Lecture 10

Machine Learning

Let's go!!

when we want to look at specific types of errors → confusion matrix

accuracy: correct preds / predictions
cheat: do well on common classes, will automatically get high accuracy
misclassification rate: 1 - accuracy

0-1 Loss matrix:

$$* \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} L(1,1) & L(1,2) \\ L(2,1) & L(2,2) \end{bmatrix}$$

Binary confusion matrix:

true \ pred	1	0
1	TP	FN
0	FP	TN

← actual positives
← actual negatives
↑ predicted as positive
↑ predicted as negative

Possibility: collapse k classes onto 1 class
True Positive Rate: TP/P
False Positive Rate: FP/N

Bayes:
$P(Y=1|x) > P(Y=2|x)$
$P(Y=1|x) > 1 - P(Y=1|x)$
$P(Y=1|x) > 1/2$ ← threshhold

Alternatives to Bayes Classifier:

Bayes: class with highest posterior prob.
Alt.: choose a different threshhold

Posterior expected loss: L_k for class k if we just classify EVERYTHING to class k

$L_1(x) < L_2(x)$
$L(2,1)P(Y=2|x) < L(1,2)P(Y=1|x)$
$L(2,1)(1 - P(Y=1|x) < L(1,2)P(Y=1|x)$
$\frac{L(2,1)}{L(1,2)+L(2,1)} < P(Y=1|x)$

* relative size of the losses
0-1 Loss has threshold 0.5
$\frac{\text{L(2,1)}}{\text{L(1,2)+L(2,1)}} = \frac{1}{1+1} = \frac{1}{2}$
losses

Loss matrix

$\begin{bmatrix} L(1,1) & L(1,2) \\ L(2,1) & L(2,2) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$p_t = \pi_0$ = prob. threshhold

$$\frac{1-p_t}{p_t} = \frac{L(1,2)}{L(2,1)}$$

$$L(1,2) = \frac{1-p_t}{p_t} L(2,1)$$

0-1 Loss → Bayes Classifier

$$\text{threshold} = \frac{L(2,1)}{L(1,2)+L(2,1)} = \frac{1}{1+1} = \frac{1}{2}$$

other Losses:

threshhold: between 0 and 1

matrix: $\begin{bmatrix} 0 & 5 \\ 2 & 0 \end{bmatrix}$ ← penalizes FN more than FP

precision TP/predicted positives
how many predicted pos. are correct?

recall TP/actual positives
how many actual positives are classed correct?

precision and recall have an inherent tradeoff

F1-score $F_1 = \frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{P \cdot R}{P+R}$ ← harmonic mean of Precision and Recall

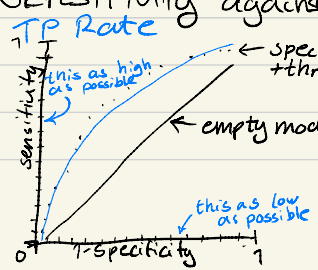
harmonic mean: $\left(\frac{1}{n}\sum_{i=1}^{n} x_i^{-1}\right)^{-1}$ ← avg. of inverse

$\left(\frac{a}{b}\right)^{-1} = \frac{1}{a/b} = \frac{b}{a}$

Macro F_1-score For k classes, we have k F_1-scores
macro F_1-score is the average of F_1-scores:

$$\frac{1}{K}\sum_{k=1}^{K} F_1\text{-score}_k$$

ROC-curve Sensitivity against 1-specificity

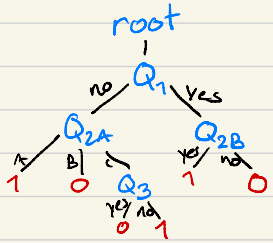


AUC: Area Under Curve
answers: which curve has least tradeoff?
empty model AUC = 1/2

Lecture 11

Decision Trees

we start from the root, and ask Q's to figure out which way to proceed:



Binary Trees: each join has exactly 2 children!

Process: we like shorter and more balanced trees, since performance is better!

1. Find feature with best split
2. Add to tree, and repeat on new subsets
3. Stop once criteria is reached

Falls under Supervised Learning

Terminology: Nodes, Edges, Root, Leaf/Terminal node - endpoints
Internal/Interior node - Leaf & root are not these
Decision node, Child & Parent, Descendant & Ancestors
Subtree & Branch

- Input features can be any type
- Output can be class or numerical (Regression)
  - if numerical; entity in leaf is average

Avoid overfitting: end criterions! f.ex. minimum # nodes in leaf

Assumption: Data is somewhat clustered or otherwise grouped!

We divide featurespace into non-overlapping decision regions!
regions are easily explainable :)

Advantage: We only need to save the tree, and not necessarily the data.
For balanced trees, time is $O(\log_2(L))$ ← levels

Method is top-down, recursive & greedy
- starting with root node,
- splitting subsets recursively
- testing all features to decide best cut!

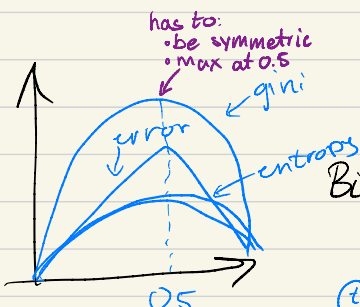
Split Criterion Regression Tree:

greatest possible reduction of RSS

minimize: R_{S1} & R_{S2} are splits of current subset

$$\sum_{i:x_i \in R_{S1}} (y_i - \hat{y}_{R_{S1}})^2 + \sum_{i:x_i \in R_{S2}} (y_i - \hat{y}_{R_{S2}})^2$$

squared error Region 1 — squared error Region 2



Binary Classification Tree:

smallest possible Gini-impurity (varies 0→0.5)

Impurity measures

$\Phi(t)$: impurity at t

to be maximized → Goodness: $\Phi(t) - (P_R \Phi(t_R) + P_L \Phi(t_L))$ — minimize this

Gini-impurity $G(t) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{k} p_k^2$

Binary: $G(t) = 2p(1-p)$

Entropy $H(t) = -\sum_{k=1}^{K} p_k \log p_k$

Decision Trees continued...

Goodness

maximize this

$$\Phi(t) - (P_R \Phi(t_R) + P_L \Phi(t_L))$$

$\Phi(t)$: impurity at parent; minimize this 'weighted average'

weighted Avg.

$\emptyset$ → impurity N → # entities

$$\frac{N_L}{N_T}\emptyset_L + \frac{N_R}{N_L}\emptyset_R = P_R \Phi(t_R) + P_L \Phi(t_L)$$

$P_R \Phi(t_R)$: right child; $P_L \Phi(t_L)$: left child

Missing features

handled by surrogate splits

↳ feature that gives most similar division to optimal

↳ we can have as many as we like, ranked

Stopping Criteria

1. Stop when having fully pure leaf nodes
2. Stop when there's no more goodness
3. Stop when leaf has < x nodes
4. Stop at depth y

Bias & Variance

Bias: spot on or shifted?

Variance: collected or spread out?

For decision trees:

few splits: high bias

many splits: overfit; high test error

Regularization / Shrinkage

How do we prevent overfitting?

- Early stopping / pre-pruning
  - We simply stop at a specific depth
- Pruning / post-pruning
  - We look at each possible subtree from a fully grown tree, compare performance w/wo subtree and Remove subtree, if leaf has equally good perf.

Ensemble Methods

Wisdom of crowds

Law of Large Numbers: $\frac{1}{m}\sum_{i=1}^{m} x_i \to \bar{x}$ as $m \to \infty$

The more predictors you use, the more likely you are to arrive at correct prediction by majority vote!

Increases accuracy of model :)

The predictors form an ensemble

BUT! predictors need to be independent

↳ in particular, errors should be independent

Heterogenous

Different types of classifiers on same data

↳ different algorithms make diff. errors

Hard voting: majority vote wins

Soft voting: all classifiers return $p(k|x)$

↳ winner is class with highest avg. $p(k|x)$

Homogenous

Different data with same classifiers

↳ less individual point Bias/weight

Bootstrapping: Random sampling with replacement

Bagging: Bootstrap, train models in parallel, aggregate preds :)

BUT! not truly independent (originate from same data)

Out of bag eval.

because we sample with replacement, there will be datapoints that are not in our training data → eval model on these

Bagging reduces variance

Pasting is like bagging, but w/o. replacement

Random Forest

A forest of bagged decision trees, but!
- we only consider few features for best split (randomly chosen) INSTEAD of all...

m is the number of trees
k is the number of features considered

Extra Trees

Extremely Randomized Trees
↳ we pick EVERYTHING at Random.
↳ at Last we compare which is best

Boosting

Bagging does not reduce bias, so we invent boosting!
Boosting is recursively reweighting & re-training the classifier, making misclassified points have (higher weight?)
The error of the tree is used to determine the weight of the prediction in the final vote :

Ada Boost

Starting weight: $w_i = \frac{1}{N}$ iterations: $c = 1, \dots, M$
trained models: $h_c(x)$ for each c
training error: $\epsilon_c = \sum_{i, h_c(x_i) \neq y_i} w_i$ summing weights of wrong preds

Model weight: $\alpha_c = \ln\left(\frac{1-\epsilon_c}{\epsilon_c}\right)$ Learning rate possible
reweighting: $w_i = w_i \cdot e^{\alpha_c}$ (only misclassifications) then normalize: $w_i = \frac{w_i}{\sum_{j=1}^{m} w_j}$ (all points)

Final Predictor: $\text{sign}\left(\sum_{c=1}^{M} \alpha_c h_c(x)\right) = \text{pred}(y \mid x)$

Gradient Boost

"small steps in the right direction"

We start with a single leaf, and train a classifier on the residuals (errors) of the current model.

We use weak learners and low learning rate to avoid overfitting.

$$pred = P_{init} + LR \cdot P_{res1} + LR \cdot P_{res2}$$

Stacking

Instead of aggregating model outputs, we train a meta-learner (another classifier) to blend the model outputs.

Hard Margin SVM

We minimize loss by tweaking parameters in a 2D space...

i.e. $\nabla f(x,y) = 0$ ← slope/gradient is 0

Remember Lagrange multipliers?

optimised when gradients of function & constraint

$\nabla f(x,y) = \lambda g(x,y)$

Generalisation:

if min. of $f(x)$ within constraint → inactive constraint

if min of $f(x)$ NOT within → $\nabla f(x) - \lambda g(x) = 0$

Lambda function: $L(x,\lambda) = f(x) - \lambda g(x)$ minimize x, maximize λ

Karush-Kuhn-Tucker (KKT): $\lambda g(x) = 0 \quad \lambda \geq 0 \quad g(x) \geq 0$

Hard SVM

we only care about the datapoints on the margin...

see slides week 15

assumption: seperable binary data points

1. find widest street/band
2. middle of this is decision boundary

Method:

we look at the points and do:

$\bar{w} \cdot x_i + b \geq 1$ for blue points

$\bar{w} \cdot x_i + b \leq -1$ for red points

$y_i(\bar{w} \cdot x_i + b) = 1$ when blue=1 & red=-1

street defined by: $y_i(\bar{w} \cdot x_i + b) - 1 = 0$ (we do not know these)

$$\text{width} = (x_+ - x_-) \cdot \frac{\bar{w}}{\|w\|} = \left(\frac{1-b}{\bar{w}} - \frac{-1-b}{\bar{w}}\right) \cdot \frac{\bar{w}}{\|w\|} = \dots = \frac{2}{\|w\|}$$ ← maximize this width of band, by;

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{constraint } y_i(\bar{w} \cdot x_i + b) - 1 \geq 0$$

Lagrange: $L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_i^n \alpha_i \left[y_i(\bar{w} x_i + b) - 1\right]$

Lagrange continued ☺

Lagrange: $L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_i^n \alpha_i \left[y_i(\bar{w}x_i + b) - 1\right]$

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{N} \alpha_i y_i x_i \qquad \frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\Rightarrow L(\alpha) = \sum_i^n \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$\text{s.t. } \alpha \geq 0 \quad \sum_i^n \alpha_i y_i = 0$$

Decision Rule

$$h(x) = \sum_i \alpha_i y_i (x_i \cdot x) + b$$

SVM Support Vector Machines

Soft margin — when we cannot seperate them easily...
But how do we decide to disregard points that are overlapping / outliers?
Softening constraint: let some points sit between

$$y_i(w \cdot x_i + b) \geq 1$$

Slack variable — a variable, each point has ξ_i, now constraint is:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

$\xi_i > 0$ for points between margins, $=0$ for correct

Penalize slack variable:

$$\min_{w,b} \underbrace{\frac{1}{2}\|w\|^2}_{\text{max street width}} + \underbrace{C\sum_{i=1}^{n}\xi_i}_{\text{minimize slack}} \quad \text{const: } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0$$

Now we also care about the things IN the street

Lagrange Result:

$$\max_{\alpha} \sum_{i}^{n} \alpha_i - \frac{1}{2}\sum_{i}^{n}\sum_{j}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

w. constraint: $0 \leq \alpha \leq C$ & $\sum_{i}^{n} \alpha_i y_i = 0$

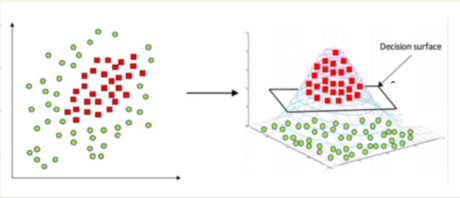
Decision Rule

$$h(x) = \sum_{i \in S} \alpha_i y_i (x_i \cdot x) + b \qquad b = \frac{1}{N_s}\sum_{i \in S}\left(y_i - \sum_{j \in S} \alpha_j y_j (x_i \cdot x_j)\right)$$

C — constant defining how much we care about error... Hyperparameter tuning on this

Non-Linear boundaries

if we can map to a higher dimension?



we define a transformation $\phi(x)$

↳ becomes slow fast!

Kernel Function:

$O(d)$ instead of $O(2^d)$

$$\phi(x) \cdot \phi(z) = \prod_{k=1}^{d} (1 + x_k z_k)$$

For training

$$\max_{\alpha} \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j (\phi(x_i) \cdot \phi(x_j))$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C \quad \text{and} \quad \sum_{i}^{n} \alpha_i y_i = 0$$

Decision boundary (for predictions):

$$h(x) = \sum_{i}^{n} \alpha_i y_i (\phi(x_i) \cdot \phi(x)) + b$$

For training

$$\arg\max_{\alpha} \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C \quad \text{and} \quad \sum_{i}^{n} \alpha_i y_i = 0$$

Decision boundary (for prediction):

$$h(x) = \sum_{i}^{m} \alpha_i y_i K(x_i, x) + b \qquad b = \frac{1}{N_S} \sum_{i \in S} (y_i - \sum_{j \in S} \alpha_j y_j K(x_i, x_j))$$

Kernels

Kernel is a dot-product between 2 points after they have been transformed

Gradient Descent

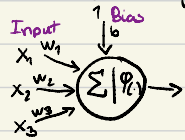
Main Idea	
Batch GD	Find Gradients for all datapoints and average/sum them... This is now the Gradient to follow for GD
Stochastic GD	Find Gradient for a single random datapoint, and use this for GD
Mini-Batch GD	Randomly choose a subset of the dataset, and do same as batch.

Basics

Artificial Neural Networks

Perceptron

Inspired by biological neural network
↳ receives a signal, triggers sending signal(s)



$\varphi(.)$ activation function

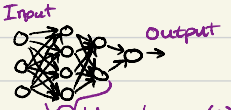
I/O

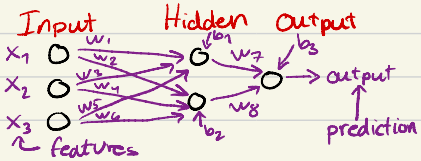
Binary: 0 or 1 ← is it active or not?

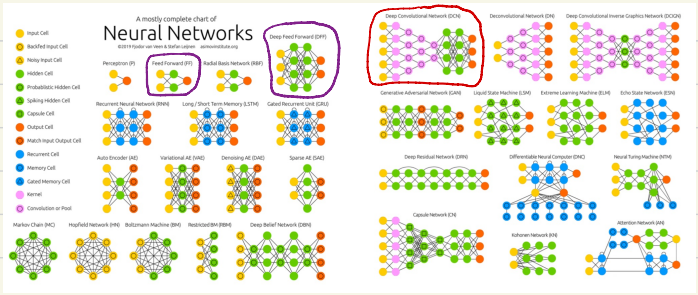
$\varphi(.)$ Activation

instead of 0-1 I/O, ⌐ sigmoid, tanh, ReLU
we can do differentiable ReLU

FF Feed Forward

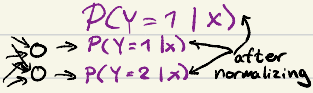






Output $\varphi(.)$

Regression : Linear function
Binary : sigmoid
Multiclass : softmax



Gradient Descent for weights

$w_1, \ldots, w_n$ & $b_1, \ldots, b_m$ are parameters.
Randomly init of parameters, GD until convergence

Forward & Back Propagation

Forward pass (starting from input layer toward output layer): compute the output and then the loss function on the training samples.

Back propagation (starting from output layer toward input layer): use the computed loss for updating the values of W and b.

(n) hidden
(o) output

Data is passed on, when received;

- summed $z_n^{(n)} = \sum x_i w_{i,n}^{(n)} + b_n$
- activated $a_n^{(n)} = f(z_n^{(n)})$
- passed on to next layer

Vectorized (matrix form):

Input layer $\mathrm{x}_{1\times 3} = [x_1, x_2, x_3]$

Hidden layer $\mathrm{z}^{(h)}_{1\times 4} = [z_1^{(h)}, z_2^{(h)}, z_3^{(h)}, z_4^{(h)}] = \mathrm{x}_{1\times 3}\mathrm{w}^{(h)}_{3\times 4} + \mathrm{b}^{(h)}_{1\times 4}$ $\quad \mathrm{a}^{(h)} = f(\mathrm{z}^{(h)})_{1\times 4}$

Output layer $\mathrm{z}^{(o)}_{1\times 1} = [z_1^{(o)}] = \mathrm{a}^{(h)}_{1\times 4}\mathrm{w}^{(o)}_{4\times 1} + \mathrm{b}^{(o)}_{1\times 1}$ $\quad \mathrm{a}^{(o)} = g(\mathrm{z}^{(o)})_{1\times 1}$

Automatic Different.

Back-propagation is the automated diff. in reverse mode; it is RECURSIVE!

$$\frac{\partial Y}{\partial x} = \frac{\partial f}{\partial g(x)} \cdot \frac{\partial g}{\partial x}$$

outer diff. ← what to find now
inner diff. ← previous step in recursive

III

Artificial Neural Networks

What if we get stuck in ~~~ ?

Momentum Gradient;
Factor in previous gradients. Becomes a weighted average, with higher weight the more recent the gradients

Nesterov Momentum optimization

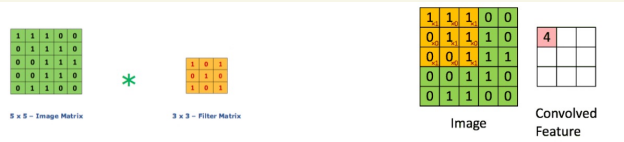
$$v := \beta v + \alpha \nabla L(W - \beta v)$$
$$W := W - v$$

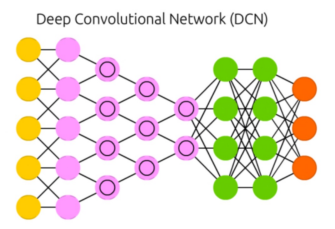
(Prior gradients; current gradient loss)

Convolutional NN

So far; layers were fully connected
Now; Transform large input into smaller set of inputs to Neural Network!

Filtering





Dimensionality Reduction

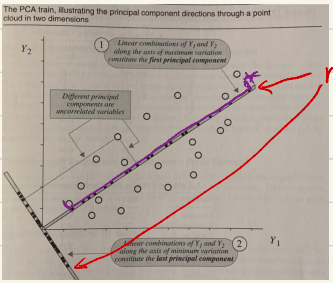
How to "throw something away"
Drawing 3D in 2D is actually a projection, which offers us a reduction of dimensions!

PCA

Principle Component Analysis
1. To represent data in a useful way
2. See if data dimensions can be reduced

!No class Labels!

We prefer features with high variance



new basis axes
1 axis for each feature
axes are Principle Components
starting with the direction of the highest variance*

Scores

The transformed values in new dimensions

$a_1^T X = a_{11}X_1 + a_{12}X_2 + \ldots + a_{1p}X_p$

↑ first column vector of A

*$\|a_1\| = 1$

$A^T X = (X^T A)^T$

X's columns are feature vectors

Find max Var

We maximise $Var(a_1^T X) = a_1^T S a_1$ ← sample variance

under constraint $a_1^T a_1 = 1$*

Lagrange

$F(a_1) = a_1^T S a_1 - \lambda_1(a_1^T a_1 - 1)$

Partial Derivative: $\frac{\partial F}{\partial a_1} = 2Sa_1 - 2\lambda_1 a_1 = 2(S - \lambda_1 I)a_1$

$(S - \lambda_1 I)a_1 = 0 \Rightarrow Sa_1 = \lambda_1 a_1$

↳ $Var(a_1^T X) = a_1^T S a_1 = a_1^T \lambda_1 a_1 = \lambda_1$

a_1 is an eigenvector of S with eigenvalue λ_1

Solution; we choose a_1 with largest λ_1

Lagrange 2

maximise $\text{Var}(a_2^T X) = a_2^T S a_2$

under constraint $a_2^T a_2 = 1$ (non-scaled features) and $a_1^T a_2 = 0$ (perpendicular)

$$F(a_2) = a_2^T S a_2 - \lambda_1 (a_2^T a_2 - 1) - \mu (a_1^T a_2)$$

$$\frac{\partial F}{\partial a_2} = 2Sa_2 - 2\lambda_2 a_2 - \mu a_1$$

Set the partial derivatives to 0

$$0 = 2Sa_2 - 2\lambda_2 a_2 - \mu a_1$$

Left-multiply this by a_1^T:

$$\begin{aligned} 0 &= 2a_1^T S a_2 - 2\lambda_2 a_1^T a_2 - \mu a_1^T a_1 \\ &= a_1^T \lambda_1 a_2 - 0 - \mu \cdot 1 \\ &= \lambda_1 a_1^T a_2 - \mu \\ &= \lambda_1 0 - \mu \qquad \Rightarrow \mu = 0 \end{aligned}$$

Since $\mu = 0$, we just need the next biggest eigenvalue as λ_2

By induction we can prove that we just need to use the eigenvalues in desc order ♡

eigendecomposition diagonalisation

Λ is D with $\lambda_1 \geq \dots \geq \lambda_p \geq 0$

now: $S = A \Lambda A^T$

and now: $A = [a_1, a_2, \dots, a_p]$ is P

Total Variance

Sum of variances $\sum_{i=1}^{p} \lambda_i$

Explained Variance

For k_{th} principle component: $\frac{\lambda_k}{\sum_{i=1}^{p} \lambda_i}$

Variance / explained variance can be plotted (cum. we can keep m% of var)