac{1}{2}(x-\hat{\mu}_K)^T \hat{\Sigma}_K (x-\hat{\mu}_K) + \log \hat{P}(G=K)$$

↑ log increasing and $-\frac{M}{2}\log(2\pi)$ doesn't change argmax

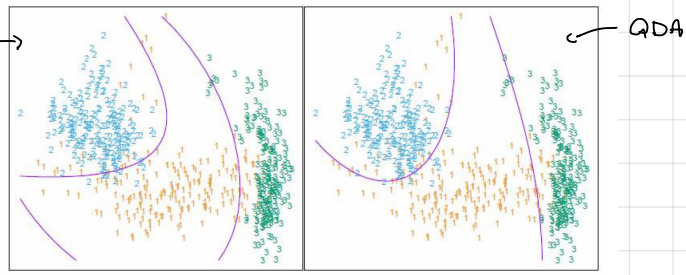
So, setting the quadratic discriminant function

$$\delta_k(x) := -\frac{1}{2}\log|\hat{\Sigma}_k| - \frac{1}{2}(x-\hat{\mu}_k)^T\hat{\Sigma}_k(x-\hat{\mu}_k) + \log\hat{P}(G=k)$$

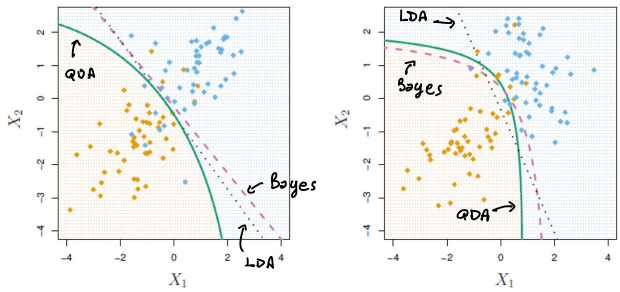
we have

$$\hat{G}(x) = \arg\max_k \delta_k(x)$$

→ LDA vs QDA : we have two methods for fitting quadratic boundaries: LDA using linear and quadratic features (ex. $\{X_1, X_2, X_1^2, X_2^2\}$) or QDA directly.



→ LDA vs QDA vs Bayes : 2 features $\{X_1, X_2\}$. So LDA is linear, QDA quadratic, Naive Bayes linear in the left picture (the covariance matrix $\Sigma_1 = \Sigma_2$) and quadratic in the right picture ($\Sigma_1 \neq \Sigma_2$):



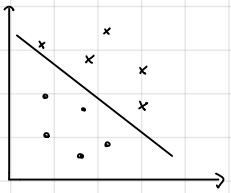
K nearest neighbors (discriminative) : the idea is that given a new data point x, that needs to be classified, we find the K nearest neighbors of x and use the majority of those neighbors to classify x. We need a metric to measure distance between data points (generally a good idea standardize features so that $\mu = 0$ and $\sigma = 1$). Need also a rule to choose the right K. Can also be used for regression. Slow with large dataset and works better with low dimensional feat.

↓

Kernel classification : K-nearest gives equal weight to all K neighbors. It makes more sense to give more weight to the closest neighbors. So every training points gets a vote $K(x, x_i)$ when classifying a new point x.

For example a Gaussian Kernel : $K(x, x_i) = e^{-d(x,x_i)/\sigma^2}$.

Support vector Machines (SVM) : start supposing we have a training data $(x_1, \dots, x_n)$ with binary targets $y_i \in \{-1, 1\}$. Furthermore, suppose it is a separable linear case (i.e. the data can be divided perfectly by an hyperplane :

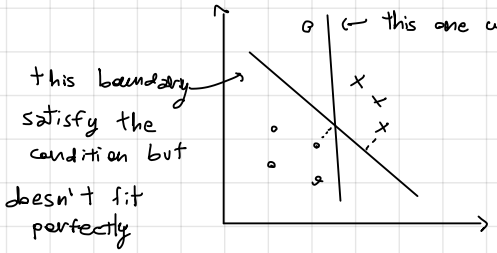


We would like to have a decision boundary to be as far as possible from training data (that works) so the prediction is "probably" correct (does not compute a probability like logistic)

So, suppose the classification rule is:

$$h_{w,b}(x) = \text{sign}(w^T x + b)$$, the decision boundary is $w^T x + b = 0$.

So, if the two closest training data are labelled with a "+1" and a "-1" this is a necessary condition to have a robust model but is not sufficient to fit perfectly



→ Functional margin : given a training sample (x_i, y_i), the functional margin of (w, b) is

$$\hat{\gamma}_i = y_i (w^T x_i + b)$$

So if $y_i = 1$, if $w^T x_i + b$ is large positive the prediction is right and confident (the boundary is $w^T x + b = 0$) ; if $y_i = -1$, if $w^T x_i + b$ is large negative the prediction is right and confident. So, in general if $\hat{\gamma}_i$ is large and positive $(+ \cdot + = +, - \cdot - = +)$ the prediction is right and confident.

The functional margin of a set $\{(x_i, y_i)\}_{i=1}^n$ is $\boxed{\hat{\gamma} = \min_{i=1,\dots,n} \hat{\gamma}_i}$

Problem: $w^T x + b = 0 \Leftrightarrow (Kw)^T x + Kb = 0$ with K large as I want, so I can choose a boundary s.t. $\hat{\gamma}_i$ is large as I want without changing the robustness of the model.

↓

Geometric margin: the geometric margin of a point is:

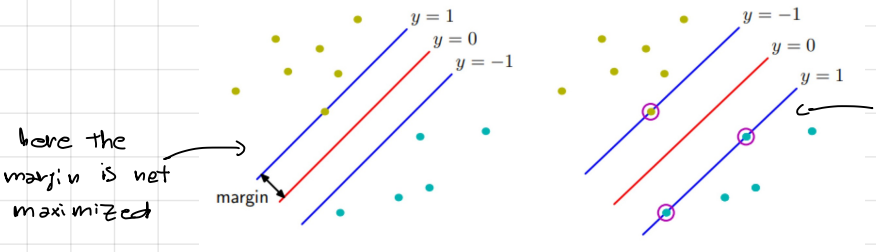
$$\boxed{\gamma_i = y_i \left(\frac{w^T}{\|w\|_2} x_i + \frac{b}{\|w\|_2} \right)}$$

(is the distance between (x_i, y_i) and its projection on the decision boundary)

Remark: for $\|w\|_2 = 1$, Geometric margin = functional margin

As before, the geometric margin of a set (x_i, y_i) is $\boxed{\gamma = \min_i \gamma_i}$

→ : the intuition of SVM:



here the margin is not maximized

here is maximized (found by SVM): the location of the boundary is determined by a subset of the data points called support vectors (circled)

So the problem of maximizing the margin is:

$$\max_{w,b} \gamma \quad \text{s.t.} \quad \underbrace{y_i (w^T x_i + b) \geq \gamma}_{\text{prediction is exact and confident}} \quad \forall i, \ \|w\| = 1$$

← is not a convex constraint!

It is equivalent to $\max_{w,b} \frac{\hat{\gamma}}{\|w\|}$ s.t. $y_i (w^T x_i + b) \geq \hat{\gamma} \quad \forall i$ ← not convex but now we can scale w and b arbitrarily

Now, so if I scale of a factor $\frac{1}{\|\hat{\gamma}\|}$ the problem we obtain:

$$\max_{w,b} \frac{1}{\|w\|} \quad \text{s.t.} \quad y_i (w^T x_i + b) \geq 1 \quad \forall i$$

$$\Updownarrow$$

$$\boxed{\min_{w,b} \frac{1}{2} w^T w \quad \text{s.t.} \quad y_i (w^T x_i + b) \geq 1 \quad \forall i}$$

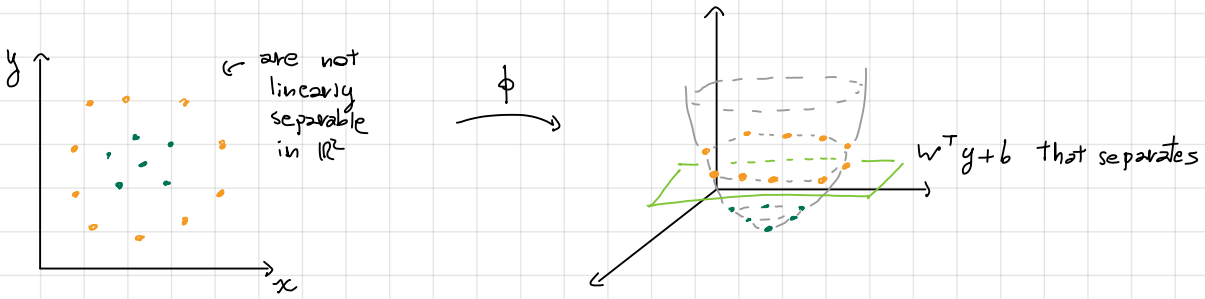
$\frac{1}{2}$: included for convenience in computation

Convex and quadratic with linear constraint ⇒ Easy to solve

Ensure all training points are correctly classified in the separable case.

→ **The Kernel trick** : increase the dimension of the problem to solve it more easily:

- Transform $x \in \mathbb{R}^m$ into $\phi(x) \in \mathbb{R}^M$ $M > n$;
- If the data are linearly separable in $\mathbb{R}^M$, solve the problem as before finding w, b for the boundary.
- The classification rule will be: $h_{w,b}(x) = \text{sign}\left(w^T \phi(x) + b\right)$



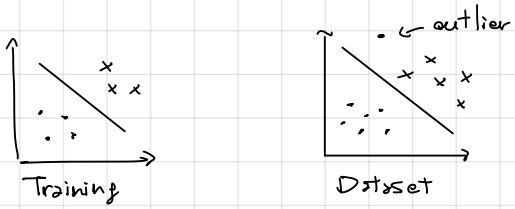
Example of Kernel : • Linear $K(x,y) = x^T y$ (K represents an inner product in $\mathbb{R}^n$, i.e. $K(x,y) = \phi(x)^T \phi(y)$, see Kernel trick in Clustering Chap. 6)

• Polynomial $K(x,y) = \left(\gamma\, x^T y + r\right)^d$ • Gaussian $K(x,y) = e^{-\gamma \|x-y\|^2}$

• Sigmoid $K(x,y) = \left(\gamma\, x^T y + r\right)$

Remark : • mapping in a higher dimension via ϕ doesn't guarantee that data become separable.

• Previous algorithm is sensitive to outliers: I can find a very robust boundary in the training set but if in the test set there is an outlier, can be a problem:



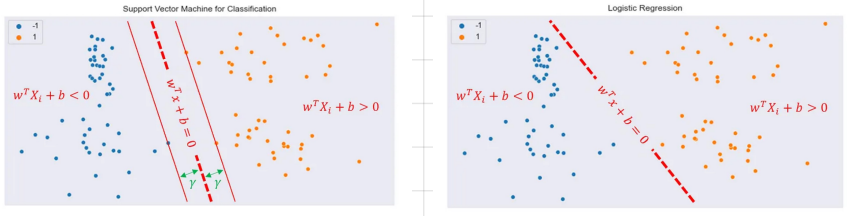
↓

Regularization : to let the model be more flexible we can use L^1-regularization:

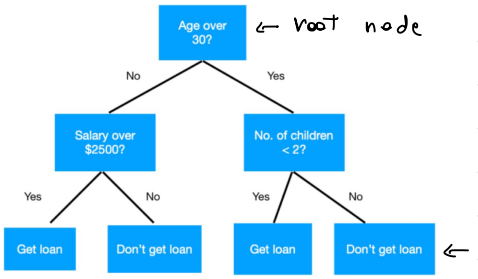
$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \quad \text{s.t. } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \;\; \forall i, \;\; \xi_i \geq 0 \;\; \forall i$$

Now the function margin can be less then 1 at a cost in classification of $C\xi_i$.

→ **SVM vs Logistic regression** :



Decision trees :



← root node

← leaf nodes

(recall that for regression trees we wanted to minimize MSE). Let's start with a definition:

→ Entropy : measure homogeneity in a set : $E = -\sum_{i=1}^{C} p_i \log_2(p_i)$

where C is the number of classes , $p_i = \frac{N_i}{N}$ (N_i = # class i , N = # set)

Now , using an example , we understand how it works a decision tree.

Dataset : Feature 1 : Balance (↙ <50K ↘ >50K) Feature 2 : Residence (↙ own ↓ rent ↘ other) Target : value of loan will increase or not (I, NI)

There 16 I and 14 NI (30 = # dataset) ;

Balance : there are 1 I and 12 NI with Balance <50K , 4 I and 13 NI in Balance >50K

Residence : there are 7 I and 1 NI with Residence "own" , 4 I and 6 NI in Residence "Rent" ,

5 I and 7 NI in Residence "other"

1 Step : $E(\text{Parent node} = \text{all the dataset}) = -\frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{16}{30}\right) \simeq 0.99$

↑ in this case is the first step

2 Step : If I choose Balance I have two "children" : "<50K" , ">50K" so I compute the "conditional entropy" :

$E[\text{Parent node} \mid \text{Child 1}] = E[\text{All dataset} \mid \text{Balance} < 50K]$ (1° step ↙ , ← 1 I and 12 NI in Balance <50K)

$= -\frac{1}{13}\log_2\left(\frac{1}{13}\right) - \frac{12}{13}\log_2\left(\frac{12}{13}\right) \simeq 0,39$

$E[\text{Parent node} \mid \text{Child 2}] = E[\text{All dataset} \mid \text{Balance} > 50K] = -\frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \simeq 0,79$

I do the weighted average :

$E[\text{All dataset} \mid \text{Balance}] = \frac{13}{30}\,0,39 + \frac{17}{30}\,0,79 = 0,62$

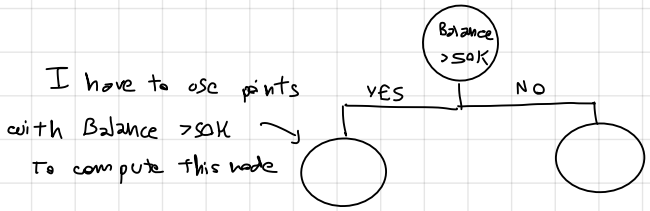
The information gain is $IG(\text{All dataset}, \text{Balance}) = E(\text{All dataset}) - E(\text{All dataset} \mid \text{Balance}) = 0,37.$

I do the same for "Residence" (there are three children) and we find that

$IG(\text{Alldataset}, \text{Residence}) = 0,13$.

So the first split is between Balance $\gtrless$ 50K because has the largest IG.

3 step: When we have lots of features we compute the same as above but not using all the dataset but only the dataset points that are in the node:



Remark: • So this method reduces the entropy as much as possible at each level

• We can also use Gini index ($G = \sum_{i=1}^{C} p_i(1-p_i)$ instead of E, it's faster because there is no log).

→ When to stop? • All the nodes have entropy 0 (pure nodes)

• The tree is higher than a threhold

• All the attributes have been used

• There is risk of overfitting → we can pruning

Random forest: similar to random forests for regression: introduce randomness into data, fit n decision trees and take the majority vote (ensembling). Two level of randomness:

• draws a random sample of data with replacement from the training dataset (bootstrapping/bagging)

• randomly select a subset of features which can be used as candidates at each split

→ Boosting: same as in random forest for regression. → Gradient boosting machine: the same as in regression

→ Adaboost: Step 1: each point gets a weight $w_i = \frac{1}{n}$

↓ Powerful and easy but sensitive to noise

Step 2: train a weak learner: Error rate = $\sum_{\text{missclassified points}} w_i \Big/ \sum_{\text{all points}} w_i$

Learners weight = $\ln\left(\frac{1 - \text{Error rate}}{\text{Error rate}}\right)$

Step 3: update the weights of the training data and iterate until the error rate does no longer decrease

Step 4: combine the weak learners prediction doing a weighted mean

Introduction to Natural Language Processing (NLP) (Chapter 5)

Text preprocessing

→ **Build Vocabulary**: breaks natural language text into chunks called **tokens**

↳ Separate punctuations, case normalization (Investors ~ investors), split contractios, eliminate stop words (a, an, ...), casual text, identify tokens: words and n-grams

Stemming (puts together investors, invest but also better, bet so its better

Lemmatization that does not compress as much as stemming)

→ **One-hot vectors**: Sentence: Elon Musk made a bid Tokens: "Elon" "Musk" "made" "a" "bid"

Lexicon / Vocabulary: "a" "bid" "Elon" "made" "Musk" so "a" ~ e_1, "bid" ~ e_2, ..., "Musk" ~ e_5

and so we have the matrix for the sentence: $\begin{pmatrix} e_3^T \\ e_5^T \\ e_4^T \\ e_1^T \\ e_2^T \end{pmatrix}$. So I can sum the rows and obtain a row-vector with the i-th position representing the frequency of i-th words in the vocabulary, this is called **bag of word vector**.

↓

Cosine similarity: to say how similar are two documents I define:

it's the cos θ! ← $\underset{[0,1]}{\text{similarity}} := \dfrac{A \cdot B}{|A|\,|B|}$ where A, B the two hot vectors. Note we have to use the same vocabulary (so there could be also some zeros). And so

if Similarity = 0 ⇒ they're orthogonal and so they share no word

→ **TD-IDF**: we want to describe the importance of words in a document relative to the rest of a document. We have $\mathcal{D} = \{D_i\}_{i=1}^m$ a set of documents and $W = \{w_j\}_{j=1}^n$ the set of all words. We define:

Document frequency of j-th word = $DF_j = \dfrac{|\{i \in \{1,\dots,m\} : w_j \in D_i\}|}{|\mathcal{D}|}$

Inverse document frequency of j-th word = $IDF_j = \log\left(\dfrac{1}{DF_j}\right)$

Term frequency of j-th word in D_j = $(TF)_{ij} = \dfrac{n_{ij}}{|D_i|}$ where n_{ij} is how many times w_j appears in D_i

$(TF\text{-}IDF)_{ij} = TF_{ij} \times IDF_j$

Classification : it's useful to do text classification (ex. spam or not spam) and sentiment analysis (positive, neutral, negative tonality)

Method
- → rule based algorithm created by human (unsupervised learning) (1)
- → Supervised learning (we need labelled data) : train a model (2)
- → Semisupervised learning (few labels) (3)

(2): • **Naive Bayes approach** : we have a set of documents labelled (ex. positive, neutral, negative). We have a lexicon = $\{ w_j : w_j \text{ is a word in the set of documents} \}$. Also, C_K are the possible output (positive, neutral, negative). So a tone of a vector of words $(w_1, \dots, w_n)$ is defined:

$$G(w_1, \dots, w_n) := \underset{C_K}{\operatorname{argmax}} \; P(C_K \mid (w_1, \dots, w_n)) \quad \text{(it makes sense)}$$

Now from Bayes Theorem we know:

$$P(C_K \mid (w_1, \dots, w_n)) = \frac{P(w_1, \dots, w_n, C_K)}{P(w_1, \dots, w_n)} = \frac{P(w_1 \mid w_2, \dots, w_n, C_K) \cdot P(w_2, \dots, w_n, C_K)}{P(w_1, \dots, w_n)}$$

we omit $P(w_1, \dots, w_n)$ since ↓ is constant for G and the argmax doesn't change

$$\propto P(w_1 \mid w_2, \dots, w_n, C_K) \cdot P(w_2 \mid w_3, \dots, w_n, C_K) \cdot \dots \cdot P(w_{n-1} \mid w_n, C_K) \cdot P(x_n \mid C_K) \cdot P(C_K) =$$

$$= P(C_K) \prod_{i=1}^{n} P(w_i \mid C_K)$$

↑ Naive independence assumption: $P(x_i \mid x_{i+1}, \dots, x_n, C_K) = P(x_i \mid C_K)$

So $$G(w_1, \dots, w_n) = \underset{C_K}{\operatorname{argmax}} \; \underbrace{P(C_K)}_{=: \text{prior prob.}} \underbrace{\prod_{i=1}^{n} P(w_i \mid C_K)}_{=: \text{likelyhood}}$$

Now, $P(C_K)$ is obviously how many documents with label "C_K" there are in the training set divided by the total number of documents.

On the other hand, $P(w_i \mid C_K)$ is obtained concatanating all the documents with labelled C_K in the training set and computing the relative frequency for w_i.

If $P(w_i \mid C_K) = 0$ we can just add a constant "a" to the formula and so:

$$P(w_i \mid C_K) = \frac{\text{count}\left(w_i \in \underset{D \in C_K}{\Vert} D\right) + a}{\sum_{D \in C_K} |D| + a}$$

← "concatanating" notation

(3): We have few label data. We can do supervised learning on this dataset and obtain a model. We use this model on the dataset not labelled, we choose documents

that have been labelled well according to some confidence estimators and we add these documents to the initial labelled dataset. We iterate.

↗ for (2) (and (3))

Evaluate Performance when we have a labelled dataset and we want to see how our model works we use these:

$$\text{Accuracy} := \frac{\#\text{ texts well predicted}}{\#\text{ texts}}$$

$$\text{Recall}(C_K) = \frac{\#\text{ texts well predicted in } C_K}{\#\text{ texts well predicted in } C_K + \#\text{ texts predicted in } C_K \text{ but that are not in } C_K}$$

$$\text{Precision}(C_K) = \frac{\#\text{ texts well predicted in } C_K}{\#\text{ texts well predicted in } C_K + \#\text{ texts not predicted in } C_K \text{ but that are actually in } C_K}$$

$$F_1(C_K) = \frac{2}{\frac{1}{\text{Precision}(C_K)} + \frac{1}{\text{Recall}(C_K)}}$$

(1) Topic analysis : unsupervised NL learning technique

→ Linear Discriminant analysis : maps each document into a topic (topics are ex ante specified). It works with labelled data (it is considered supervised indeed). We consider the TF-IDF (*) vector x for each document and:

$$G(x) = \arg\max_K \; P(C_K)\, f_K(x) \qquad \text{where}$$

$P(C_K)$: prior prob. (computed as in Naive Approach); $f_K(x)$: likelihood

$$f_K(x) = P(x \mid G = K) \overset{\text{in this model}}{=} \frac{1}{(2\pi)^{M/2}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_K)^T \Sigma^{-1} (x-\mu_K)\right)$$

where : • M is the dimension of TF-IDF vectors ((*) we fix a document and let's vary the words, so the dimension of these vectors is the number of words in our vocabulary).

• μ_K is the mean vector defined as $\mu_K = \frac{1}{N_K} \sum_i x_i$ where N_K is the number document in class K and x_i is the TF-IDF vector of document in K.

- Σ is the "pooled covariance matrix": $\Sigma = \frac{1}{N-K} \sum_{j=1}^{K} \sum_{i \in j} (x_i - \mu_j)(x_i - \mu_j)^T$
  - N: total number of doc.
  - K: number of classes
  - j: each classes
  - $i \in j$: documents in class j

→ Latent semantic Analysis: apply PCA/SVD to the covariance matrix of TF-IDf (so we compress the information contained in TF-IDF into a smaller matrix). Issue: interpretability because of the abstract nature of mathematical transformations. (It is unsupervised)

→ Latent Dirichlet Allocation: unsupervised learning. The underlying idea is that each document is a mixture of topics and each topic is a mixture of words. (the order of words in the document and the order of documents in the corpus don't matter). LDA assumes that a new document is created in the following way:

1. Determine number of words in the document
2. Choose a topic mix for the document over a fixed set of topics (ex. $\frac{1}{5}$ topic A, $\frac{3}{10}$ topic B, $\frac{1}{2}$ topic C) (Assumption: the topic distribution for each document follows a symmetric Dirichlet distribution $\sim$ Dirichlet(α)) (α: hyperparam.)
3. Generate the words in the document by: pick a topic based on the document's distribution of topics and pick word based on the topic's distribution of words (Assumption: the words distribution in each topic $\sim$ $\sim$ Dirichlet(η). (η: hyperparameter)

Remark on Dirichlet distribution: when we fixed the hyperparameter α/η, Dirichlet(α) is a distribution and has support in the simplex $\Delta^{n-1} = \{(t_1, \dots, t_n) : \sum_i t_i = 1 \quad t_i > 0\}$, so for example $(\frac{1}{5}, \frac{3}{10}, \frac{1}{2})$ as in the ex. above. The higher α/η is the higher is the probability to contain a mixture of topics/words.

How LDA works: 1. There are K topics and we randomly allocate each word to one topic.
(K: the number of topics has to be chosen ex ante (but not which topics))

(2) Now we exclude a specific word $\overline{w} \in \overline{d}$ (to avoid bias) and we compute:

- Topic distribution within document: $P(\text{topic } t \mid \text{document } \overline{d})$ not counting word $\overline{w}$. (there are $|\text{topic}|$ computations)
- Word $\overline{w}$ distribution across topics: $P(\text{word } \overline{w} \mid \text{topic } t)$ (there are $|\text{topic}|$ computations)
- So we assume that:

$$P(\text{word } \overline{w} \in \text{Topic } t) = P(\text{word } \overline{w} \mid \text{Topic } t) \cdot P(\text{Topic } t \mid \text{document } \overline{d})$$

(actually is proportional but it is still right because "argmax")

$\forall t$ Topic. Then we reassign topic t that maximixes the above computation to the word $\overline{w} \in \overline{d}$. We iterate.

Remark: for this metod, words that appears in the same document have higher odds to be labelled in the same way.

→ Key ATM: is basically an unsupervised method but with elements of supervised learning since the users can give Keywords to the model (difference from LDA).

How it works: (1) Assign keywords to some topics (total number of topics K)

(2) Fore each word i in document d, a latent topic z_{di} si drawn from the document's topic distribution θ_d. (as usual is a K-dimensional vector s.t. $\sum_{j=1}^{K} \theta_{dj} = 1$).

(3) For no Keyword-topics: the word w_{di} is drawn from the word distribution of the topic ϕ_K (ϕ_K is a V-dimensional vector s.t. $\sum_{j=1}^{V} \phi_{Kj} = 1$ where V is the vocabulary size)

(4) For Keyword-topics: s_{di} r.v. $\sim$ Bernoulli(π_K) (π_K: parameter for topic K):

- $s_{di} = 0 \Rightarrow$ draw w_{di} from the usual distr. ϕ_K
- $s_{di} = 1 \Rightarrow$ draw w_{di} from a different word distribution $\hat{\phi}_K$

↓

Evaluate Performance (LDA and Key ATM):

- Perplexity : divide the dataset into training and test set.

$W_{TEST} := \{ w_d : w_d \in \text{test set} \}$.

$$L(W_{TEST}) := \sum_{w_d \in W_{TEST}} \log L(w_d \mid \text{topics, topic distribution})$$

(L is the probability; w_d is a document!)

where $\log L(w_d \mid \text{topics, topic distribution}) := \sum_{w \in w_d} \log P(w \mid \Theta_d, \Phi)$

where $P(w \mid \Theta_d, \Phi) := \sum_{K=1}^{K} P(w \mid \text{topic } K) \cdot P(\text{topic } K \mid \Theta_d)$

$$\boxed{\text{Perplexity} = \exp\left(- \frac{L(W_{TEST})}{N_d} \right)}$$

Issue: not correlated to human judgement

- Topic Coherence : measure the degree of semantic similarity between high-scoring words in each topic. Examples:

$$\text{UMass coherence scores} := \overbrace{\frac{1}{\binom{N}{2}} \sum_{i<j}}^{\text{average on the pairs}} C_{UMass}(w_i, w_j)$$

where $C_{UMass}(w_i, w_j) := \log \left(\frac{D(w_i, w_j) + 1}{D(w_i)} \right)$

where $D(w_i, w_j) := \#\{ w_i \text{ and } w_j \text{ appears in the same document} \}$

and $D(w_i) := \#\{ w_i \text{ appears alone} \}$

$$\text{UCI coherence scores} := \frac{1}{\binom{N}{2}} \sum_{i<j} C_{UCI}(w_i, w_j)$$

where $C_{UCI}(w_i, w_j) = \log \frac{P(w_i, w_j) + 1}{P(w_i)\, P(w_j)}$ where we decide a sliding window (ex. a sliding window of 10 words: $[\overrightarrow{-10} \; \bar{w} \; \overrightarrow{+10}]$) and

$P(w) = \dfrac{\#\{ w \text{ appears in a window "centered" on a word } \bar{w} \}}{\text{total number of words in all windows}}$

$P(w_i, w_j) = \dfrac{\#\{ w_i \; w_j \text{ appears in the same window} \}}{\text{total number of windows}}$