MLR 570

Recording from Sept. 16: (part 1)

↳ PCA: maximizing variance vs. minimizing reconstruction error.

maximizing variance → eigenvalue decomp. on covariance matrix.

minimizing reconstruction error → singular value decomp. on dataset itself.

Covariance: Derivation.

Feature selection: Filter methods [independent of models]

- var threshold
  $var(X_j) = \frac{1}{n}\sum(x_{ij}-\mu_j)^2$
  ↳ feature $\in \mathbb{R}^n$
  Below threshold, remove. because doesn't contribute.
- corr. coef.
  $\rho(X_j, y) = \frac{cov(X_j, y)}{\sigma_{X_j}\sigma_y}$
- mutual information
  Not the equation here.

Wrapper methods [based on model] → importance of features depends on the specific models we are using.

If full data, leakage of data?

- RFE: rank based on importance & then recursively eliminate. → train w/ all, rank (model coefficients), remove least important.
  recursive feature elimination
- FFS (forward feature selection)
  No features → add features based on improvement
  1st most important feature, ++++.

Embedded methods [introduce something to obj. function to force features to be zero]
↳ During training

L1 regularization

force some feature coefficients to become 0, selecting subset of features.

$$\min_w \frac{1}{2n}\underbrace{\|Xw-y\|_2^2}_{L2\ norm} + \lambda\underbrace{\|w\|_1}_{L1\ norm}$$

Initialization:
Bad → algo in bad point. Not converge to good place.
Non-convex space

Norm: map vector to scalar
$a = [1\ 2\ 3]$

$\|a\|_2 = \sqrt{1^2+2^2+3^2}$

$\|a\|_1 = 1+2+3$
↳ L1, encourage vector to be sparse during training.

Similarity measures:

→ correlation coeff.: $\rho = \frac{cov(A,B)}{\sigma_A \sigma_B}$.

linear relationship between variables
Fails for non-linear relationships.

def. variance: $E(X^2) - [E(X)]^2 = var(X)$ ← Definition based on expectation.

↳ μ^2 mean

σ^2 = variance

σ = standard deviation $\sqrt{var(X)}$

$var(cA) = c^2 var(A)$

$cov(A,B) = \frac{1}{n}(A_i - \bar{A})(B_i - \bar{B})$

↳ covariance value.

- ρ cannot capture non-linear relationship. Does not mean that there's no relationship.
  - ↳ Rolling coefficient? As opposed to the whole data.

Euclidean distance: straight-line distance between two points in Euclidean space

↳ In high dimensions, it will not convey same information

High dimensional statistics

Limitations: does not have a meaning.

$$d(A,B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

Manhattan distance: Along right angles. Grid structure. Restricted, streets etc...

$$d(A,B) = \sum_{i=1}^{n} |A_i - B_i|$$

Recording from Sept. 16: (part 2)

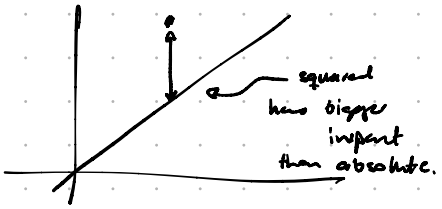
Adv. Manhattan over Euclidean ——→ grid

——→ High dimensional data: curse of dimensionality. dist. converges in higher dim.

Manhattan more stable. Axis independent.

——→ Better if features are independent. (Manhattan better)

——→ Manhattan more robust to outliers.



Closeness of clusters.

Cosine Similarity: cos of angle between two vectors. Direction instead of magnitude.

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Same direction, diff magnitudes ⟹ $\cos(\theta) = 1$ → angle is 0

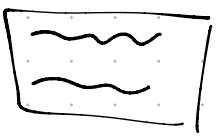
Robust to diff sizes when comparing. How change word to vector? encoding frequency. ($\cos^{-1}(1) = 0$)

Word2vec embedding. Not part of course

Jaccard similarity: Binary or categorical data. Document similarity is an application.

$$J = \frac{|A \cap B|}{|A \cup B|}$$

Also robust to size diff sizes.



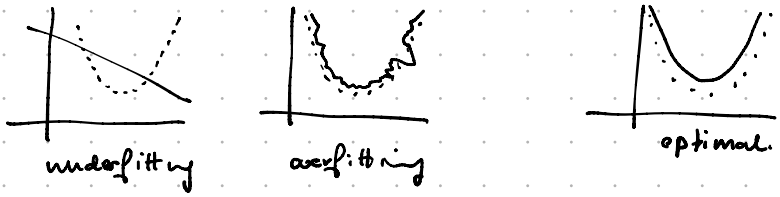
Recording from Sept. 23: (part 1):

Lecture 5: Model evaluation

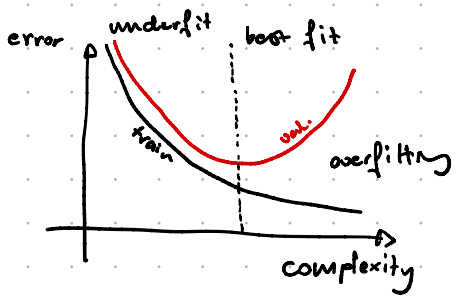
Exam: Think deeply about concepts.
Understand concepts well.

Training → training error
Validation → { tuning the hyperparameters.
→ validation error. ← intermediate testing set.
Testing → Generalization on unseen data.

Underfitting / Overfitting



← model too complex

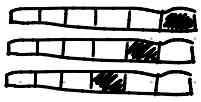


/ this can be the num. epochs as well.

Expressive power: how powerful model is regardless of data. Depends on parameters of the algorithm.
outside of course

We select model based on the different errors. Similar to model hyperparameter tuning.

K-fold cross validation
K-1 folds for testing.

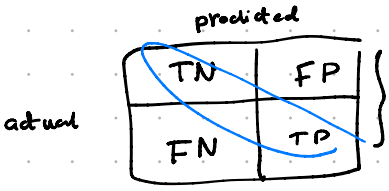


$\text{error}: \frac{1}{k}\sum_{i=1}^{k} \text{error}_i$

Can also look at the maximum possible error instead of average.
Reduce overfitting risk
More reliable.

Central limit theorem. error converges to certain value / distribution.

$\binom{n}{k}$ business? Shuffling? Other things.



accuracy misleading for imbalanced data.

We don't know acc. based on the classes. Just overall.

Precision: proportion of true positives among all predicted positive } limitation: not useful if we care about FN

Type I error: 1 − precision

Recall: proportion of true positives among all actual positives } high recall comes at the cost of precision.

Type II error: 1 − recall

F1 score: $2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$ ← does not account for TN.

Harmonic mean of precision & recall.

Why the harmonic mean? Balance between prec. & recall. Penalizes the extreme values more than the geometric or arithmetic means.
↳ Esp. if one is significantly smaller than the other.

Arithmetic: equal weights to both metrics

Harmonic mean is monotonic to both precision & recall. If one increase, it also increases

Geometric mean: multiplicative process. Growth, returns, compounded.

Harmonic $\leq$ geometric $\leq$ arithmetic } Equality if precision = recall.
↑ ↑
lower bounds each

Recording from Sept. 23: (part 2)

Begin with numerical examples. Harmonic mean penalizes extreme values.

If we swap FN & FP, then the f_1 score does not change. If prec. & recall swapped while everything else stayed the same, it wouldn't change f_1 [huge limitation].

↳ weighted f_1 score allows us to differentiate

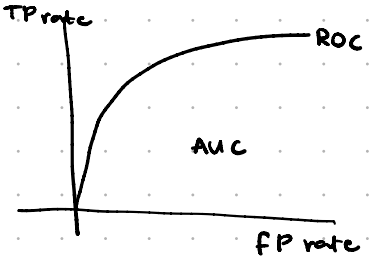
$$F_\beta = (1+\beta^2)\frac{\text{Precision} \cdot \text{recall}}{(\beta^2\,\text{precision}) + \text{recall}}$$

What if β value too large? then we have bad representation.

Generalized mean:
maximum $p=\infty$
cubic mean $p=3$
rms $p=2$
arithmetic $p=1$
geometric $p=0$
harmonic $p=-1$
minimum $p=-\infty$

$$M_p(x_1,\ldots,x_n) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^{\,p}\right)^{1/p}$$

Receiver-operator characteristic (type I, II error)



Mean squared error: $\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$ ⟶ heavy penalty for large values (squaring), amplified sensitive to outliers.

Mean absolute error: $\frac{1}{N}\sum_{i=1}^{N}|y_i-\hat{y}_i|$ Better for handling outliers.

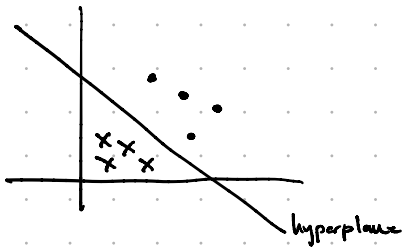
↳ optimize this loss function if we have outliers
↳ Cannot take derivative of absolute value function. Need to perturb, approximate, etc...

Side point: regularization does not only prevent overfitting, it helps us make sure the model does not learn things we don't want it to learn.

Recording from Sept. 30th: (part 1)

Supervised learning: $f: X \to Y$, $X \in \mathbb{R}^n$ to output label Y

regression / classification



↳ Linear classification:

↳ Linear separability: A dataset is linearly separable if $\exists$ hyperplane s.t. all datapoints belonging to one class are on one side, & vice versa.

For $\mathbb{R}^n$, hyperplane = $wx+b=0$, $x \in \mathbb{R}^n$. For linear separability condition, $\exists\, w, b$ s.t. $y_i(wx_i+b)>0\ \forall i$

w = weights, b = biases

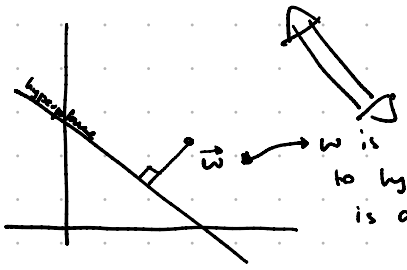
$y_i(wx_i+b) \longrightarrow$ for some w, b, if $y_i(wx_i+b) > 0$, $y_i, (wx_i+b) > 0$ or $y_i < 0$, $wx_i + b < 0$

$y_i(wx_i+b) < 0$, $\Rightarrow$ $y_i > 0$, $wx_i+1 < 0$

or $y_i < 0$, $wx_i + b > 0$

Orthogonal to the hyperplane.

Moving above is one class, moving below will give another class.



w is orthogonal to hyperplane. It is a vector.

If we move in direction of $\vec{w}$, positive class. If opposite direction, negative class.

Write the equation in standard form,

$wx + b = 0$

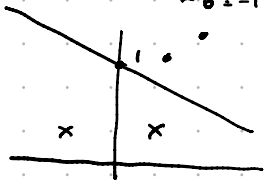
hyperplane

hyperplane: $w_1x_1 + w_2x_2 + b = 0$

e.g. $x_1 + x_2 - 1 = 0$

$b = -1$

$w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

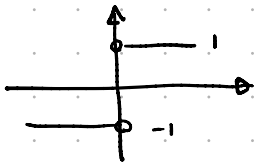


for $(1,-2)$: $wx+b = 1-2-1 = -2 < 0$

$y = -1 \Rightarrow -2(-1) > 0$

$\therefore$ correct classification.

Perceptron Algorithm: $f(x) = \text{sign}(wx+b)$ $w \in \mathbb{R}^n$, $b \in \mathbb{R}$



Hinge loss: $L(w,b) = \max(0, -y_i(wx_i+b))$

↳ no update if loss is 0.

if correctly classified, $y_i(wx_i+b) > 0$

$\therefore -(y_i(wx_i+b)) < 0$, $\Rightarrow \max(0, \epsilon)$

$= 0$. No loss.

↳ no loss $\rightarrow$ no update

Otherwise, $\max(0, (+)) > 0$ for incorrectly classified.

loss $\neq 0$. update.

Partial derivatives:

$$\frac{\partial L}{\partial w} = \begin{cases} -y_i x_i & \text{if } y_i(wx_i+b) \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial L}{\partial b} = \begin{cases} -y_i & \text{if } y_i(wx_i+b) \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

Gradient descent approach

updates:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} = w + \eta\, y_i x_i$$

$$b \leftarrow b - \eta \frac{\partial L}{\partial b} = w + \eta\, y_i$$

The direction of w for optimizing (where w is orthogonal to hyperplane) is super important in general.

Perceptron has infinite number of solutions.

order that we approach with also matters.

Local updates, not global updates

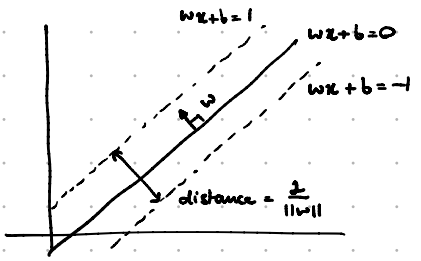
↳ computationally expensive.

↳ many algorithms do this.

Limitations:
- Linearly separable requirement. Diverge for non-linear data.
- No "margin" optimization.

Recording from Sept. 30th: (part 2)

SVMs: builds on the limitations of perceptron (margin optimization)



Now, the correctly classified points satisfy $y_i(wx_i+b) \geq 1 \ \forall i$

To maximize the margin, we minimize $\|w\|$ or $\frac{\|w\|^2}{2}$ (makes taking derivative easier).

Optimization: $\min_w \frac{1}{2}\|w\|^2$ subject to $y_i(wx_i+b) \geq 1 \ \forall i$

Using Lagrange multipliers: $L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(w\cdot x_i+b)-1)$

$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$

$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$

↳ replace $w = \sum_{i=1}^{n} \alpha_i y_i x_i$: $\max_\alpha \left[-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i x_j) + \sum_{i=1}^{n} \alpha_i \right]$

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0, \ \alpha_i \geq 0 \ \forall i$$

Why $\min_w \frac{1}{2}\|w\|^2$?

↳ The margin is the perpendicular distance between hyperplane & closest data points.

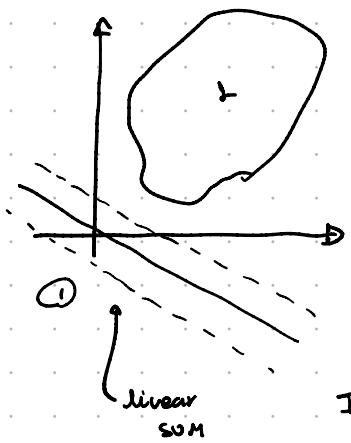
The distance from x_0 to hp: $d = \frac{|wx_0+b|}{\|w\|}$

The support vectors pass through $wx+b = \pm 1$. These are two other hyperplanes, not the decision boundary.

Total margin size: $\frac{2}{\|w\|}$ ⟶ why? $d = \frac{|wx_0+b|}{\|w\|}$, at top margin, $|wx_0+b| = 1$, at bottom $|wx_0+b| = -1 \Rightarrow 1$

$\Rightarrow \frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}$. Margin is inversely proportional to $\|w\|$. Larger $\|w\|$ means smaller margin.

When SVM / maximum margin bad?



In this case, class 2 has huge variance. Thus maximum margin may be troublesome. ⟶ solution can be soft margined SVMs.

Huge limitation of SVMs: computational complexity.

In general, bigger margin means more robustness.

Linear regression:

$f: X \in \mathbb{R}^{n \times p} \longrightarrow Y \in \mathbb{R}^n$

where $y = X\beta + \varepsilon$

↳ If ε is distributed normally with constant variance, then we have optimality.

$$\min_\beta \|y - X\beta\|_2^2 = \min_\beta \sum_{i=1}^{n} (y_i - X_i\beta)^2$$

Assume $p \leq n$.

$X = \begin{bmatrix} x_1 \\ x_n \end{bmatrix}$, $\|X\|_2^2 = \left(\sqrt{x_1^2 + x_n^2}\right)^2 = x_1^2 + x_n^2 = X^TX = [x_1 \ x_n]\begin{bmatrix} x_1 \\ x_n \end{bmatrix}$

$\|(y - X\beta)\|_2^2 = (y - X\beta)^T(y - X\beta)$

$= y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta$ ← minimize

$\frac{\partial}{\partial \beta} = 0 = 0 - (y^TX)^T - (X^Ty) + X^TX\beta + (\beta^TX^TX)^T = 0$

$-X^Ty - X^Ty + X^TX\beta + X^TX\beta = 0$

$= -2X^Ty + 2X^TX\beta = 0$

$\Rightarrow 2X^Ty = 2X^TX\beta$

$\boxed{\beta = (X^TX)^{-1}X^Ty}$ ← This is what we use to optimize.

$\beta = (X^TX)^{-1}X^Ty$

When can we move $(\cdot)^{-1}$ inside?

So $\beta = X^{-1}\underbrace{(X^T)^{-1}X^T}y$; $X \in \mathbb{R}^{n \times p}$,

$= X^{-1}y$ → if $n = p$ assuming full rank.

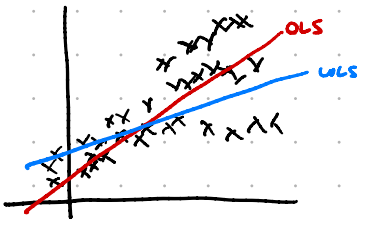
Why do we do $(X^TX)^{-1}$ instead of $(XX^T)^{-1}$?

$p \leq n$, so (XX^T) is singular, cannot take the inverse. $\text{rank}(X) \leq \min(n, p)$.

Limitations:
- Assumes linear relationship
- Constant variance
- Singularity in $(X^TX)^{-1}$ → perturbs $(X^TX + \varepsilon I)^{-1}$ exists

Weighted least squares (WLS): assumes observations have non-constant variance. errors associated w/ different observations are not the same.

↑ observations w/ higher variance have less weight, and vice versa.



low variance → high variance

$\min_\beta \sum_{i=1}^{n} w_i (y_i - X_i\beta)^2$

$w_i = \frac{1}{\sigma_i^2}$ ← variance of observation

↳ w is a diagonal matrix.

$\begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_n^2 \end{bmatrix}$

Derivation:

$\min_\beta (y - X\beta)^T w (y - X\beta)$

$\Rightarrow \frac{\partial}{\partial \beta} = X^Twy = X^TwX\beta$

$\Rightarrow \beta = (X^TwX)^{-1}X^Twy$

XX^T can be singular, often because of multi-collinearity. Unstable → OLS unreliable

Ridge regression: $\min_\beta \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$

↳ regularization term.

λ shrinks coefficients to stabilize the solution.

$\Rightarrow (y - X\beta)^T(y - X\beta) + \lambda \beta^T\beta$

$= -2X^Ty + 2X^TX\beta + 2\lambda\beta = 0$

$X^Ty = X^TX\beta + \lambda\beta$

$X^Ty = (X^TX + \lambda I)\beta$

$\beta = (X^TX + \lambda I)^{-1}X^Ty$

↑ new factor that does perturbation to stabilize.

$\lambda \geq 0$. if $\lambda = 0$, we get OLS

if $\lambda > 0$, penalize large coefficients.

Recording from Oct. 7th: (part 1)

Lasso and ElasticNet added.

Lasso: $\min_\beta \|y - X\beta\|_2^2 + \lambda\|\beta\|_1$,

- good for feature selection, encourages sparsity
- high-dim data
- Interpretability

but
- instability with correlated features
- bias
- no closed form solution.

Elastic: $\min_\beta \|y - X\beta\|_2^2 + \lambda_1\|\beta\|_2^2 + \lambda_2\|\beta\|_1$,

Not part of course.

- flexible reg.
- stability for correlated features
- flexible regularization.

- more complex tuning
- over-selecting features
- computationally expensive.

Huge limitation of linear classification

Projection of lower dim to higher dim. for some $x \in \mathbb{R}^{d_1}$, $\phi(x): \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ with $d_2 > d_1$. In this space, we can use the same linear models (because of linear separability)

Let $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$. Then define $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix} \in \mathbb{R}^3$

The transformation makes the data linearly separable.

$$h(\phi(x); \theta, \theta_0) = \text{sign}(\theta^T \phi(x) + \theta_0)$$
$$= \text{sign}(\theta_1 x_1 + \theta_2 x_2 + \theta_3(x_1^2 + x_2^2) + \theta_0)$$

The linear hyperplane in $\mathbb{R}^{d_2}$ would became non-linear when projected back down to $\mathbb{R}^{d_1}$

Learning non-linear shape by using linear model in higher dimensional linearly separable data.

Recording from Oct. 7th: (part 2)

Mapping from lower dimension to higher can be computationally expensive. Choice of kernel function is important.

Polynomial function: $x \in \mathbb{R}^n$, polynomial degree d.

$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$

Why? Relationship can be non-linear. This allows us to capture those.

Let $x \in \mathbb{R}^2$; $d = 2$ $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

size: $\binom{n+d}{n}$ ← size of vector of our polynomial

If I have $x_1^2 \Rightarrow j_1 = 2, j_2 = 0, j_3 = 0$

$x_2^2 \Rightarrow j_1 = 0, j_2 = 2, j_3 = 0$

$$\phi(x) = \left[\frac{\sqrt{d!}}{\sqrt{j_1! j_2! \cdots j_{n+1}!}} x_1^{j_1} \cdots x_n^{j_n} \cdot 1^{j_{n+1}} \right]_{j_1 + \cdots + j_{n+1} = d}$$

→ for $x \in \mathbb{R}^2$, $d = 2$:

Not sure about the scaler factor.

$$\binom{n+d}{d} = \binom{4}{2} = 6$$

$$\begin{Bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{Bmatrix}$$

Highest power is 2 because $d = 2$

Size of vector grows quickly. Lots of memory.

How does polynomial kernel allow good modeling? / linearly separable.

If n is fixed, d varies, what is the dim? $\frac{(n+d)!}{d!\,n!}$ $\Longrightarrow$ As d increases $\to \infty$; It grows moderately and then polynomially

Bottom line is, if n is fixed & only d grows, $\frac{(n+d)!}{d!\,n!}$ also grows.

Radial basis function: $\phi(x) = \exp(-\gamma x^2)\begin{bmatrix} 1 \\ \sqrt{\frac{(2\gamma)^1}{1!}}x \\ \sqrt{\frac{(2\gamma)^2}{2!}}x^2 \\ \vdots \end{bmatrix}$

Let's go back to perceptron algo.

$\theta \leftarrow 0$
for $k = 1, \dots, T$
  for $i = 1, \dots, n$
    if $y^{(i)}(\theta^T x^{(i)}) \le 0$:
      $\theta \leftarrow \theta + y^{(i)} x^{(i)}$

Rewrite: $\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)}$

updates:

1	$y^{(1)}x^{(1)}$	✓	✓
2	✓	✓	✓
3	✓	$y^{(3)}x^{(3)}$	✓
4	$y^{(4)}x^{(4)}$	$y^{(4)}x^{(4)}$	✓

$\theta = y^{(1)}x^{(1)} + y^{(4)}x^{(4)} + y^{(3)}x^{(3)} + y^{(4)}x^{(4)}$

$= y^{(1)}x^{(1)} + y^{(3)}x^{(3)} + 2y^{(4)}x^{(4)}$

$\alpha_1 = 1,\ \alpha_2 = 0,\ \alpha_3 = 1,\ \alpha_4 = 4$

Just a different way of showing the same thing.

$\Longrightarrow$ rewrite algo.:

$\theta \leftarrow 0$; $\alpha_i \leftarrow 0\ \forall i$
for $k = 1, \dots, T$:
  for $i = 1, \dots, n$
    if $y^{(i)}\left[\sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)T} x^{(i)}\right] \le 0$
      $\alpha_i \leftarrow \alpha_i + 1$

The two algorithms are equivalent.

If in higher dim, $\phi(x)$ instead.

Why do we do transpose?? Because $\theta^T x$

Indices are different as well.

for $k = 1, \dots, T$
  for $i = 1, \dots, n$
    if $y^{(i)}\left[\sum_{j=1}^{n} \alpha_j y^{(j)} \phi(x^{(j)})^T \phi(x^{(i)})\right] \le 0$
      $\alpha_i \leftarrow \alpha_i + 1$

$\phi(x)$ can be huge. Use dot product instead. Does this improve things?

Hopefully this makes things simpler.

for $k = 1, \dots, T$
  for $i = 1, \dots, n$
    if $y^{(i)}(\theta^T \phi(x^{(i)})) \le 0$
      $\theta \leftarrow \theta + y^{(i)}\phi(x^{(i)})$

Only for poly & rbf kernels.

$\phi(x^{(j)})^T \phi(x^{(i)}) = (1 + x^{(j)T} x^{(i)})^d$, if ϕ poly
$\phi(x^{(j)})^T \phi(x^{(i)}) = \exp(-\gamma \|x^{(j)} - x^{(i)}\|_2^2)$, if ϕ rbf.

} This is done in the original space!!

Numerical example: $d = 2$, $n = 2$. $\phi(x) = [1\ \sqrt{2}x_1\ \sqrt{2}x_2\ x_1x_2\ x_1^2\ x_2^2]^T$

$\phi(x') = [1\ \sqrt{2}x_1'\ \sqrt{2}x_2'\ x_1'x_2'\ x_1'^2\ x_2'^2]^T$

$\phi(x)^T \phi(x') = 1 + 2x_1x_1' + 2x_2x_2' + 2x_1x_1'x_2x_2' + x_1^2(x_1')^2 + x_2^2(x_2')^2$

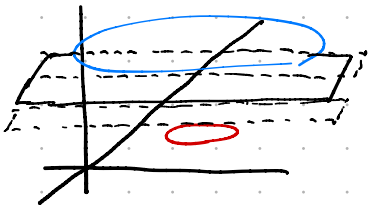
Calculate $(1 + x^T x')^2$

$= 1 + 2x_1x_1' + 2x_2x_2' + 2x_1x_1'x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2$

They are equivalent. This is the kernel trick. Inner product is the same. Simplest & most convincing approach to kernel trick.

NO SLIDES on kernel SVM.

↳ Exactly the same idea. Kernel SVM considers maximum margin.



Kernel details:

- Additivity: kernel + kernel = kernel
- Scalar multiplication is valid kernel
- Product of kernels is kernel
- Exponentiation of kernel is a kernel

Kernel Least squares (KLLS): Just add the ϕ parameter.

$$\min_\beta \|y - \phi(X)\beta\|_2^2 \longrightarrow (y - \phi(X)\beta)^T(y - \phi(X)\beta)$$

$$\frac{\partial}{\partial\beta}\left(y^Ty - y^T\phi(X)\beta - \beta^T\phi^T(X)y - \beta^T\phi^T(X)\phi(X)\beta\right)$$

$$= 0 - \phi^T(X)y - \phi^T(X)y + \phi^T(X)\phi(X)\beta + \phi^T(X)\phi(X)\beta$$

$$= -2\phi^T(X)y + 2\phi^T(X)\phi(X)\beta$$

$$\Longrightarrow \phi^T(X)y = \phi^T(X)\phi(X)\beta \Longrightarrow \beta = (\phi^T(X)\phi(X))^{-1}\phi^T(X)y$$

What if we add the regularization term?

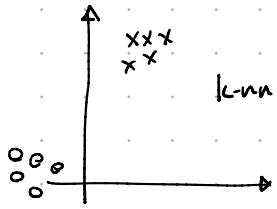
$$\min_\beta \|y - \phi(X)\beta\|^2 + \lambda\|\beta\|_2^2 \longrightarrow \text{same derivation as before.}$$

$$\beta = (\phi^T(X)\phi(X) + \lambda I)^{-1}\phi^T(X)y$$

Limitations:

- complex patterns, like time varying
- overfitting: Easily overfit. 100 samples, fit poly on 99, 100 can overshoot
- Curse of dimensionality (arbitrarily high dim.)
- Parameter tuning is non trivial.

According from Oct. 20th:



k-nn clustering. Closest neighbours.

Euclidean distance.

	distance to d_i	class
d_1	0.05	+1
d_2	0.25	−1
d_3	0.13	−1
d_4	0.08	+1

1-NN for d_q: +1 (d_1)

2-NN for d_q: +1 (d_1, d_4)

⋮

mode operation

What if there is a tie in mode?

→ weights for closer points

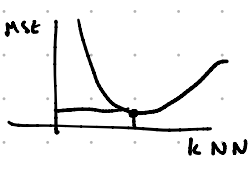
→ Randomly choose

→ Prefer closer.

Many different distance metrics.

Higher dimensions → Euclidean does worse.

Choose k:



k-NN for regression:

weight	height
60	160
65	165
90	
95	195
80	180

1-NN: 175 or 165

2-NN: $\frac{175+165}{2} = 190$.

arithmetic mean.

If tie, change k or something else?
Multi-objective optimization.

For query point x_q:

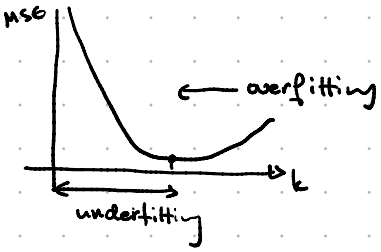
$$\hat{y}_I = \frac{1}{k} \sum_{i=1}^{k} y_i$$

If there are cases where better to use geometric or harmonic mean?
↳ Target values & features have multiplicative nature. Like stocks.

To improve performance, can use weighted.

$$\hat{y}_q = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i} \quad ; \quad w_i = \frac{1}{d(x_q, x_i)}$$

Weight of decay of information in high dim. is lower than Euclidean for Manhattan.



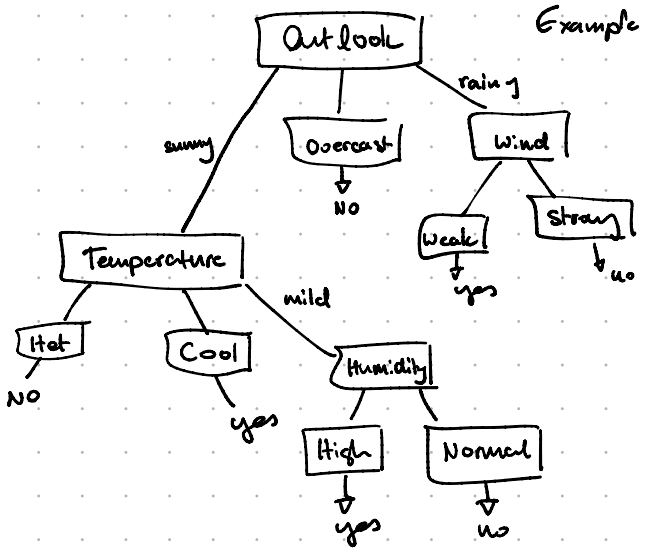
Approximate algorithms
Random algorithms

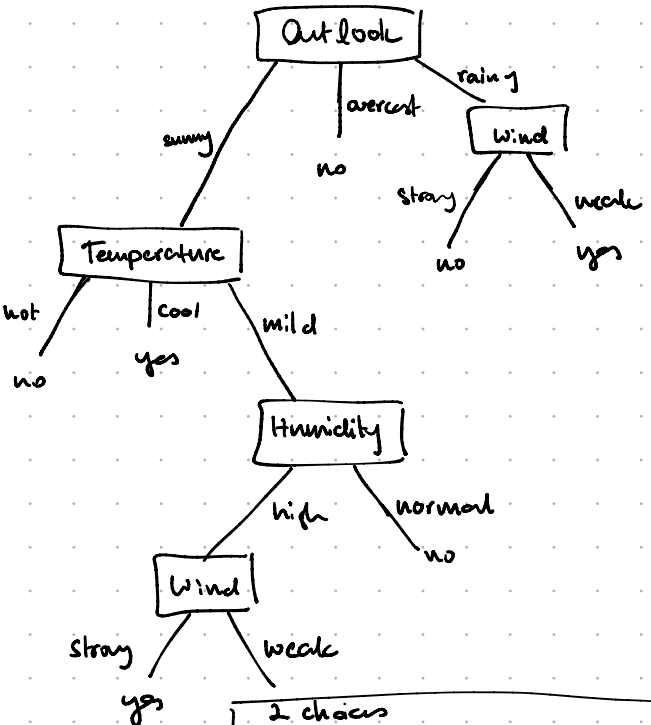
Limitations of k-NN:
- Curse of dimensionality (Use LDA, PCA, etc...)
- Computational complexity (Use KD-trees, ball trees)
- Imbalanced datasets bad
  ↳ Use weighted voting.

Decision Trees:
↳ Internal nodes split data, leaf nodes classify.
How do we construct tree? When do we stop?

} He goes through a DT example here.

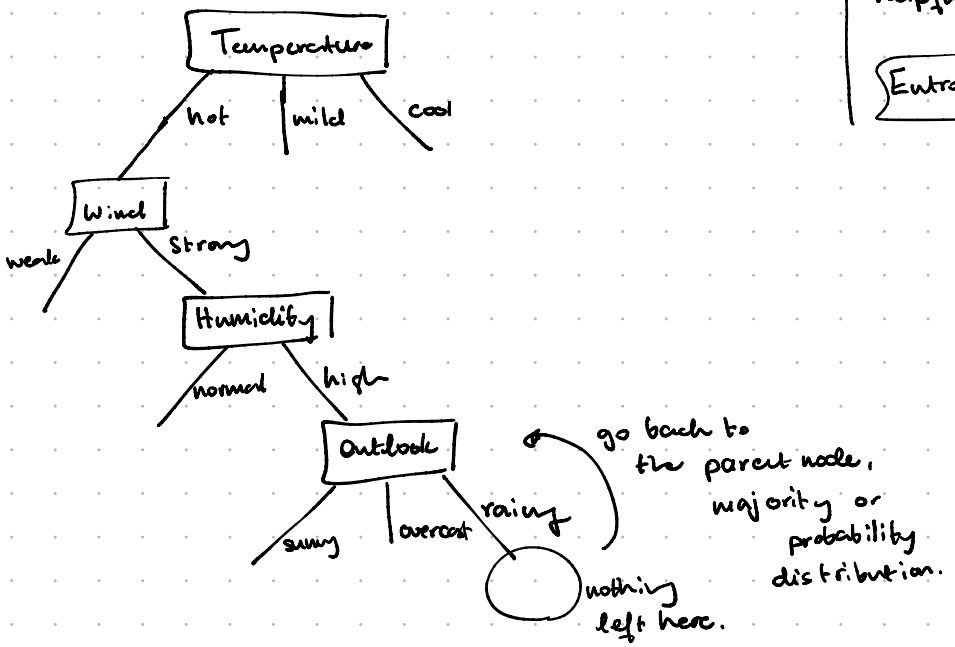
Example 1.





Local calculations in DT: at start, outlook was best. Does not mean optimal by the end.

Example 3: Sunny → hot → high → weak



Kernel decision trees? Not really helpful (or possible)?

Entropy minimization function!

Two interesting cases:
- → no features left to split on
- → no data left for a feature
  - Go back to parent,
  - └→ Majority or probability distribution.

Which feature? Optimal order is computationally expensive. Use greedy approach instead. Very local.

Data → temp, humid, wind, outlook } then keep going. But we don't know what happens next.

Local exploration

We want to reduce the uncertainty at each step. At root, maximum uncertainty

└→ If all classes equally distributed, the uncertainty is maximized. We want to maximize the entropy. What pdf does this? Uniform!!

$-\sum_{i=1}^{k} P(c_i)\log_2(P(c_i)) = H.$ → $\log_2$ because we are referring to bits.

If two classes (both equally likely), how many bits to represent them? 1 bit. If we start to skew the data in one direction (more likelihood), then we need less bits to represent it.

ID3 type of decision tree that uses information gain.

Go back to example: $I(9/14, 5/14) = -(9/14\log_2(9/14) + 5/14\log_2(5/14))$

$= 0.94.$

		Info-gain
Humidity:	0.789	0.151
Temperature	0.911	0.029
Wind	0.891	0.0485
Outlook	0.694	0.249

Outlook: sunny | overcast | raining

sunny	overcast	raining
y: 2, n: 3	y: 4, n: 0	y: 3, n: 2
$I(2/5, 3/5)$	$I(4/4, 0) = 0$	$I(3/5, 3/5)$
$= 0.971$	$= 0$	$= 0.971$
$5/14\,(0.99)$	0	$5/14\,(0.971)$

⟹ $0.94 - 0.694$
$= 0.249$

Why choose highest information gain?
⟹ lowest possible entropy.

So the root node is gonna be the outlook.

We can keep going. Get the new entropy when we go in.

What if there is a tie in information gain? No idea. Diana was saying some stuff about ratios.

Another metric we can use is Gini Index (CART): $Gini(t) = 1 - \sum_{i=1}^{c} p_i^2$

Where c is number of classes, p_i proportion of instances belonging to class i at node t.

Unify the framework: Tsallis Entropy. We have $S_q(X) = \frac{1}{1-q}\left(\sum_{i=1}^{n} p_i^q - 1\right)$; $q \in \mathbb{R}$

Choosing an optimal q is still an open problem.

$\lim_{q \to 1} S_q(x) = H(x)$

$S_2(X) = Gini(X)$

Limit growth of tree:

→ pre-post pruning

- max depth,
- Ø minimum
- examples in leaf nodes, etc...

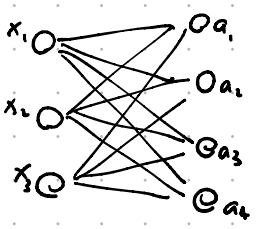
fully grow the tree first.

Recording from Oct. 28th:

Feed forward Neural Networks (FNNs):

$f: X \to Y$, $X \in \mathbb{R}^n$, Y labels. Components:

- layers (depth)
- width (Neurons in each layer)
- Activation functions
- Loss functions

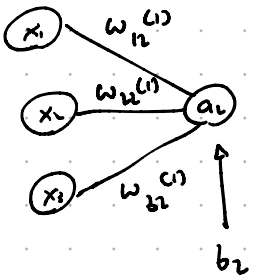


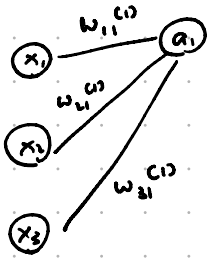
$a_1 = f(w_1 X + b_1)$; $X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}$, $b_1 \in \mathbb{R}$

↳ activation function.

Based on this notation:

$a_2 = f\left[w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)} x_3 + b_2 \right]$





$a_1 = f\left[w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + b_1 \right]$

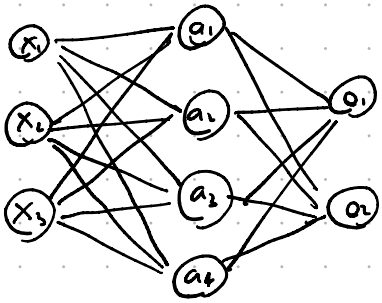
f can be linear or non-linear.

Let's look at a classification example:

$a_1 = f\left[w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 \right]$

Recall one-hot encoding:

1	2	3	4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



$o_1 = f\left[w_{11}^{(2)} a_1 + w_{21}^{(2)} a_2 + w_{31}^{(2)} a_3 + w_{41}^{(2)} a_4 + b^{(2)} \right]$.

What would be the activation function here? Softmax. Let $o_1 = f(\alpha_1)$

At this stage, we don't how to initialize weights, update weights, but assuming weights are there, we can do forward propagation to set outputs o_1 & o_2.

$$P(C_1) = \frac{e^{\alpha_1}}{e^{\alpha_1} + e^{\alpha_2}}$$

$$P(C_2)\ o_2 = f(\alpha_2) = \frac{e^{\alpha_2}}{e^{\alpha_1} + e^{\alpha_2}}$$

} sums to 1 (probability)

Loss function: $l = -(y \log_2(o_2) + (1-y)\log_2(o_1))$

$= -[y \log_2(o_2) + (1-y)\log_2(1-o_2)]$

Since $o_2 + o_1 = 1 \Rightarrow o_1 = 1 - o_2$

What if we have 3 classes? Assuming we have linear activations. $l = -\sum_{i=1}^{3} p_i \log_2(o_i)$

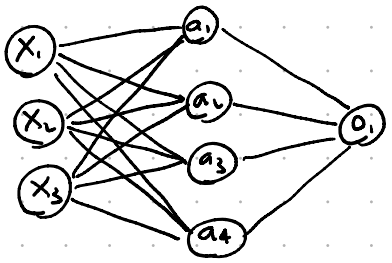
	p_0	p_1	p_2
class 0	1	0	0
class 1	0	1	0
class 2	0	0	1

↳ noiseless labeled data.

$l = -\sum_{i=1}^{3} p_i \log_2(o_i)$ This is cross-entropy

After training, take argmax to get the prediction.

What if we decide to do regression? Exactly the same. Output are regressors now.



For demonstration:

$a_1 = f[w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + b_1]$

$o_1 = f[w_{11}^{(2)} a_1 + w_{21}^{(2)} a_2 + w_{31}^{(2)} a_3 + w_{41}^{(2)} a_4 + b_2]$ ← linear function.

Loss function: $l = (y - o_1)^2$ ↳ Squared error loss.

What if we had two outputs? i.e. $y \in \mathbb{R}^2 = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \Longrightarrow$

$$\begin{cases} o_1 = f(w_1^{(2)} a + b_1^{(2)}) \\ o_2 = f(w_2^{(2)} a + b_2^{(2)}) \end{cases}$$

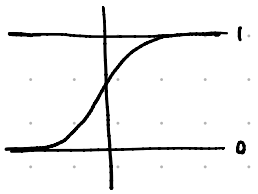
↳ linear function

$$l = \frac{1}{2}\left[(y_1 - o_1)^2 + (y_2 - o_2)^2\right]$$

↳ mean squared error.

This only works for linear data. If we want to work w/ non-linear data, use non-linear activation.

↳ Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$

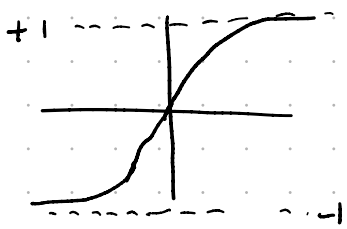


Limitation: Vanishing gradient at small or large values.

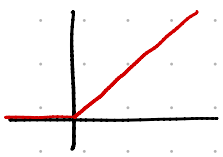
↳ tanh function $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

0-centered output (this apparently helps with the convergence)

Larger gradients compared to the sigmoid, but can still vanish for Deep NNs.



↳ ReLU: $f(x) = \max(0, x)$



Solves the vanishing gradient problem for large values. But now we have exploding.

- Dead neurons issue.

Vanishing gradient: so small that we have no updates.

ReLU ⟺ few neurons

What did kernels do? Low-dim to high to make data linearly separable. Goal of activation functions is to do the same across the depth / layers to separate data in higher dim.

Before: non-linear data → kernel → linearly separable space (high-dim) → kernel perceptron

NNs: Data → NN → O_1, O_2

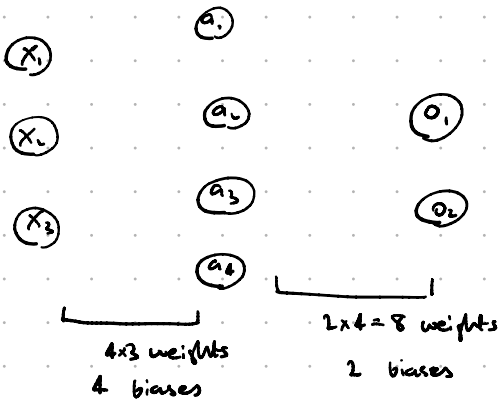
l_1 l_2 ....

This, we want to be linearly separable. } Just the last layer.

Lots of mappings & transformations, but only at the last layer do we care about linear separability.

Activation functions are element-wise activations. $f\left(\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}\right) = \begin{bmatrix} f(a_1) \\ f(a_2) \\ \vdots \end{bmatrix}$

↖ softmax layer requires linearly sep. data.



Update weights & biases using the gradients. We turn to the same idea of gradient descent.

Stochastic = random. Probabilistic, not deterministic.

1000 samples → gradient descent] large number of calculations.

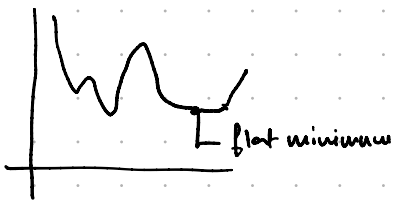
↳ 10 batches of 100 samples → batch gd or sgd] much simpler.

↳ The gradients calculated have higher variance ⟹ allows us to explore the non-convex optimization landscape.

What if we use momentum?
Exponential decay of previous gradients, gives us global information about the shape of the function we are exploring. } "Why not explore a little more?" Velocity vector.

Adaptive learning rate: slow down as we get to optimal point.

The optimization landscape does not have to be convex. Some minima are hard to get out of. That's why we use momentum / adaptive LR.



fractional derivative?
To get the update.
Not local anymore.
w.r.t. other parameters.

→ Backpropation: Recall o_1, o_2

$\Rightarrow l = \frac{1}{2}\left[(y_1 - o_1)^2 + (y_2 - o_2)^2\right]$

$$\frac{\partial l}{\partial w_{11}^{(2)}} = \frac{\partial o_1}{\partial w_{11}^{(2)}} \frac{\partial l}{\partial o_1}$$

$$w_{11}^{(2)} = w_{11}^{(2)} - \gamma \frac{\partial l}{\partial w_{11}^{(2)}}$$

$$\frac{\partial l}{\partial o_1} = -(y_1 - o_1)$$

$$\frac{\partial o_1}{\partial w_{11}^{(2)}} = a_1$$

$$\Rightarrow \frac{\partial l}{\partial w_{11}^{(2)}} = -a_1(y_1 - o_1)$$

→ $w_{11}^{(2)} = w_{11}^{(2)} + \gamma a_1 (y_1 - o_1)$

We have a batch size. What is a_1?

$a_1 = f\left[w_{11}^{(1)} X_1 + w_{21}^{(1)} X_2 + w_{31}^{(1)} X_3\right]$

$w_{11}^{(1)}$, $w_{21}^{(1)}$, $w_{31}^{(1)}$ ← random. X inputs.

We update in backprop. after we do forward prop.

This is just for one. But we can extend to a lot more.

For final: need to have mathematical maturity to play around with these things. Describe scenario → use algo., get results.

Might get backprop., not framed as neural network.

Possible questions:

Weighted least squares. (non constant variance)

Derive, get computational complexity, etc...

Data points + labels, but don't say that they are non-linearly separable.

He will not tell us what exactly to do.

Won't just be mechanical.

Another question: $A \in \mathbb{R}^{m \times n}$

Derive least squares: $(A^T A)^{-1}$ & $(A A^T)^{-1}$ } differences.

↓ if $m \leq n$, invertible or not, regularization, etc...

Practice not same as the exam.

Novel, looking for some interesting function, creative solution.

Activation functions separate in different spaces / shapes. Look at the shapes they are representing

↳ Traing process: move the lines to come up with something more interesting.

Weight initialization: → Random from distribution. Can cause large or small gradients

→ He: $W \sim N(0, \sqrt{\frac{2}{n_{in}}})$

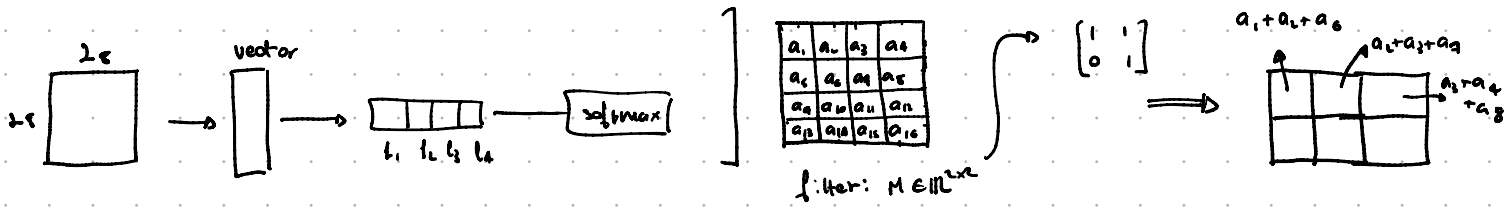
→ Xavier: $W \sim U\left(\frac{-\sqrt{c}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{c}}{\sqrt{n_{in}+n_{out}}}\right)$

Depth more important than width for expressive power. Proven.

Shuffling: Exam review example. Shuffle data. Don't order / sequential. The more you shuffle, the better.

Recording from 3rd November:

Convolutional Neural Networks (CNNs)

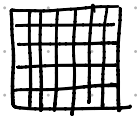


filter: $M \in \mathbb{R}^{2 \times 2}$

Element-wise multiplication. Padding. The 1st filter would have the same parameters. This extracts some local features.

We need to move the filter. Called stride. over x & y axes.

We can use multiple features to extract different features. } Spatial structure. Feature extractor layer before we put into the neural network.



convolved features } pass this through the activation function

There are some different filters.

Regardless, our goal is the prediction made at the end. NN is end-to-end, from feature extraction to the classification.

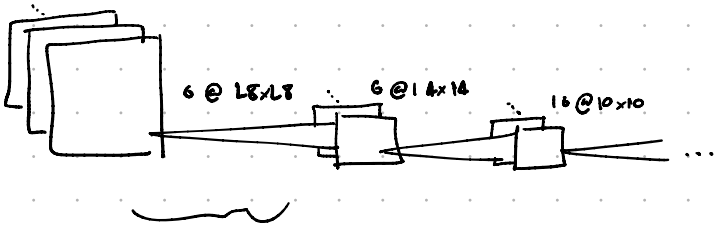
Feature engineering → reduces burden of feature extraction from NN.

Padding: → zero padding controls the output size.
same convolution.
→ valid-only convolution: output only if the entire kernel contained in input.

Is there change in trainable parameters? / number of operations
→ yes, more operations.

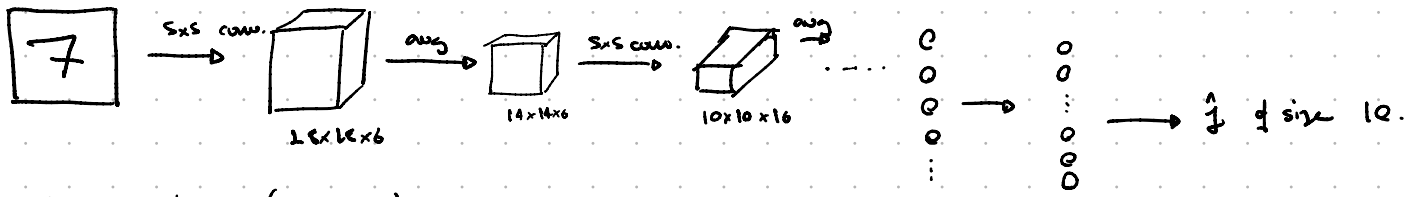
This is powerful for images, 2D data, speech.

LeNet-5: 3 convolutional layers (different hierarchies of features)



What does pooling do? Some sort of dimensionality reduction.
→ max, avg, etc...

6, 5x5 filters
= 150 trainable parameters.



→ $\hat{y}$ of size 10.

Output size: $\frac{(N+2P-F)}{\text{stride}} + 1$.

AlexNet: deeper in terms of layers, conv. + ReLU.

227 x 229 x 3 (224 before padding)

First layer: 96 filters, size 11x11, stride = 4

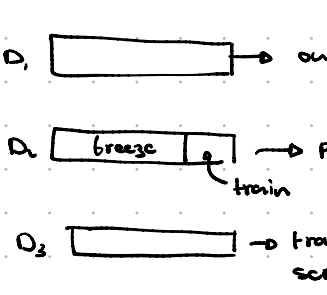
⇓

5 convolutional layers, size of filters important.

Equivalence between point-wise & something.

Usually NNs have been trained before transfer learning. AlexNet on ImageNet.

AlexNet(pretrained) + fine-tuning on new dataset



is there a case where $p_2 > p_1$?
Transfer learning is popular for a reason.
↳ Dimensions that contribute to the classification.

Right now, every data point is completely independent. We do not assume any dependencies between samples.

Dropout gives you more regularization → equivalent to L2.

VGG Net: deep, better performance. Gives us hierarchical rep. of visual data to work.
↳ smaller filters, deeper.

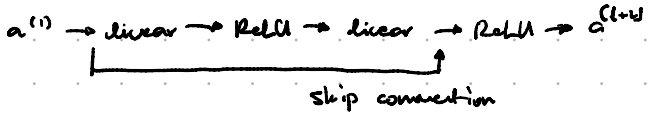
For now, same filter size even though maybe different init.
What if we vary filter size & then concatenate?
→ This is Inception models.

GoogleNet: no fully connected layers.
No more stacking of conv. + pooling layers.

The idea of vanishing gradients. For deep NNs.

Skip connections?? → helps w/ vanishing gradient.

ResNet uses skip connections.

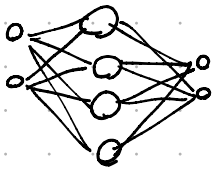


What happens if we just keep stacking?
Deeper models perform worse, but do not overfit.

ResNet 152 layers, but better performance.
Mechanism of skip connections not explained.

Right now, we deal with one-one. Input → output.
many-one. Inputs → output
one-many Image + caption
many-many (RNNs)

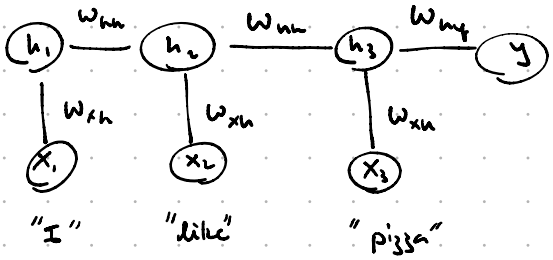
FFNs:



} Limitation: no "memory" of past, since weights learned independently → many parameters.

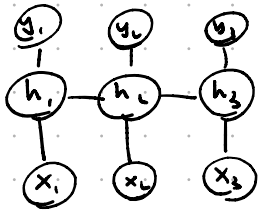
] Solution: Recurrent neural nets. (RNNs)

RNNs:

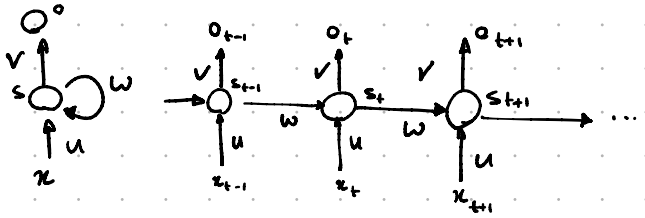


computational efficiency → weights are shared.

many-many



Many-one



The model shares the parameters (W, U, V). The things that are different are $s_{t-1}, s_t, s_{t+1}, \dots$
$x_{t-1}, x_t, x_{t+1}, \dots$

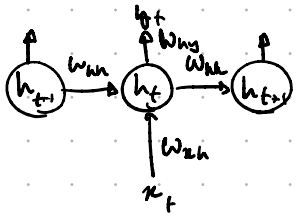
In lecture he uses h_t instead of s_t.

$h_t = \sigma(W_{xh} x_t + W_{hh} h_{t-1} + b)$ ← simplest RNN.
↳ activation function.

$y_t = W_{hy} \cdot h_t$ [might have softmax, etc... or linear]

If we get to h_{t+100}. Backprop, the intuition is it needs to propagate all the way. The longer your data, the worse (longer sequence). ⇒ vanishing gradient possible

$W = T \Sigma T^{-1}$
$W^n = T \Sigma^n T^{-1}$
} eigenvalue decomp.
↳ if $\lambda > 1$, exploding.
$\lambda < 0$, vanishing



$h_t = \tanh[W_{xh} x_t + W_{hh} h_{t-1} + b]$
current input, previous output

$\sum_{a=t}^{T} \frac{\partial l_a}{\partial W_{hh}}$

$\Rightarrow \frac{\partial l_T}{\partial y_T} \frac{\partial y_T}{\partial h_T} \boxed{\frac{\partial h_T}{\partial h_t}} \frac{\partial h_t}{\partial W_{hh}}$

↑ interested in this.

Recording from Nov. 4th: (part 1)
→ continuation of RNN

$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_{t+1}}{\partial h_t} \cdots \frac{\partial h_{k-1}}{\partial h_{k-2}} \frac{\partial h_T}{\partial h_{t-1}}$

Assume $T = 0$; $t = 0$:

$\Rightarrow \frac{\partial h_1}{\partial h_0} \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial h_4}{\partial h_3} \frac{\partial h_5}{\partial h_4} = \frac{\partial h_5}{\partial h_0}$

$$h_{t+1} = \tanh\left[W_{xh} x_{t+1} + W_{hh} h_t + b\right]$$

$$\frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial}{\partial h_t}\left[\tanh\left(W_{xh} x_{t+1} + W_{hh} h_t + b\right)\right]$$

$$= \text{diag}\left[\tanh'\left(W_{xh} x_{t+1} + W_{hh} h_t + b\right) \cdot W_{hh}\right]$$

$$W_{hh} = T \Sigma T^{-1}$$

$$W_{hh}^{\alpha} = T \Sigma^{\alpha} T^{-1}$$

exploding / vanishing gradient depending on the Σ^{α}.

We can solve exploding gradient problem by clipping the gradients.

↳ Vanishing is more common.

Let's change the way RNN works to solve vanishing gradient.

↳ Gates: → update gate: long term dependencies → $z_t = \sigma(W_{xz} x_t + W_{hz} h_t + b_z)$

↑ $0 \text{ or } 1 \Rightarrow$ sigmoid.

→ reset gate: short term dependencies.

↳ $r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1} + b_r)$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \tanh\left(W_{xh} x_t + r_t \odot (W_{hh} h_{t-1}) + b_h\right)$$

- Let z_t be 1: $\Rightarrow h_t = h_{t-1}$. There is no effect from x_t. Get rid of that if $z_t = 1$. [update gate]
- Let $r_t = 0$: $z_t \odot h_{t-1} + \tanh\left[W_{xh} x_t + b_h\right]$
- Let $z_t = 0$: $\tanh\left[W_{xh} x_t + r_t \odot (W_{hh} h_{t-1}) + b_h\right]$

↑ what if $r_t = 1$?

$\Rightarrow \tanh\left[W_{xh} x_t + W_{hh} h_{t-1} + b_h\right]$

This is the usual RNN.

Extreme cases.

By setting z_t & r_t, control the memory factor.

Why do we do this? Get rid of the vanishing gradient.

$\Longrightarrow$ final result:

$$\frac{\partial h_{t+1}}{\partial h_t} = \text{diag}(z_t) + \text{diag}\left(\tanh'\left(W_{xh} x_{t+1} + r_{t+1} W_{hh} h_t + b_h\right)\right) \cdot \text{diag}(r_{t+1}) \cdot W_{hh}$$

↑ controls things.

We no longer study only W_{hh}, no more vanishing gradient.

Recording from Nov. 4th: (part 2)

Unsupervised Learning: no labels.

↳ Clustering: group things together based on similarity measure.

Assume k clusters $[C_1, \dots, C_k]$. All have some representative point $[z_1, \dots, z_k]$

$$\min_{j=1,\dots,k} \text{dist}(x_i, z_j)$$

cost function: $\sum_{j=1}^{k} \sum_{i \in C_j} \| x_i - z_j \|_2^2$

$$\text{dist}(x_i, z_j) = \| x_i - z_j \|_2^2$$

Two types: Hard, soft. ← varying degree of membership / confidence.

↳ $C_i \cap C_j = \{\phi\}$

Certainty

k-means clustering (Hard):

specify k num clusters.

for each cluster, randomly select $(z_1, \dots, z_k)$

$\min_{j=1,\dots,k} \|x_i - z_j\|_2^2$ for i size of dataset.

cost: $\sum_{i=1}^{n} \min_{j=1,\dots,k} \|x_i - z_j\|_2^2$

$\min_{z_1,\dots,z_k} \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - z_j\|_2^2 = \text{cost.}$

$\frac{\partial}{\partial z_j} \sum_{i \in C_j} \|x_i - z_j\|_2^2 = 2 \sum_{i \in C_j} x_i - z_j$

$\therefore \sum_{i \in C_j} x_i = \sum_{i \in C_j} z_j \Rightarrow z_j = \frac{\sum_{i \in C_j} x_i}{|C_j|}$

Only applies for Euclidean distance.

Alternating manner.

↳ Now, we update the representative point. Two separate steps for better computation.

Example:

Point A (2,3) $z_1 = (2,3)$

B (3,3) $z_2 = (6,5)$

C (6,5)

D (8,8)

Point A: $(2-2)^2 + (3-3)^2 = 0$ → z_1; $(2-6)^2 + (3-5)^2 = 20$ → z_2

$A \in C_1$

Point B: $(3-2)^2 + (3-3)^2 = 1$ → z_1; $(3-6)^2 \dots$ → z_2

$B \in C_1$

$C \in z_2$

$D \in z_2$

$z_1^{(2)} = \left(\frac{2+3}{2}, \frac{3+5}{2}\right) = (2.5, 4)$

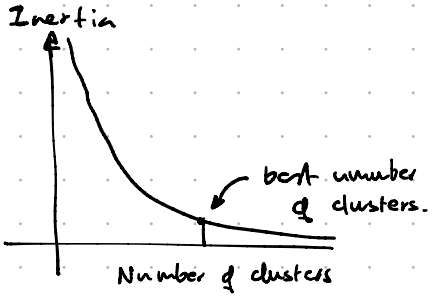
$z_2^{(2)} = (7, 6.5)$

} do calc. again

$A, B \in C_1^{(2)}$

$C, D \in C_2^{(2)}$

Done. stop the algo.



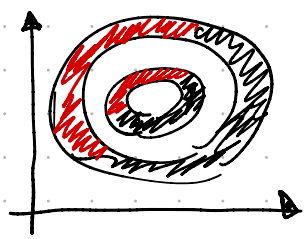
We can also use k-means for quantizing. k-means kernel clustering?

Initialization is important (Limitation)

↳ Interesting initialization perhaps. No manual clustering. k-means ++. Maximize distance to get better init.

Choose random rep. Then distance to all other points calculated. p(select next rep.) based on the dist. squared.

What if we have spherical clusters?



Not good clusters. ⟹ Kernel k-means clustering

Apply ϕ kernel function. Everything else is the same.

1. $\min_{j=1,\dots,k} \|\phi(x_i) - z_j\|_2^2$

2. $\min_{z_1,\dots,z_k} \sum_{j=1}^{k} \sum_{i \in C_j} \|\phi(x_i) - z_j\|_2^2$

$\Rightarrow z_j = \frac{\sum_{i \in C_j} \phi(x_j)}{|C_j|}$ Again, kernel trick: $\phi^T(x_j)\phi(x_i) = (1 + x_j^T x_i)^d$ for poly.

$\phi^T(x_j)\phi(x_i) = \exp(-\gamma \|x_j - x_i\|_2^2)$, for rbf.

$$\|\phi(x_i) - z_j\|_2^2 = \left\| \phi(x_i) - \frac{\sum_{i \in C_j} \phi(x_j)}{|C_j|} \right\|_2^2 = \phi^T(x_i)\phi(x_i) - 2\sum_{i \in C_j} \frac{\phi^T(x_i)\phi(x_i)}{|C_j|} + \sum_{t \in C_j} \frac{\phi^T(x_t)\phi(x_t)}{|C_j|^2}$$

k means only works for squared Euclidean distance. If another cost function, k-medoids.

1. Specify k
2. $(z_1, \dots, z_k) \subset (x_1, \dots, x_n)$
3. Iter.: $\sum_{i=1}^{n} \min_{j=1,\dots,k} dist(x_i, z_j)$

$\sum_{j=1}^{k} \sum_{i \in C_j} dist(x_i, z_j)$

for each z_j will go over $\{x_1, \dots, x_n\}$ & find based on lowest error.

→ not necessarily closed form, could be a search space

The rest is the same

Maybe cost function captures it better.

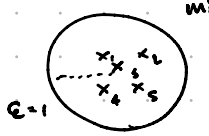
Data dependent problem.

Recording from Nov. 11th:

Density-based clustering: clusters are dense regions separated by low density regions.

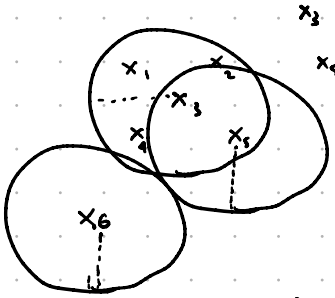
↳ DBScan: { core points: more than pts_{min}
border points: less than pts_{min}, in neighbourhood.
noise points: not core or border

Density: points within circle w/ radius r.



num. points in neighbourhood of x_2: 5 > 4.

x_2 core point.



x_3 core point

x_5 contains 3 < 4

∴ border point.

x_6 contains 1;

∴ noise point

This has nothing to do with centroids.

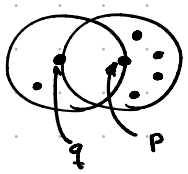
Only core points.

Algo. first finds all the core points, then checks for border & noise.

Check if x_j is in the neighbourhood of a core point, if points < pts_{min}.
If it is, border point. Else noise.

Directly density reachable: q ddr from p if p is a core point & $q \in \epsilon$-ball of p.

Assymetric



q ddr from p

p is not ddr from q

↳ because q is not a core point.

ε-ball of q < pts_{min}

There is also indirect density reachable.

DBscan algo.: ε & pts_{min}.

Identify core, border, noise points.

Choose core point randomly. All core points close to this assigned to same cluster, until all the core points assigned. ⟹ allows arbitrary shape.

Then, if there are border points to the core points, they will be added, but if a point is in ε-ball of border point & not core point, not added

Example: $\varepsilon = 2$, $pts_{min} = 3$.

A	1	2
B	2	2
C	2	3
D	8	7
E	8	8
F	25	80

for A: B & C in ε-ball
for B: ⋮

There are no core points based on $\varepsilon=2$, $pts_{min}=3$.

change $pts_{min} = 2$

A, B, C core points.

D, E, F noise. Why not border?

Not in ε-ball of any core points.

Change $\varepsilon = 8$, $pts_{min} = 3$

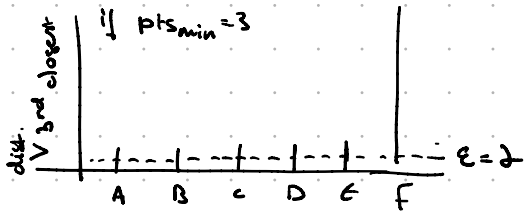
B, C, D core

A, E border ⟶ In ε-ball of B, C for both.

F noise

How do we specify ε & pts_{min}? $O(n^2)$ comp. complexity.

We can divide by 2.



We face issues because same density ε for all data. Assumes clusters homogeneous.

HDBscan: varying density. Multi-scale clustering

DBscan better than k-means:
- No need for k
- Handles noise
- Arbitrary shape

Soft-clustering: Model the data using a probability distribution. Gaussian dist. w/ different means & covariance
Multiple Gaussians models, combine to get the Gaussian mixture model.
What can we say about diagonal covariance? if cov = 0, implies independence for Gaussian distribution.

Bernoulli dist.: p { head P, tail 1-P

Binomial dist: k successes from n trials

Multinomial dist.: multiple outcomes, sums to 1.

k components in the mixture model. $\mathcal{N}(x; \mu_j, \sigma_j^2 I)$; $j = 1, \dots, k$

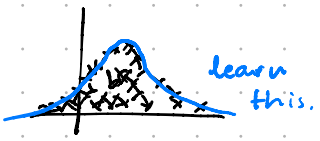
↳ Weight parameter (multinomial dist.) to combine the Gaussian dist.

$$p_1, \dots, p_k \; ; \; p_j \geq 0, \; \sum_j p_j = 1$$

$$\theta = \{p_1, \dots, p_k, \mu_1, \dots, \mu_k, \sigma_1^2, \dots \sigma_k^2\} \leftarrow \text{parameters.}$$

$$\begin{aligned} P(S_n | \theta) &= P(x_1, \dots, x_n | \theta) \\ &= P(x_1|\theta) P(x_2|\theta) \dots P(x_n|\theta) \\ &= \prod_{i=1}^{n} P(x_i|\theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} p_j \mathcal{N}(x_i; \mu_j, \sigma_j^2 I) \end{aligned}$$

How do we learn these parameters?



Can represent everything w/ two parameters.
Also, generator.

term to maximize. Gaussian dist.

$\prod_{i=1}^{\hat{n}_1} \frac{1}{\sqrt{2\pi}\,\sigma_1} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_1^2}\right)$. Take derivative w.r.t μ.

dist. between 0,1. if values in denom. large, then goes to 0. Take log.

$$\log\left[\prod_{i=1}^{\hat{n}_1} \frac{1}{\sqrt{2\pi}\,\sigma_1} \exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right)\right]$$

$$\Rightarrow \sum_{i=1}^{\hat{n}_1} \log\left(\frac{1}{\sqrt{2\pi}\,\sigma_1} \exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right)\right)$$

$$= \sum_{i=1}^{\hat{n}_1} \log\left(\frac{1}{\sqrt{2\pi}\,\sigma_1}\right) - \frac{(x_i-\mu_1)^2}{2\sigma_1^2}$$

$$\rightarrow -\frac{1}{2}\sum_{i=1}^{\hat{n}_1} \log(\sqrt{2\pi}\,\sigma_1) - \frac{1}{2}\sum_{i=1}^{\hat{n}_1} \frac{(x_i-\mu_1)^2}{\sigma_1^2}$$

Derivative, w.r.t μ_1

$$\frac{\partial}{\partial \mu_1} = 0 - \frac{2(-1)}{2}\sum_{i=1}^{\hat{n}_1} \frac{(x_i-\mu_1)}{\sigma_1^2} = 0$$

$$\sum_{i=1}^{\hat{n}_1} \frac{(x_i-\mu_1)}{\sigma_1^2} = 0 \Rightarrow \sum_{i=1}^{\hat{n}_1} x_i = \sum_{i=1}^{\hat{n}_1} \mu_1$$

$$\Rightarrow \hat{n}_1\mu_1 = \sum_{i=1}^{\hat{n}_1} x_i \Rightarrow \mu_1 = \frac{\sum_{i=1}^{\hat{n}_1} x_i}{\hat{n}_1}.$$

Now we found μ.

How to get the variance?

Simplified version:

$\delta(j|i) = \begin{cases} 1 & \text{if } x_i \text{ assigned to } C_j \\ 0 & \text{otherwise} \end{cases}$

↳ maximum likelihood.

"forcing hard clustering in soft cluster"

$\hat{n}_j = \sum_{i=1}^{n} \delta(j|i)$ ← check all points in C_j.

$\hat{p}_j = \frac{\hat{n}_j}{n}$ ← probability belong to C_j

$\sum_{j=1}^{\hat{n}} \hat{n}_j$

$\hat{\mu}_j = \frac{1}{\hat{n}_j}\sum_{i=1}^{n} \delta(j|i)\, x^{(i)}$

$\hat{\sigma}_j^2 = \frac{1}{\hat{n}_j}\sum_{i=1}^{n} \delta(j|i)\, \|x^{(i)} - \mu_j\|_2^2$

$$\frac{\partial L}{\partial \sigma_1^2} = \frac{2\pi}{2\pi\,\sigma_1^2}$$