Linear Regression Presentation:

- Imagine you are tryin to predict the $ of a 🎁 based on its size....
- How do we find the best way to make this prediction?
- At it's core:
  - LN helps us find relationship between an input & an output
  - It does this by fitting a straight line through data points!
  - This line is defined by a function $f(x)$
  - $f(x)$ represents our prediction model.

LAME!

Neural Network

Input Layer

- Neurons in input layer directly output the input values, no activation function

Hidden Layers

- Must use non-linear activation function a useful ANN
- Famous: Sigmoid, tanh, ReLU

Output Layer

- Depend on what our ANN is used for:
  - Regression Problems → Linear Function
  - Binary Classification → Sigmoid Function → $\frac{e^z}{1+e^z}$
  - Multi-class classification → Softmax Function → $p(y=i|z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

Weights

- In a fully connected feed forward Neural Network, each neuron in one layer is connected to every neuron in the next layer. These connections are represented by weights, which determine the strength of the influence of one neuron on another.

Forward Pass

∀ sample (x,y)

- Present x at input Layer
- Compute $z^{(l)}$ & $a^{(l)}$ for all layers
- Compute output & Loss function

1 Sum or Avg loss of all samples = total Loss

Training

- Train Parameters = Weights & Biases

Process

- Initialize W & b randomly for all layers
- Choose a α
- Repeat until convergence or Stopping Condition
  - Update Parameters = ∂d of cost function wrt each W & b

How to find?

$$W_{(t+1)} = W_{(t)} - \alpha \nabla L \qquad b_{(t+1)} = b_{(t)} - \alpha \nabla L$$

Backpropagation

- Backward pass
- Calculate ∂d from output layer towards input layer
- Combine them using chain rule to get how loss changes wrt to each weight

Prevent Overfitting

- Regularization methods
  - Weigh Decay = Squash high magnitude W towards zero
  - Dropout Regularization = Leve some neurons unused in the training
    Use full network for prediction though
  - Early Stop = When test error stops improving, stop training
  - Data Augmentation = Distorts data in a human recognisable way. Shift, rotate.

Convolutional Neural Networks

- Dealing with Fully connected Neural Network mandatorily imposes us
  - to flatten i.e. image
    - leads to loss of spatial information
    - And we need a huge amount of parameters → may lead to overfit
- CNN works better for this

Process

- Check for certain features in different image patches
- Create Hierchy of features
  1. Detect low level features
  2. Extract mid level features
  3. Detect high level features

Extended Process

1 Convolution layers which extract features from low-level to high-level

1 then flatten output & apply fully connected NN to classify

1 So network learn non-linear combs of high level features

Convolutional Layers

- The core of CNN are filtering layers & pooling layers
- Where filtering layers:
  - Apply filters to input data performing feature extraction
- While Pooling layers:
  - Perform downsampling of the data
    - Most common method is max-pooling
    - Can emphasis dominant feature

Filters, Padding, Stride

- A filter is a smaller matrix than input matrix that slides on it by a certain stride (# of steps). Creating a convolved feature
  - Multiply overlapping values & sum them

thank to this multiple layer we are then able to have a better classification!

Ensemble Methods

- In machine learning no single model can always perform best on every problem. Some models overfit, others underfit. What if we could combine multiple models to balance out their weaknesses & amplify their strengths?
- Well with Ensemble Methods we can!

two types

Homogeneous = ≠ types of classifiers trained on the same data (diverse predictors)

Heterogeneous = same types of classifiers trained on different data (bootstrapping)

Aim = Introduce diversity!

Why Ensemble?

- Can perform better than any of its base learners
- Under condition that base-learners are completely independent.

Aim of ≠ Groups

- One group aims to reduce Variance another to reduce Bias

Reduce Variance

Bagging → Bootstrap Aggregation

- Make bootstrapped subset of the training data
- Train a separate model on each subset
- Train can be done in parallel

Prediction: Combination of predictions of all models! AVG

Classification: Use voting!! Hard Voting

Soft & Hard Voting

Hard: Each classifier makes a prediction & Majority vote wins

Soft: Requires estimates $p(k|x)$, final pred is class with highest prob

Evaluation: Out-of-Bag Evaluation

Bagging = Bootstrapped data = Sampling with Replacement

The non sampled points will be use for validation

Can be replaced by a validation set!

Error = Avg error between models that have those as OOB

Random Forest:

- Essentially a Bagging Decision tree method with modified split crit.
  - Make bootstrapped data sets.
  - Train a full tree on this data sets
  - Before finding each split:
    - Choose randomly k features
    - Consider only those for split
  - Output = Aggregation of all trees
- Really nice because not many hyperparams:
  - m = # bootstrapped dataset
  - k = # of randomly selected features.

* Picks best out-come

Extra Trees: Extremely Randomized Trees

- Fit each D.T on whole dataset
- Still randomly sample k features
- Randomly selects a cut-point for all features & picks best

Reduce Bias

Boosting:

Slightly guesses better than random!

Sequence of weak learners combined into a stronger one!

Sequential !!!!!

How it works:

Train a weak learner → Get training Error → train next based on error

Prediction : Combination of all weak learners trained

AdaBoosting → most commonly used for classification!

Process :

Equal weights for all training samples $w_i^{(1)} = \frac{1}{N}$

For c=1....M iterations (trees):

- train a model $h_c(x)$
- Calculate total Error on training set $\epsilon_c = \sum_{i, h_c(x) \neq y_i} w_i^{(c)}$ (where pred was different)
- Based on calculate amount of say : $\alpha_c = \ln\left(\frac{1-\epsilon_c}{\epsilon_c}\right)$ (performance of each learner)
- Update weights:
  - Increase misclassified: $w_i^{(c)} = w_i^{(c)} e^{\alpha_c}$ (model has to focus more on this!!)
  - Correct classifications:

    Unchanged or decrease

    $w_i^{(c)} = w_i^{(c)}$
- Normalize weights to prevent instability :

$$w_i^{(c+1)} = \frac{w_i^{(c)}}{\sum_{j=1}^{n} w_j^{(c)}}$$

Decision trees

- Supervised data
- Great interpretability due to its if-else structure
- Each split focuses on a single feature
- Process:

  Ask question recursively to create a tree
- trees can be used both for regression & classification
  - regression → mean
  - classification → prediction : output label
- Most commonly a split is binary but can also be multi-way

Training

- A balance tree is a good tree
- To choose the node splitting condition:
  - Try all possible splits for current node

- Use Heuristic to measure how good each split is
- Pick the best split!

Evaluating a split

Regression: Calculate mean $\bar{y}$ resulting region → $\bar{y}_{R1}$, $\bar{y}_{R2}$

$$RSS = \sum_{i:x_i \in R_1} (y_i - \bar{y}_{R1})^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y}_{R2})^2$$

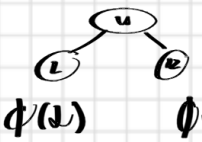
Pick the split with the minimum RSS!

Classification: Concept of **Impurity**

→ A measure of how mixed data is wrt class members

Combine the impurity using their weighted avg

$$p_L \phi(L) + p_R \phi_R$$



$p_L = \frac{N_L}{N_u}$ $p_R = \frac{N_R}{N_u}$

split that minimizes this is the best split!

Impurity Function

- can be defined as a function of posterior probs.

$\phi(p_0, p_1, \ldots, p_{K-1})$ where estimated p.b. $p_k = \hat{p}(k|t) = \frac{N_k}{N_t}$

- Some of the most used impurity functions:

Gini Impurity: $\phi(t_i) = 1 - \sum_{k=1}^{K} p_k^2$ ↔ **Gini Index**

Entropy: $\phi(t_i) = -\sum_{k=1}^{K} p_k \log(p_k)$

PCA

In data science datasets are always high dimensional

So we wish to represent data in a useful way

We try to see if data can be reduced to lower dimensions!

- Unsupervised method for such things
- PCA finds a pattern that is:

  What directions in our dataset spread the datapoints the most?

  Seek the direction of maximal variance

Process

- Find new basis vectors! (Principal Components)
  - Build an orthogonal basis where new basis vectors explain the direction of greatest variance in data.
- Project to lower dimensions!
  - First k principal components span a k-dimensional subspace
  - It is the best k-dim view of data

Feature Example

- Consider set of p features $X_1, \ldots, X_p$
- PCA = Gives new set of the p features, Principal Components, each being a linear comb of the original p features!
- Every principal component is going to capture a $\neq$ amount of variance

Imagine that we have n data points, each with p features.

Data point $i \longrightarrow x_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix}$, we can define $\bar{x} = \sum_{i=1}^{n} x_i / n$

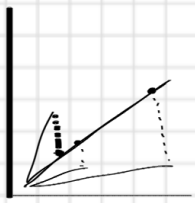
Goal: Find direction of maximal variance

Linear Algebra Problem

- In order to solve this we need to transform it into a linear algebra problem
  - Imagine that x_i = one data point
  - a_i = some vector

} When we dot two vectors, one gets projected onto the other

- We can a_i, a col vector, our first principal component $a_i = \begin{pmatrix} a_{i1} \\ \vdots \\ a_{ip} \end{pmatrix}$



- If a_1 points in the direction of maximal variance

our goal is then to maximize $\text{Var}(a_1^T x)$

Maximization turns into $\text{Var}(a_1 x) = a_1^T S a_1$ ← Data Covariance matrix

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T$$

Ensures a_1 is a unit vector

- We can maximize this to constraint $a_1^T a_1 = 1$ by Lagrange multipliers:

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

$= 0$

Note:

Principal component is an eigenvector of the data covariance matrix, representing direction of maximal variance in dataset

So we, max wrt a_1

$$\frac{\partial F}{\partial a_1} = 2 S a_1 - 2 \lambda_1 a_1$$

$$S a_1 - \lambda_1 a_1 = 0$$

$$S a_1 = \lambda_1 a_1$$

Which means that a_1 is the eigenvector corresponding to λ_1 eigenvalue

and max wrt λ_1

we recall ∞...

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

$$= a_1^T \lambda_1 a_1 - \lambda_1 a_1^T a_1 + \lambda_1$$

$$= \lambda_1$$

Which leads to choosing λ_1 to be largest eigenvalue

To find the second + other components we will add 0 for each new constraint we impose.

Eigendecomposition

- In PCA we consider eigendecomposition of S given by: $S = A \Lambda A^T$

- WHERE Λ IS THE DIAGONAL MATRIX WITH EIGENVALUES $\lambda_1 > \ldots > \lambda_p > 0$
- AND WHERE $A = P_{CA}$ ORTHOGONAL MATRIX WHERE CVS = CORRESPONDING EIGENVECTORS

ALL OF THIS IS TO SAY THAT PCA CHOOSES EIGENDECOMPOSITION THAT ORDERS THE EIGENVECTORS ACCORDING TO DECREASING EIGENVALUES

SO A PRINCIPAL COMPONENT CAN BE REPRESENTED BY:

$$z_k^t X = a_{k1}X_1 + a_{k2}X_2 + \ldots + a_{kp}X_p$$

(COEFFICIENTS ALSO CALLED LOADINGS)

PCA CREATES UNCORRELATED FEATURES THAT ARE LINEAR COMBINATIONS OF EXISTING FEATURES.

RECONSTRUCTION OF X

BY USING ALL PC'S WE CAN RECONSTRUCT EXACTLY A DATA POINT X

BY USING ONLY THE FIRST M PRINCIPAL COMPONENTS WE GET AN APPROXIMATION

DATA CENTERING

HELPS US MINIMIZE APPROXIMATION ERROR

MEANS MOVING COORDINATE SYSTEM TO THE MEAN $\bar{x}$

CHOOSING # OF PC

- THERE ARE MANY HEURISTICS ON HOW TO CHOOSE A GOOD # OF THEM
- TREAT THEM AS TUNING PARAMETERS
- WE COULD I.E HELP US WITH A SCREE PLOT EXPLAINS:
  - HOW MUCH VARIANCE IS EXPLAINED BY EACH PC
  - CUMULATIVE IS MORE INTUITIVE

STANDARDIZATION

- WE NEED TO STANDARDIZE OTHERWISE FEATURES WITH LARGER VARIANCES WOULD DOMINATE THE PRINCIPAL COMPONENTS BASED ON ITS VARIANCE.
- SO BY CENTERING & STANDARDIZING WE GET:

  VARIABLES WITH MEAN 0 & VARIANCE 1

BAYES THEOREM

- THE BAYES THEOREM ESTIMATES THE POSTERIOR PROB OF CLASSES GIVEN OBSERVED FEATURES

$$P(C_K|X) = \frac{P(X|C_K)P(C_K)}{P(X)}$$

- WITH THE AIM OF MAKING AS FEW MISCLASSIFICATION AS POSSIBLE
- IN ORDER TO DO THIS WE NEED TO DEFINE A DECISION RULE:
  - DEFINES WHEN DO WE CLASSIFY A POINT AS C_K?
  - ANY DECISION RULE DIVIDES THE FEATURE SPACE INTO DECISION REGIONS:

    $R_k \quad k \in 1, \ldots, K$

- We said that the aim = as few misclassification as possible

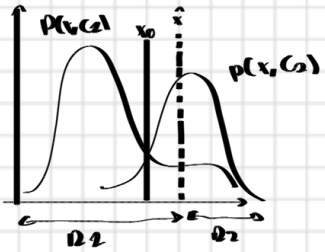
$$\text{minimize}[p(\text{mistake})] = p(X \in R_1, C_2) + p(X \in R_2, C_1)$$

$$= \int_{R_1} p(X, C_2)\,dX + \int_{R_2} p(X, C_1)\,dx$$

$$\int_{R_1} p(x)\,p(C_2|x) + \int p(x)\,p(C_1|x)$$

We know that: $p(X, C_k) = P(X)\,p(C_k|X)$

Limit by assigning class with heighest $p(C_k|x)$



- Just developed classifier is called **Bayes classifier**
  - We classify K with heighest prob:

$$d(x) = \text{ARGMAX}\, P(y=g|x)$$

- **Not** all mistakes are equal though!
  - Dealing with cancer:
    - * False positive causes stress to patient
    - * False negative may lead to death

Goal = Limit certain mistakes

- Loss & Loss matrix:

$$\text{true}\begin{matrix} \\ C \\ N \end{matrix}\begin{matrix} \text{pred} \\ C \quad N \end{matrix}\begin{bmatrix} 0 & 1000 \\ 1 & 0 \end{bmatrix}$$

We assign a loss = $L_{kj} > 0$ which depends on the true class

- New way to express our goal
  - Minimize the expected loss!

$$E[L] = \sum_k \sum_j \int_{R_j} L_{kj}\,p(X, C_k)\,dx$$

$$E[L] = \sum_k \sum_j \int_{R_j} L_{kj}\,p(x)\,p(C_k|x)\,dx$$

} Assign a new X to class j that minimizes $\sum_k L_{kj}\,P(C_k|X)$

SVM

- It is Supervised Learning Method
- Can be used both for:

  Classification - Regression

- Intuition:

  Researches the best decision boundary for training datasets

- If C_n are linearly separable → Best decision boundary is line

  meaning a hyperplane with the maximum margin (widest street)

  from the closest samples of both classes

- Recap linear classifier:

$$y(x) = w \cdot x + b \longrightarrow y(x) = 0$$

(decision boundary)

If $y(x_i) \geq 0$ then sample $i = +1$

If $y(x_i) < 0$ then sample $i = -1$

- THE RELEVANT SAMPLES IN ORDER TO MAKE THE WIDEST STREET

  • ARE THE ONLY SAMPLES THAT CONTRIBUTE TO DECISION BOUNDARY! SUPPORT VECTORS!!

How to find Support Vectors?

- SVM FINDS THEM DURING TRAINING
- USES THEM TO CLASSIFY NEW SAMPLES!

Hard Margin VS Soft Margin

- USED WHEN TWO CLASSES ARE LINEARLY SEPARABLE

INSTEAD OF $y(x) = W \cdot X + b$:

$\alpha_i \geq 0$ WHEN $\alpha_i > 0$ SUPPORT VECTOR!

$$y(x) = \sum_{i=1}^{n} \alpha_i y_i (X_i \cdot X) + b$$

TRAINING SAMPLE

UNSEEN VALUE

DOT PRODUCT → EVERY TR SAMPLE & X

- AFTER TRAINING WE CAN KEEP ONLY SUPPORT VECTORS AND THEIR ASSOCIATED α_i's FOR PREDICTIONS

DISCARD REST OF THE TRAINING SET!

- RULE: NO TR SAMPLES ARE ALLOWED ON THE WRONG SIDE, NOT EVEN ON THE MARGIN!

• to MAXIMIZE WIDTH & PREDICT:

① TRAIN MODEL, FIND α_i ∀ TRAINING SAMPLE

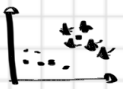
$$y(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + b$$

② KEEP ONLY SUPPORT VECTORS & THEIR α_i

WHERE $\alpha_i > 0$

③ USE SUPPORT VECTORS & α_i TO CLASSIFY NE SAMPLES

WHAT HAPPENS IF CLASS CONDITIONAL DISTRIBUTION OVERLAP?



- IT IS IMPOSSIBLE TO FIT A HARD MARGIN SVM FOR THIS
- WE RELAX THE CONSTRAINT
- SOME TR SAMPLE CAN FALL ON WRONG SIDE
- WE PENALIZE IT!
- OBJECTIVE:

MAXIMIZE STREET'S WIDTH WHILE MINIMIZING THE PENALTY

- END UP WITH THE FOLLOWING:

$$y(x) = \sum_{i}^{n} \alpha_i y_i (x_i \cdot x) + b \qquad 0 \leq \alpha_i \leq C$$

BOUND TO A REGULARIZATION PARAM

- C MAKES A TRADEOFF BETWEEN PENALTY AND MARGIN MAXIMIZATION!

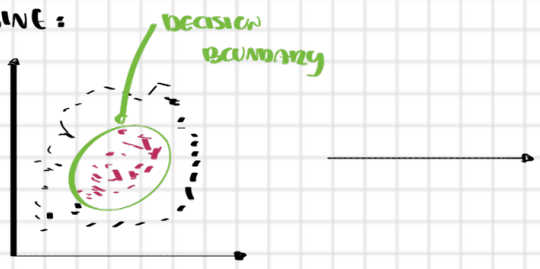
What makes SVM interesting?

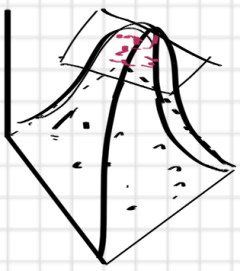
- It's the Dot Product
- Let's think about NON-LINEAR DECISION BOUNDARY

IMPORTANT

SOMETIMES WE CAN MAP FEATURES FROM THE ORIGINAL DIMENSION SPACE TO A HIGHER DIMENSION WHERE THE CLASSIFICATION PROBLEM IS LINEAR

- IMAGINE:





A LINEAR DECISION BOUNDARY IN THE HIGH DIMENSIONAL FEATURE SPACE CAN REPRESENT A NON-LINEAR BOUNDARY IN THE ORIGINAL SPACE

- TRANSFORMATIONS CAN ADD MANY NEW NON-LINEAR FEATURES TO HOPEFULLY ANGLE A GOOD BOUNDARY!
  - IT IS WHAT GIVES US THE POSSIBILITY TO MAP FROM A LOWER TO HIGHER DIM.

HENCE, PREDICTION MODEL BECOMES

$$y(x) = \sum_{i=1}^{n} \alpha_i y_i (\phi(x_i) \cdot \phi(x)) + b$$

- WE UNFORTUNATELY KNOW THAT:

HIGHER DIMENSION = HIGHER COMPUTATIONAL POWER REQUIRED

KERNEL TRICK!

- DEFINES A KERNEL FUNCTION THAT COMPUTES THE DOT PRODUCT OF THE TRANSFORMED INPUT WITH A TIME COMPLEXITY BASED ON THE ORIGINAL SPACE
- SO NOW PREDICTION MODEL:

$$y(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

- THE BEST THING IS THAT:

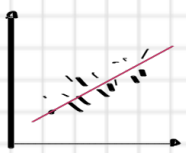
WE CAN TRANSFORM FEATURE VECTORS EVEN TO INFINITE DIMS & STILL DON'T BOTHER ABOUT NEEDING MORE COMPUTATIONAL POWER

- WE COULDN'T NOT KNOW $\phi(x)$

WE ONLY NEED $K(x, z)$

LINEAR REGRESSION

- IMAGINE YOU ARE TRYING TO PREDICT THE $ OF A 🍉 BASED ON ITS SIZE....
- HOW DO WE FIND THE BEST WAY TO MAKE THIS PREDICTION?
- AT IT'S CORE:
  - LR HELPS US FIND RELATIONSHIP BETWEEN AN INPUT & AN OUTPUT
  - IT DOES THIS BY FITTING A STRAIGHT LINE THROUGH DATA POINTS!
  - THIS LINE IS DEFINED BY A FUNCTION $f(x)$
  - $f(x)$ REPRESENTS OUR PREDICTION MODEL.



↳ WE ASSUME A FUNCTIONAL RELATIONSHIP BETWEEN X & y ⟶ $y = f(x) + \varepsilon$

↳ IN LINEAR REGRESSION WE TAKE $f(x)$ TO BE LINEAR ⟶ $y = \beta_0 + \beta_1 x_1 + \varepsilon$

↳ MORE GENERALLY WITH p FEATURES & AN INTERCEPT TERM THE MODEL ⟶ $y = X\beta + \varepsilon$ (DESIGN MATRIX!)

DUMMY VARIABLES

VARIABLES ARE NOT ALWAYS CONTINUOUS:

- YES/NO 0 or 1

WE THEN INTRODUCE DUMMY VARIABLES

WHERE WE INTRODUCE ONE DUMMY VAR FOR EACH LEVEL A OF FEATURE X

$$\mathbb{1}\{x = A\} = \begin{cases} 1 & x = A \\ 0 & x \neq A \end{cases}$$

(ONE HOT ENCODING)

Complex Functional Relationships

- Variables can also interact with each other
- To handle this we need to account for interaction terms

  defined as X_1X_2 i.e.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \varepsilon$$

NON-LINEAR!!

Trick:

Introduce transformation of feature where we get

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

Note Relation is complex if plotted against original features but linear in new ones.

Estimate Coefficients β

→ Ordinary least squares regression finds β that minimizes the RSS

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y})^2$$

→ More Generally we prefer to fit our model to maximize the model likelihood

- Means: which model params would be most likely to give us the data?

→ Start by defining a likelihood function

$$p_y(y_i|X) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\hat{y})^2}{\sigma^2}}$$

Given that we assume residuals to be N distributed we can use PDF of it

PDF for each error

The likelihood for the entire dataset is then:

$$\prod_{i=1}^{n} p_y(y_i|X) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\hat{y})^2}{\sigma^2}}$$

$$L(\beta, \sigma^2) = -\frac{1}{2}\left(n \log \sigma^2 + \frac{1}{\sigma^2}\sum_{i=1}^{n} (y_i - \hat{y})^2\right)$$

Prediction

- Once model is ready:

  Given a set of features $x_i = (x_{i1}, \ldots, x_{ip})$

  We can predict y_i

- We predict by saying that:

  y_i = value of regression line at x_i

Confidence & Prediction Intervals

- We can interpret coefficients using confidence intervals
- Prediction interval > confidence interval

Don't say, would go overtime

Scrutinizing Model!

- Now that we have our model we should ask ourself:

  Does the model fit well? ⟶ Effectively scrutinize model

- i.e test whether a specif β is 0:

$$\frac{\beta_j - 0}{\hat{SE}(\beta_j)}$$

Evaluate significance of predictor variable!

i.e if of post-boxes

- See if several β's are β_j:

  Creating a simpler copy of the model

$$M_1: y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \varepsilon$$

$$M_0: y = \beta_0 + \beta_1 x_1 + \varepsilon$$

  use F-test:

  measures improvement in RSS in the two models

$$F = \frac{(RSS_{M0} - RSS_{M1})/q}{RSS_{M1} / (n-p-1)}$$

q: # of params dropped

n: total # of observations

  High: Significant Reduction in M_1's RSS

  Low: It is not worth to have such complex model

Machine Learning Project!

In the Project we explore three different:

- Classification Methods
- to classify clothing images!

The three methods we used are:

- Decision Trees
- FFNN

} from scratch

- KNN and Bagged KNN ⟶ implemented with library support

Exploratory Analysis

- At the beginning we did some exploratory analysis.
- Pooling the features distributions to immediately gain some valuable informations
  - We were able to see that each class was equally represented
  - We knew that model would have had an equal representation during learning!
- Given the High dimensionality of the data we scaled the features & applied PCA in order to reduce as much as possible the dimensions!
- A scree plot helped us understand wether how many PC's we needed in order

TO CAPTURE AT LEAST 90% OF THE TOTAL VARIANCE

- WE NEEDED AT LEAST 112 PCs

Decision Trees

- IN ORDER TO EVALUATE THE SPLITS IN A TREE THERE IS THE NEED OF A IMPURITY FUNCTION:

  AS OUR IMPLEMENTATION CHOICE, OUR MODEL WORKS BASED ON: INFORMATION GAIN

  SO WE CHOOSE ENTROPY OVER GINI INDEX

- TRAINED THE MODEL SCALED & REDUCED FEATURES.
- INFORMATION GAIN CRITERION WAS DUE MAINLY TO BETTER HANDLE CATEGORICAL OUTPUTS
  - ENTROPY PROVIDES A MORE INTUITIVE MEASURE OF DISORDER!
- WE ADOPTED THREE STOPPING CRITERIAS:

  MAXIMUM DEPTH: CAPPED AT MAX_DEPTH VAR

  MINIMUM SAMPLES SPLIT: ONLY SPLIT IF CONTAIN AT LEAST A MIN # OF SAMPLES [5]

  HOMOGENEOUS LABELS: DECLARE LEAF IFF ALL SAMPLES HAVE SAME LABEL
- BY A GRID SEARCH WE WERE ABLE TO FIND BEST VALUES FOR OUR HYPER PARAMS:

  MIN_SAMPLES_SPLIT = 2
  MAX_DEPTH = 10
  PCA = 40 COMPONENTS → 80% VARIANCE EXPLAIN
- ENDED UP WITH 97% F1 SCORE & ALMOST THE SAME CONFUSION MATRICES

Feed Forward Neural Networks

KEY ELEMENTS OF OUR IMPLEMENTATION ARE:

EARLY STOPPING: TO PREVENT OVERFITTING, AT MOST 20 EPOCHS WITH NO LOSS IMPROVE ?
DROP-OUT REGULARIZATION: ONLY ON KERAS MODEL
MODEL EVALUATION METHODS: ACCURACY, PLOT LOSS CHANGE, CONFUSION MATRIX PRECISION, RECALL & CLASS

- AFTER NUMEROUS EXPERIMENTS WITH THE MODEL WE DECIDED TO USE A SINGULAR HIDDEN LAYER WITH 64 NEURONS & 2^{ND}-SIGMOID FUNCTION (FOR KERAS)
- WEIGHTS INITIALIZED WITH GLOROT-UNIFORM DISTRIBUTION
- BIASES INITIALIZED TO 0
- α SET TO 0.0001
- EPOCHS TO 1000

SCRATCH MODEL USES ReLU ACTIVATION
KERAS MODEL USES SIGMOID ACTIVATION

WHILE 128 NEURONS FOR THE 'SCRATCH'

- WE SAW THAT MODEL CONSTANTLY IMPROVED DURING SUCH HIGH # OF EPOCHS
- GIVEN HIGH # OF EPOCHS → EARLY STOPPING SET TO [20] & [70] AFTER SEVERAL TRIES =SCRATCH
- RESULTS OF LIBRARY & SCRATCH WERE VERY SIMILAR 83%-86% ACC
- BOTH MODELS STRUGGLED IN CLASSIFING SHIRTS, SOLVABLE BY DATA AUGMENTATION
- 'SCRATCH' → 95% ACC, A BIT OF OVERFITTING
- BY COMPARING CONFUSION MATRIX CAN SEE THE SIMILARITY!

KNN

- I initially implemented a **version of KNN trained on non-reduced features**
  - Used as **bagging** & as **comparison model**!
  - The model chose was model with **k=7** due to high **recall** in recognition of class 4, **shirt**,
  - took the model **301 seconds** to compute folds for **cross-validation**.
- Then started working with **PCA-reduced features**
  - **Significantly faster**, took only **3.4s** to compute folds
  - W between a range of **k [4,8]** trained models, **choosing with k=7** reported higher accuracy & better metrics of **precision, recall, f1** for most classes

Especially class 4 ← (why we picked it)

Bagged KNN

Using it to reduce variance hence overfitting

- Made sure that tradeoff between **Variance & bias** was taken into account
  - Seemed that a good tradeoff was given by models with **k=4 or k=5**
- We got greater acc for lower k's **as expected**

  Bagging & variance = lower k's have higher variance

  Then began analyzing the matrics that models with **k≤5** are to analyze

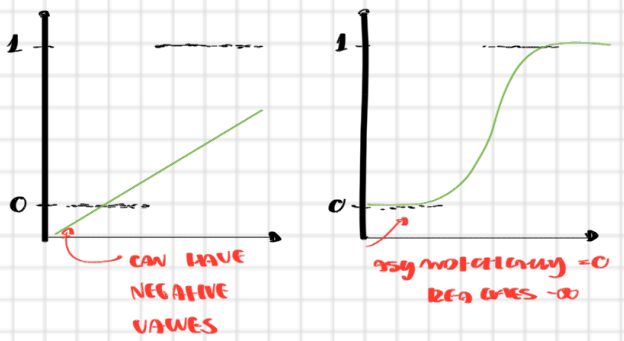
Testing!

So testing normal KNN PCA-applied gives us **82.5% acc!**

While out of all models with **k≤5**, **k=3** is the best with acc **83%**

Logistic Regression

- In many applications we are interested in predicting **what class a datapoint belongs to:**
- Predict Binary outcomes → Solve a classification problem
- Let's imagine we want to classify a certain point as 0 or 1 based on some features:



$$P(X) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$ → Returns P(X) in between 0-1

- Sometimes we like to think about likelihoods **not in term of P(X) but as odds:**

$$\text{odds} = \frac{P(X)}{1-P(X)} = e^{\beta_0+\beta_1 x}$$

What are the odds that this happens rather than the other?

They are positive real if $\text{odds}(A) > \text{odds}(B)$ iff $P(A) > P(B)$

So if we take the log ⟶ $\log(\text{odds}) = \beta_0 + \beta_1 x$ ← A one unit change is a change of β_1 in logit

Multiple Features!

- Generalizing to general feature :

  - Logistic Regression models conditional probs of $y=1$ given $X_1, \dots, X_p$

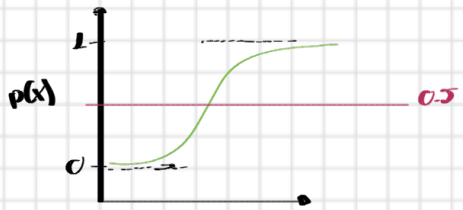
$$P(y_i=1 \mid X_i=x) = \frac{e^{x\beta}}{1+e^{x\beta}}$$

$$\text{odds} = \frac{P(y_i=1 \mid X_i=x)}{1-P(y_i=1 \mid X_i=x)} = e^{x\beta}$$

$$\log(\text{odds}) = x\beta$$

Prediction

- to make $\hat{y}$ to the class with highest probability
  - take into account a threshold ⟶ 0.5 default



$$\hat{y}_i = \begin{cases} 1, & P(y_i=1 \mid x_i) > 0.5 \\ 0, & P(y_i=1 \mid x_i) \leq 0.5 \end{cases}$$

Evaluation

	$\hat{y}=1$	$\hat{y}=0$
$y=1$	50	10
$y=0$	10	30

Accuracy, Precision, Recall

Test Error Rate - prob of $\hat{y}_i \neq y_i$

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

Estimating Coefficients

- Parameters are estimated by maximum Likelihood
- Likelihood Function → Binomial Likelihood

$$L(\beta, y) = \prod_{i:y_i=1} p(x_i) \prod_{j:y=0} 1-p(x_j)$$

$$\log(L(\beta, y)) = \sum_{i=1}^{n} (y_i \log(p_{x_i}) - (1-y_i)\log(1-p(x_i)))$$

Multiclass Classification

- Softmax Regression

y_i $C_k = 1, \dots, k$ Use one-hot-encoding $y_i = (0, 1, \dots, 0)$

clearly follows a multinomial distribution with $(p_{i1}, \dots, p_{ik})$

So now we have $k-1$ log-odds

$$P(y=k \mid X=x) = \frac{e^{x\beta^k}}{\sum_{c=1}^{k} e^{x\beta^c}}$$

THEN CHOOSES $\hat{y}$ THAT BELONGS TO CLASS WITH HIGHEST PROB

$$\hat{y} = \text{argmax}\, \hat{P}(y=k \mid X=x)$$

PREDICT BIAS & VARIANCE

- WHEN WE TALK ABOUT PREDICTION MODELS,

  AIM: PREDICT UNSEEN DATAPOINTS WELL!!

- RECALL LINEAR CLASSIFIER:

$$y = f(x) + \varepsilon$$

IMAGINE THAT WE HAVE TWO MODELS TO FIT THE DATA

$$\hat{f}_1(x) \text{ & } \hat{f}_2(x)$$

AND WE ARE NOW ASKED TO PREDICT A UNSEEN DATA POINT (x_0, y_0)

HOW DO WE EVALUATE WHICH MODEL IS THE BEST?

NOTE:
LATER WE ARE GOING TO BE ASKED TO PREDICT n UNSEEN (x_i, y_i) $i = 1 \ldots n$

- WITH ACCES TO TRAINING DATA WE WANT:

$$E[\text{TEST MSE}] = E[(y_0 - \hat{f}(x_0))^2]$$ → AS SMALL AS POSSIBLE!

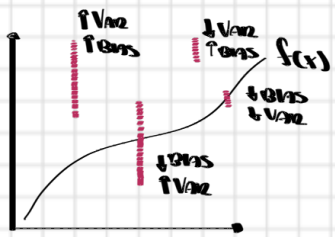
- IMAGINE TRAING A LEARNER $\hat{f}$ & PREDICTING A POINT (x_0, y_0)

$$E[\text{TEST MSE}] = E\left(\hat{f}(x_0) - E\hat{f}(x_0)\right)^2 + \left(E\hat{f}(x_0) - f(x_0)\right)^2 + \text{Var}(\varepsilon)$$

VARIANCE: SENSITIVITY OF DATA

BIAS: ERROR INTRODUCED BY APPROXIMATING A REAL LIFE PROBLEM

RECALL:



= PREDS MADE BY ≠ INSTANCES OF $\hat{f}$

GIVEN THAT ALL THREE ARE NON-NEGATIVE
IF ANY IS LARGE, MSE IS LARGE

WE WANT LOW BIAS & LOW VARIANCE !!!

RECALLING FORMULA:

$$E[\text{TEST MSE}] = \underbrace{E\left(\hat{f}(x_0) - E\hat{f}(x_0)\right)^2 + \left(E\hat{f}(x_0) - f(x_0)\right)^2}_{\text{REDUCIBLE ERROR}} + \underbrace{\text{Var}(\varepsilon)}_{\text{IRREDUCIBLE ERROR}}$$

LOWER IT BY USING ESTIMATOR $\hat{f}$ WITH BOTH:

LOW BIAS
LOW VARIANCE

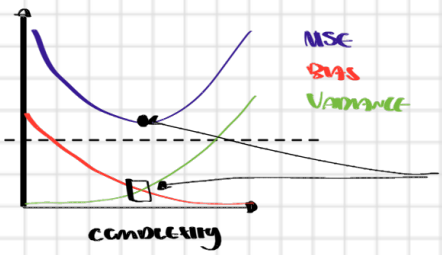
COMPLEXITY

INTERPOLATES DATA
FITS A STRAIGHT LINE
} THINK ABOUT THEIR BIAS & VARIANCE

= HAS LOW BIAS, HAS HIGH VARIANCE

= HAS HIGH BIAS, LOW VARIANCE

DIFFICULT TO FIND ONE THAT HAS BOTH LOW!!!



SWEET SPOT - TRADEOFF !!!

FINAL FORMULA

- TO BE A GREAT MODEL FIT, WE WANNA DO WELL WITH ALL POINTS NOT JUST ONE!

  WE WANT $E[(\text{test MSE})]$ = GOOD ACROSS ALL POSSIBLE VALUES OF X_0 SO...

  SO WE END UP WITH:

$$\text{MSE} = E[\text{VAR}(\hat{f}(x_0))] + E[\text{BIAS}(\hat{f}(x_0))]^2 + \text{VAR}(\varepsilon)$$

CLUSTERING

- CLUSTERING ANALYSIS IS ALL ABOUT DISCOVERING GROUPING IN THE DATA
  - UNSUPERVISED METHODS
  - GIVEN 'UNLABELED DATA' → GROUP DATA INTO SOME # OF CLUSTERS
  - IT IS VITAL THE CHOICE OF K
- WE SEEK TO PARTITION DATA INTO K CLUSTERS $C_1, \ldots, C_K$

  ASSUMING THAT EACH DATA POINT CAN ONLY BELONG TO A SINGLE CLUSTER
- WE SAY THAT DATAPOINTS IN THE SAME CLUSTER ARE SIMILAR
- 2 METHODS COVERED:

  K-MEANS CLUSTERING
  HIERARCHICAL CLUSTERING

K-MEANS CLUSTERING

- FIRST CHOOSE K
- ALGO THEN PARTITIONS DATA INTO K CLUSTERS $C_1, \ldots, C_K$

  WHERE: ALL DATAPOINTS BELONG TO EXACTLY ONE CLUSTER & ALL ARE NON-EMPTY.

WE SEEK A PARTITION THAT MINIMIZES WITHIN-CLUSTER VARIATION

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \sum_{k=1}^{K} W(C_k)$$

SIMILARITY = DISTANCE

INSTEAD USE CENTROIDS!

SO...

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|^2$$

$$W_{C_k}(r_k) = \sum_{i : i \in C_k} \|x_i - r_k\|^2.$$

IF WE KNOW WHICH OBS $\in C_k$ THEN r_k = CLUSTER MEAN

$$r_k = \frac{1}{|C_k|} \sum_{i : i \in C_k} x_i$$

THE ALGO may NOT FIND AN OPTIMAL PARTITION OF DATA

STEPS

1. RANDOMLY INITIALIZE $r_1, \ldots, r_K$
2. ASSIGN ALL DATA POINTS TO CLOSEST r_k
3. CALCULATE MEAN & MOVE CENTROID

} REPEAT UNTIL CENTRES STOP MOVIN'

Careful

- Choice of k heavily impact which data points are clustered together
- Anything that impacts distances can impact clustering
  - High Dimensionality
  - Outliers
  - Scale of Variables

Silhouette

- We can use Silhouette score to evaluate a partition

We calculate $a(x_i)$ & $b(x_i)$

$\forall\ x_i \in C_k$

$$a(x_i) = \frac{1}{|C_k|-1} \sum_{x_j \in C_k, j \neq k} d(x_i, x_j)$$ Avg distance between x_i & all other points in the partition

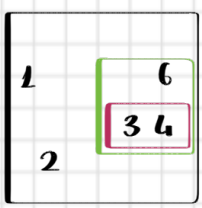
$$b(x_i) = \min_{j \neq k} \frac{1}{|C_j|} \sum_{x_l \in C_j} d(x_i, x_l)$$ Avg distance within x_i & all points in closest cluster

$$S(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))} \qquad -1 \leq S(x_i) \leq 1$$ High is better

Hierarchical Methods

- Agglomerative vs Divisive strategies
  - Agglomerative = From k → $k-1$ cluster
  - Divisive = From k → $k+1$ clusters
- It is all based on dissimilarity
  - The higher the better

Agglomerative

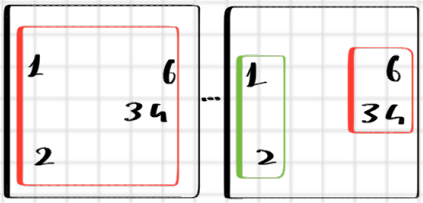


Start with every obs being a separate cluster

Repeatedly merge two cluster

We merge the once with smallest dissimilarity

Divisive



Works the opposite direction

Start with all obs in one cluster

Recursively split one cluster

Measure Dissimilarity Between Clusters

- Dissimilarity based on distance

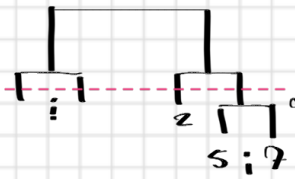
Single-Link Clustering $D(C_i, C_j) = \min \{d(x,y) \mid x \in C_i, y \in C_j\}$ ← shortest d within any pair of point

Complete-Link Clustering $D(C_i, C_j) = \max \{d(x,y) \mid x \in C_i, y \in C_j\}$ → Largest d within any pair of pts

Group-Average Clustering $D(C_i, C_j) = \text{mean} \{d(x,y) \mid x \in C_i, y \in C_j\}$ → Avg d within all pairs of pts

Dendograms

- Hierarchical clustering gives us dendograms



Moving bottom up clusters fuse

While moving bottom down clusters get divided

We can use horizontal cuts to partition data

All nodes below intersection are grouped together

Performance Metric

- Up until now we've understood that the common goal in prediction models is

minimizing the misclassification rate

- When introducing Loss, we emphasized the fact that not all errors are equal
- Given that ≠ task we require ≠ performance metrics:

  We have many of them!

- Metrics are usually evaluated on a test set!

Confusion Matrix, Precision, Recall

- At the ❤ of many performance matrix there is the confusion matrix.

	$\hat{y}=1$	$\hat{y}=0$
$y=1$	50	10
$y=0$	10	30

Accuracy = proportion of correct classification

(50+30) / 100 = 0.8 acc

Misclassification rate = Proportion of misclassification

(10+10) / 100 = 0.2 mr

- Whenever we deal with binary matrix:

  we can easily write the following scheme

	$\hat{y}=1$	$\hat{y}=0$
$y=1$	TP	FN
$y=0$	FP	TN

We can use matrix entries to construct a range of informative measures!

Precision = # of **true positives** out of **predicted pos.**

tp / (tp+fp) → 50 / (50+10)

How precise is our model?

Recall = # of **true positives** out of **positive cases**

tp / (tp+fn) → 50 / (50+0)

How good is it at detecting cases?

- Tasks have ≠ aims:

  i.e It is not important to classify an C as Cs, but when do we **need to right!!** (shoes) **high precision**

  i.e It is important that we classify an C as Cs, some **misclassifications are ok!** (cancer) **high recall**

WE TEND TO TRAIN A CLASSIFIER THAT HAS BOTH HIGH!

- IT IS VERY IMPORTANT TO LAYOUT THAT:

  OUR CHOICES CAN IMPACT THE CONFUSION MATRIX! i.e (LOGISTIC REGRESSION THRESHOLD)

F1-Score

- WE SAID THAT WE WANT BOTH PRECISION & RECALL TO BE HIGH

  WE CAN INTRODUCE F1-SCORE

- IT IS THE HARMONIC MEAN BETWEEN PRECISION & RECALL!

- GIVEN THAT HARMONIC MEAN PUNISHES SINGLE LOW NUMBERS MORE THAN ARITHMETIC DOES

  TO HAVE HIGH F1-SCORE → NEED HIGH PRECISION & RECALL

$$F1 = 2\frac{P \cdot R}{P+R}$$

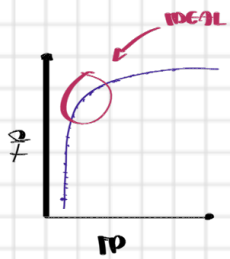
- FOR K CLASSES THERE ARE K SETS OF PRECISION & RECALL SO ALSO F1-SCORES

  WE CAN GET MACRO-F1 SCORE → AVG ACROSS CLASSES

$$\frac{1}{K}\sum_{k=1}^{K} F1_k$$

ROC CURVE

- USED FOR BINARY CLASSIFICATION
- PLOTS TP RATE (RECALL) AGAINST FP RATE $\frac{FP}{FP+TN}$



- ONE POINT = ONE CLASSIFIER
- ONE CURVE = ONE SET OF CLASSIFIERS
- LEFT TO RIGHT = CHANGE THRESHOLD

CLASSIFIER GOOD?

- IS MODEL GOOD?
- DO I WANT A SIMPLER ONE?
- BEST-PERFORMING MODEL CAN STILL BE BAD!

GRADIENT DESCENT

- IT IS AN OPTIMIZATION ALGORITHM
- MEANS MIN OR MAX AN OBJECTIVE FUNCTION WRT SOME PARAMETERS.

WE DEAL WITH:

ANALYTICAL METHODS & NUMERICAL METHODS ← GRADIENT DESCENT

- FIND A CLOSED FORM SOL EXACT SOLUTIONS
- APPROXIMATED SOLUTIONS MIGHT BE FASTER

- MOST COMMONLY USED TO FIND THE $\min(f(x))$

ITERATIVE

- SET A STARTING HYPERPARAMETER α

- Randomly initialize parameters $\theta^{(0)}$
- Repeat until Convergence ⟶ take a step at a time down the gradient

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla f$$

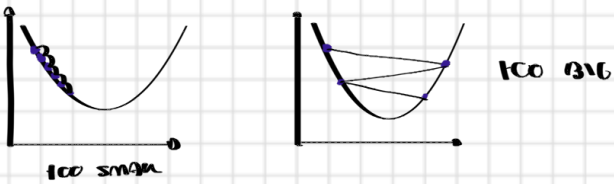
∇f → Gradient Vector $\begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{bmatrix}$ Vector of partial derivatives of a function wrt its params

We stop once we've reached the minimum.

Note

∇f points in the direction of greatest ascent, to move down the gradient, we use it's negative

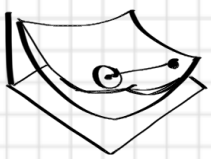
Sensitivity



Convex & Differentiable

- Basic GD methods find the optimum minimum if function is:

Differentiable & Convex



Linear Regression Application

- Use GD to train machine learning models
- where $f(\theta)$ is the cost/loss function & θ = vector of model parameters
- Take i.e Linear Regression with just one parameter

$$\hat{y} = \beta_0 + \beta_1 x \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Optimization: $\min_{\beta} (MSE)$

$$MSE = J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \longrightarrow \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i)^2 \qquad \hat{y}_i = \beta_0 + \beta_1 x_i$$

$$\frac{\partial J(\beta)}{\partial \beta_0} = \frac{2}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i) = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \quad (A)$$

$$\frac{\partial J(\beta)}{\partial \beta_1} = \frac{2}{n}\sum_{i=1}^{n} x_i(\beta_0 + \beta_1 - y_i) = \frac{2}{n}\sum_{i=1}^{n} x_i(\hat{y}_i - y_i) \quad (B)$$

$$\begin{bmatrix} A \\ B \end{bmatrix} = \nabla f$$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla f$$

$$\beta_0^{(t+1)} = \beta_0 - \alpha \cdot A$$

$$\beta_1^{(t+1)} = \beta_1 - \alpha B$$

Update Simultaneously!!!

Different types

Batch GD → All training data is used to compute GD

Stochastic GD → Considers just 1 datapoint at the time ⇝ Good in escaping local minima

Mini-Batch GD → Computes ∇f on small random sets called mini-batches → may escape local minima

LDA & QDA

- Up until Generative Models we've only seen discriminative ones
  - Both model do the same thing → $P(y|x)$
  - Just discriminative model them directly from the data i.e Logistic Regression
  - While Generative ones take a step back & model the joint distribution of features

$$P(y|x) = \frac{P(x,y)}{p(x)}$$

Joint Distribution

- The joint distribution of features x & class y can be specified in two parts:

$$p(x,y) = P(y=k)\,p(x|y=k)$$

$P(y=k)$: Class prior — $p(x|y=k)$: class conditional → describes $\forall k$ the distribution of features

- What do they do?
  - They give us the posterior class distribution
  - But also give estimates of joint distribution, can be used for simulating data
- Modeling the full joint distribution gives posterior probabilities

LDA

- Generative model that gives linear decision boundaries in classification problems
  - Model the joint distribution $p(x,y)$
  - End up with a discriminant function $g_k(x)$ → used to classify the correct class where among all values of $g_k(x)$: we choose class with highest

$$y = \text{argmax}\, g_k(x)$$

Assumptions:

Class conditionals are Gaussian with:

- Class-specific mean
- Common Variance

We can build a Bayes Classifier Based on this

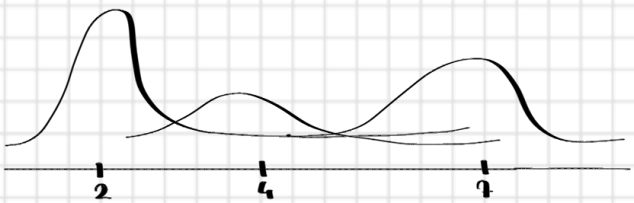
We can derive an optimal solution under the assumption that we know the true dist of data

- Class conditionals are Gaussian
- We know an parameters (mean, variance)
- We know the class priors.

$P(x|y=\text{BLACK}) = N(2,1)$ $\quad \pi_{\text{BLACK}} = 0.6$

$P(x|y=\text{RED}) = N(4,1)$ $\quad \pi_{\text{RED}} = 0.1$

$P(x|y=\text{BLUE}) = N(7,1)$ $\quad \pi_{\text{BLUE}} = 0.3$



Now if we look at the log of joint distribution:

$$\log p(x,y) = \log p(y=k) + \log p(x|y=k)$$
$$= \log \pi_k + \log \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu_k)^2}{\sigma^2}} \right\}$$

For which by further simplification

$$g_k(x) = \frac{\mu_k}{\sigma^2}x - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

QDA

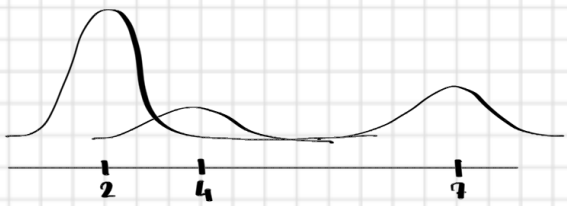
- Given that the assumption of a common variance in LDA may be too strict
- In QDA:
  - Class conditionals are Gaussian with both

Class specific mean & class specific variance

$P(x|y=\text{BLACK}) = N(2, 0.25)$ $\quad \pi_k = 0.6$

$P(x|y=\text{RED}) = N(4, 1)$ $\quad \pi_k = 0.1$

$P(x|y=\text{BLUE}) = N(7, 0.8^2)$ $\quad \pi_k = 0.3$



QDA results in higher posterior

more flexibility & quadratic boundaries

Plug-in Classifiers

- Gold Standard → a lot of assumptions!!!
  - All the knowledge of the params are approximations

Priors: $\hat{\pi}_k = \frac{n_k}{n}$ $\qquad$ Mean: $\hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i$

Var(LDA): $\hat{\sigma}^2 = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2$ $\qquad$ Var(QDA): $\hat{\sigma}_k^2 = \frac{1}{n_k-1}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2$

Multiple Features

- With multiple features we need multivariate distributions for class conditionals

We talk about:

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix} \qquad EX = \begin{bmatrix} EX_1 \\ \vdots \\ EX_p \end{bmatrix}$$

Covariance matrix

$$\text{Var}\,X = E[(X-EX)(X-EX)^T]$$

- Looking at factors involving x the pdf is

$$p(x,\mu,\Sigma) = \frac{1}{(2\pi)^{p/2}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \rightarrow \propto e^{-\frac{1}{2}\|x-\mu\|_2^2}$$

EFFECTIVELY BEING A CONTOUR CURVE

LDA & QDA MULTIPLE FEATURES

LDA → INDIVIDUAL MEANS, EQUAL COVARIANCE MATRICES

QDA → INDIVIDUAL MEANS, CLASS SPECIFIC COVARIANCE MATRICES

FOR LDA $g_k(x) = 2\log\pi_k - (x-\mu_k)^T \Sigma^{-1} (x-\mu_k)$

AND THE DECISION BOUNDARIES BETWEEN CLASS j AND k ARE ALL POINTS:

$$g_j(x) = g_k(x)$$