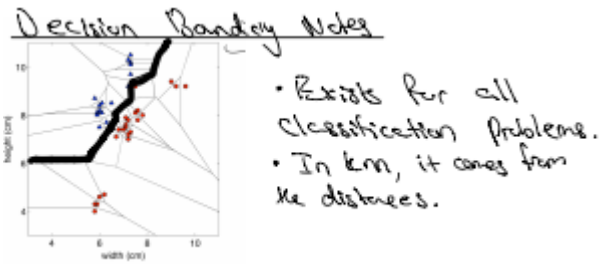
LECTURE 1

Machine Learning: Learn from Data or experience
Supervised Learning: Access to labeled examples of the correct behavior
(Two types of problems here, Classification or Regression)
Unsupervised Learning: No Labeled Examples
Reinforcement Learning: Learning to maximize a reward system
Input Vectors: Representing images, audio as a vector (or matrix) of numbers.

KNN GOAL & NOTES - MOSTLY for Classification

- Goal is to classify the input x into which of the 1, 2, or 3, or.. N class it falls into
- What we do is read in the training data
- Then put the test data over it, and see for a particular x, what are it's K closest neighbours.
- "Closest" is defined by a measure, usually as Euclidean distance.

  X(a) - X(b) = Sqrt(sum (x_j - x_i))
- The decision boundary between two classes, is be NON-linear with KNN



- KNN is hard to work with in High dimensions, or mislabelled data (you can solve this by choosing more than 1 closest neighbour)
- KNN has a Hyperparameter K, which determines the number of closest neighbours to choose.
- Always good to NORMALIZE your data when working with KNN to help reduce the curse of dimensionality (basically working with high dimensions)
- Does all of its work at TEST time -- No learning required!

KNN ALgorithm:

- Find K samples {x,t} closest to x.
- Classify the input based on the majority of the neighbours it falls within.

HOW TO choose a good K?

- Small K:
  - Good at capturing small fine grained patterns
  - May overfit

- Large K:
  - May underfit, fails to capture important irregularities
  - Makes stable predictions by averaging over lots of examples

Validation and Test Set

- We want something to validate how well our model is doing, so we have a validation set and a test set to keep track of this.
- If we are doing good in training set, but validation set is bad / not improving -- We are over fitting our model
- If Validation accuracy > training accuracy -> Underfitting

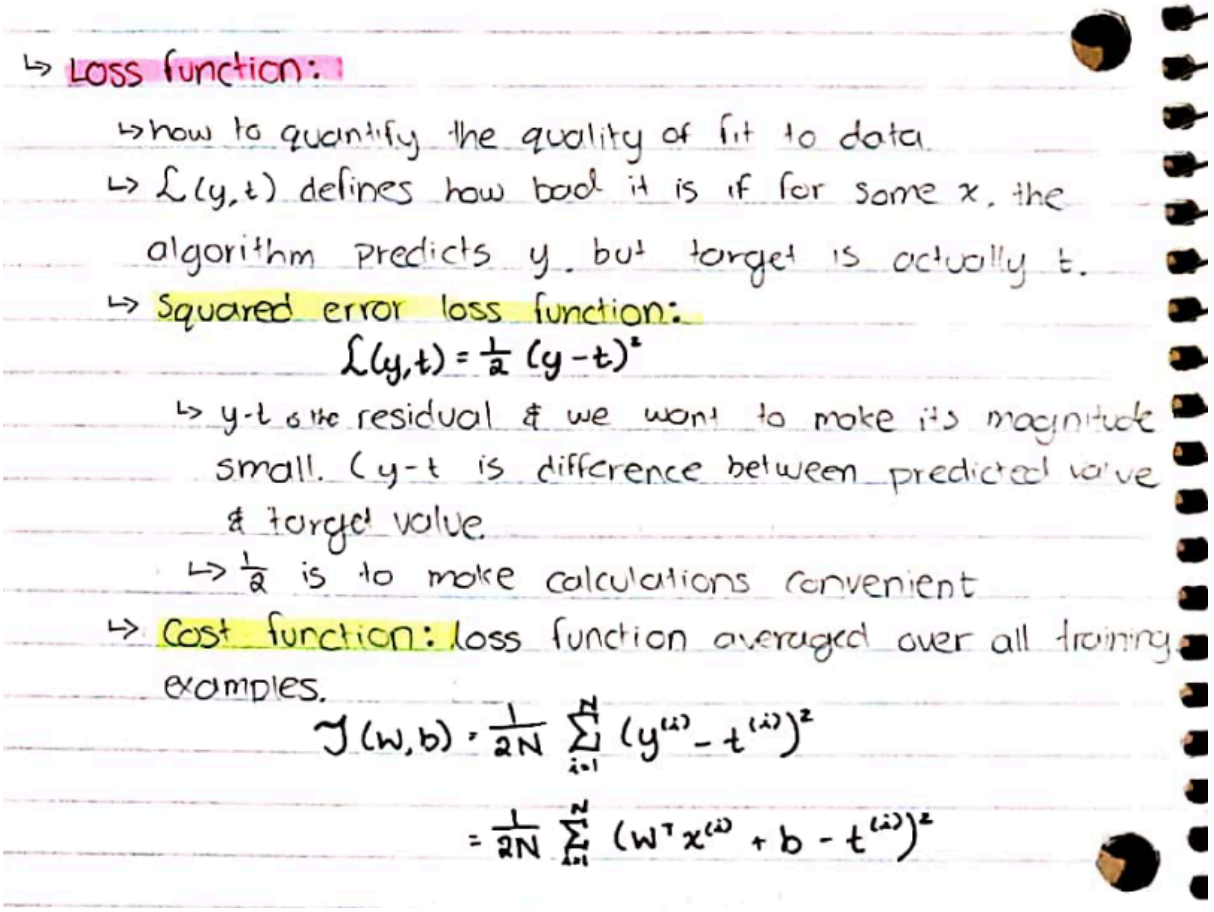
Computational Costs

- O(ND)
- Sort distances O(N LOG N)
- Need to start entire dataset in memory

LECTURE 2

Linear Regression - Supervised learning model

- Make a prediction y given inputs, and with some weight matrix w.
- $y = Wx + b \rightarrow$ vectorized it is: $y = w^t x$
- To loss function (to see how well our model is performing during training) is: predictions - the true value



- In vectorized notation, our predictions are just y = Xw
- Linear Regression (or MLR) have a closed form solution to find the weights W (The parameters in this case)
- This closed form solution is:
    - W = (X.Transpose() * X)^ -1 X.transpose * t

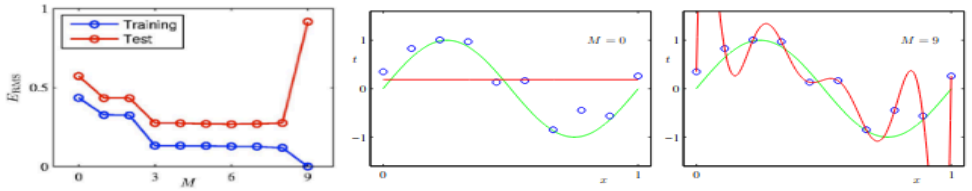
$$\mathbf{w}^{\mathrm{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

Polynomial Regression / Basis Expansion

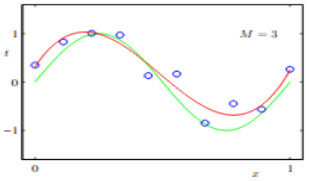
- Sometimes we want to use a more “smooth” / “Curved” model to represent our data. We can still do this with MLR / a linear model, (Note that is is Linear in Parameters, for example the weight vector isn’t w^2 or w^3)
- We do this by applying NONlinear functions to our input data -- Such as making x^2, or X^3, etc.
- NOTICE that you will have to featurize your train data (X), and the Test data (X).
- To find the weights, we can still use the closed form formula for W still! (As shown above)
- The predictions also work in the same way as before, y = featurized(x) * W
- You can control the “smoothness” of the prediction by the degree of the polynomial, i.e Degree =3,9,etc. (Value of M, the degree is a hyper parameter here and we can tune it using validation).

Model Complexity and Generalization

Underfitting (M=0): model is too simple — does not fit the data.
Overfitting (M=9): model is too complex — fits perfectly.



Good model (M=3): Achieves small test error (generalizes well).



L2 or L1 Regularization:

- Restricting the number of parameters of a model (M here) is a crude approach to control the complexity of the model.
- SO we bring in a penalty Term, lambda, to penalize the weights by scaling them.

- If the weights are too big, and are making the model too complex -- We can “tune” it by choosing a larger value of Lambda.
- Large values of Lambda penalize weights more, where as small lambda penalizes less.
- For the least squares problem, the model now takes into account the y = Wx + b and the penalized term. This is known as the ridge regression

For the least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$.

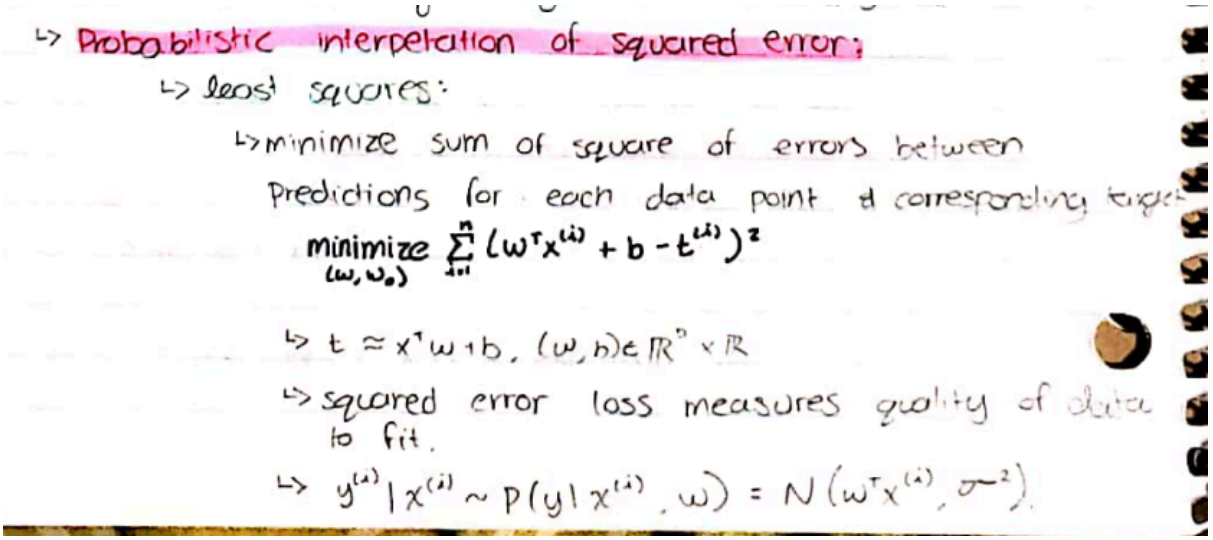
- When $\lambda > 0$ (with regularization), regularized cost gives

$$\mathbf{w}_\lambda^{\text{Ridge}} = \underset{\mathbf{w}}{\text{argmin}}\, \mathcal{J}_{\text{reg}}(\mathbf{w}) = \underset{\mathbf{w}}{\text{argmin}}\, \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$
$$= (\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$$

- The case $\lambda = 0$ (no regularization) reduces to least squares solution!
- Q: What happens when $\lambda \to \infty$?
- Note that it is also common to formulate this problem as $\text{argmin}_{\mathbf{w}}\, \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ in which case the solution is $\mathbf{w}_\lambda^{\text{Ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$.

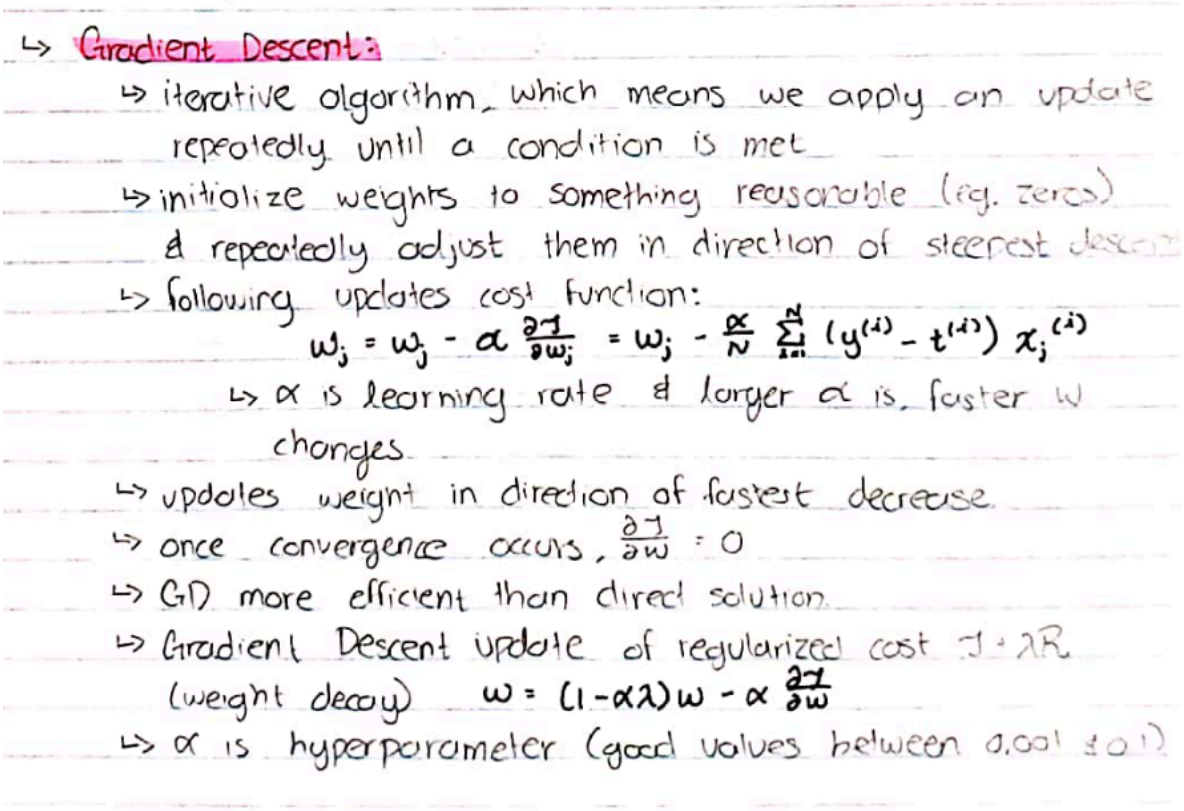
- IF Lambda = 0, this would mean NO regularization, and we are back to least squares
- If lambda -> infinity, the coefficients would go close to 0.

Probabilistic Least Squares



GRADIENT DESCENT

The OTHER way to train your model: (Instead of direct closed form solution) would be Gradient Descent:



- Weight updates are performed AFTER you compute the loss.
- The Pipeline is as follows: Prediction -> compute loss -> update weights -> repeat.

LECTURE 3

LINEAR Classification

- Same thing as Linear Regression, but now you pass the result through a SIGMOID to be able to classify the output into one of the 2 classes.

↳ **Binary linear classification:**

↳ classification: predict a discrete-valued target

↳ Binary: predict binary target $t \in \{0, 1\}$

↳ training examples with $t = 1$ are called positive examples

↳ training examples with $t = 0$ are called negative examples.

↳ linear: model of linear function x, followed by threshold r:

$$z = w^T x + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

↳ **Simplifications:**

↳ eliminating threshold: we can assume w.l.o.g. that $r = 0$.

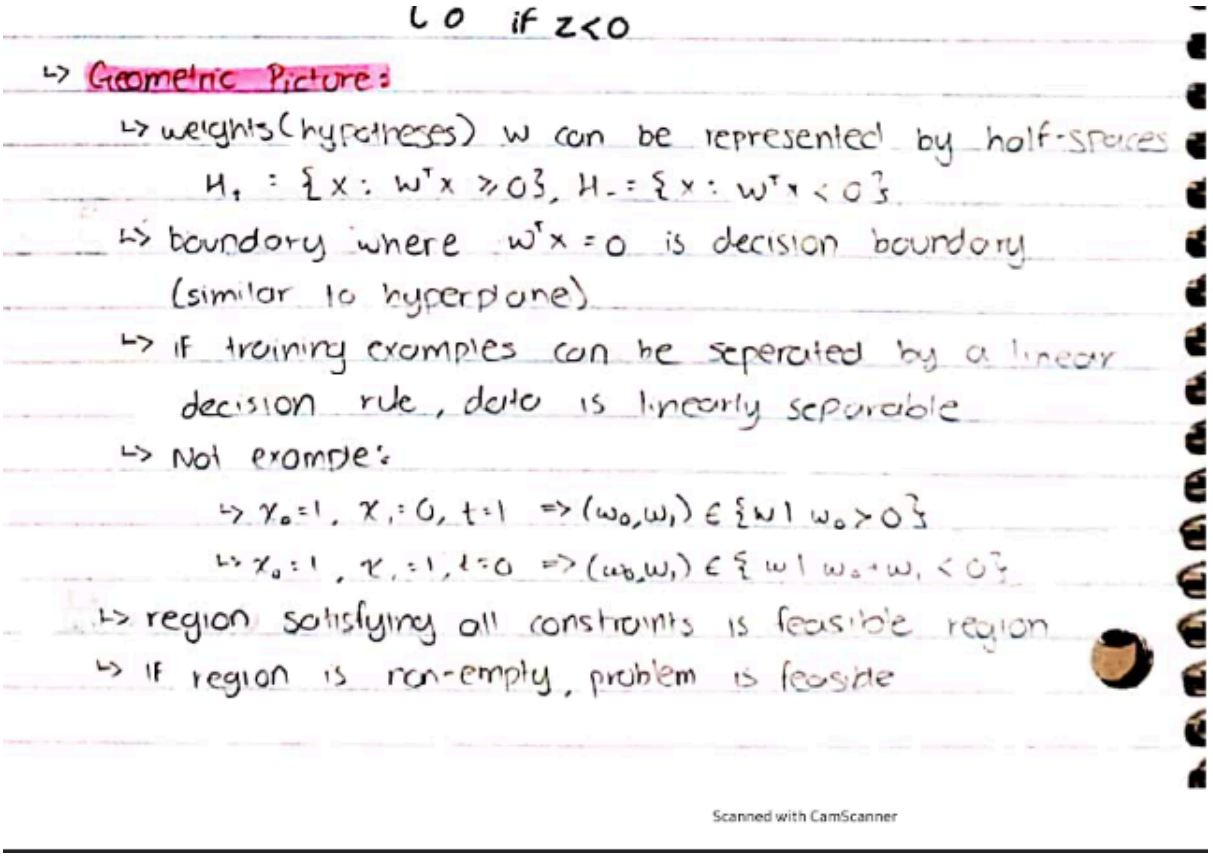
$$w^T x + b \geq r \iff w^T x + \underbrace{b - r}_{\triangleq w_0} \geq 0$$

↳ eliminating bias: add dummy feature x_0 which always takes on value 1. weight $w_0 = b$ is bias.

$$z = w^T x$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- **Geometric Picture for Linear Classification**



Cross Entropy Loss

↳ **loss functions:** (0-1) loss

↳ $\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases}$

$= \mathbb{I}[y \neq t]$

↳ $J = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]$ (cost / misclassification error).

↳ minimum of function at critical points.

$$\frac{\partial \mathcal{L}_{0-1}}{\partial w_j} = \frac{\partial \mathcal{L}_{0-1}}{\partial z} \cdot \frac{\partial z}{\partial w_j}$$

↳ problem with 0-1 loss function is that it's defined in terms of final prediction, which involves discontinuity.

↳ loss function penalizes you for making correct predictions with high confidence. (if $t=1$, loss is larger if $z=10$ than $z=0$.

↳ **Logistic Activation Function:**

↳ logistic function is kind of sigmoid:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \sigma^{-1}(y) = \log\left(\frac{1}{1-y}\right) \text{ (logit)}$$

↳ linear model with logistic non-linearity is log-linear:

$$z = w^T x + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{se}(y,t) = \frac{1}{2}(y-t)^2$$

↳ $z \ll 0$, we have $\sigma(z) \approx 0$

↳ $\frac{\partial L}{\partial z} \approx 0 \Rightarrow \frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_j} \approx 0$

↳ if prediction is wrong we are far from critical point

↳ **Logistic Regression:**

↳ since $y \in [0,1]$, we can interpret it as estimated probability $t=1$.

↳ Cross-entropy loss:

$$\mathcal{L}_{ce}(y,t) = \begin{cases} -\log y & \text{if } t=1 \\ -\log(1-y) & \text{if } t=0 \end{cases}$$
$$= -t\log y - (1-t)\log(1-y).$$

STOCHASTIC GRADIENT DESCENT

↳ **Gradient for Logistic Repression:**

$$w := w - \alpha/N \sum_{i=1}^{n} (y^{(i)} - t^{(i)}) x^{(i)}$$

↳ **Stochastic Gradient Descent:**

↳ update parameters based on the gradient for single training example

1. choose i uniformly at random
2. $\theta := \theta - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$

↳ cost of SGD update is independent of N

↳ unbiased estimate of batch gradient:

$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \theta}\right] = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \theta} = \frac{\partial J}{\partial \theta}$$

↳ variance may be high

↳ we can't exploit efficient vectorized operations

↳ compromise approach is to compute gradients on randomly chosen medium sized set of data called mini-batch.

↳ stochastic gradients computed on larger mini-batches have smaller variance

↳ $|M|$ is hyperparameter (reasonable $|M| = 100$)

↳ **SGD Learning Rate:**

↳ use large learning rate early in training to get close to optimum

↳ gradually decay learning rate to reduce fluctuations

↳ gradient descent with small step-size converges to first minimum it finds.

LECTURE 4

Classification

Binary: 2 targets

Multiclass: > 2 targets

One hot encoding:

Suppose t = {1,0,2}

One hot = {[0,1,0],[1,0,0],[0,0,1]}

Multi class classification vectorized: z = Wx+b

Softmax:

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

- Outputs positive & sum to 1 (like probabilities)

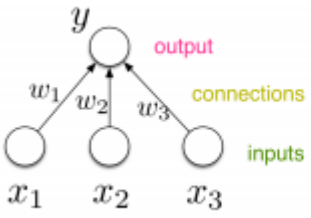
Cross entropy as loss function (log applied elementwise):

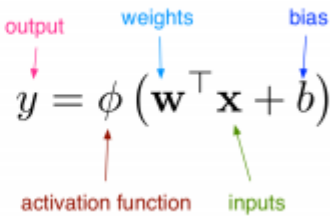
$$\begin{aligned}\mathcal{L}_{\text{CE}}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log y_k \\ &= -\mathbf{t}^\top (\log \mathbf{y}),\end{aligned}$$

Softmax + cross entropy combined:

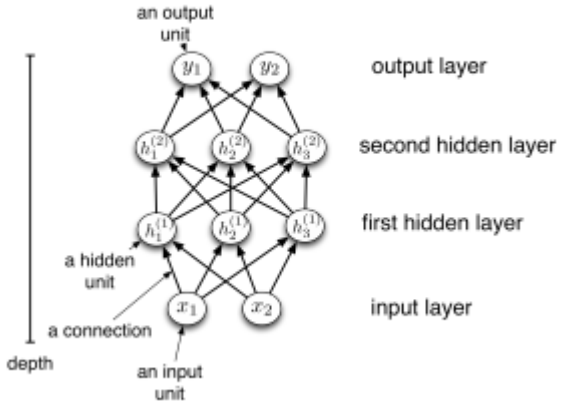
$$\begin{aligned}\mathbf{z} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{y} &= \text{softmax}(\mathbf{z}) \\ \mathcal{L}_{\text{CE}} &= -\mathbf{t}^\top (\log \mathbf{y})\end{aligned}$$

Neural nets Model:





Multilayer Perceptrons



- Feed-forward network (directed acyclic graph (no cycles))
- There's also recurrent neural networks (can have cycles - we didn't cover it in class)
- Fully connected layer: all input units connected to all output units
- Output units are a function of input units

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

  - Theta is the activation function, like sigmoid etc

Feature Learning

Feature detector: each first layer hidden unit which computes $\phi(\mathbf{w}_i^T \mathbf{x})$

Linear layer: the activation function of a layer is the identity

- Any sequence of linear layers can be equivalently represented as a single linear layer $\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$

Deep linear networks more expressive than linear regression

Universal function approximators: multilayer feed-forward neural nets with nonlinear activation function; they can approximate any function arbitrary well

Limits:

Might need to represent exponentially large network

Finding weights to represent a given function?

Might overfit if you can just learn any function

We want a compact representation

Univariate chain rule (loss & derivatives)

Computing the loss:

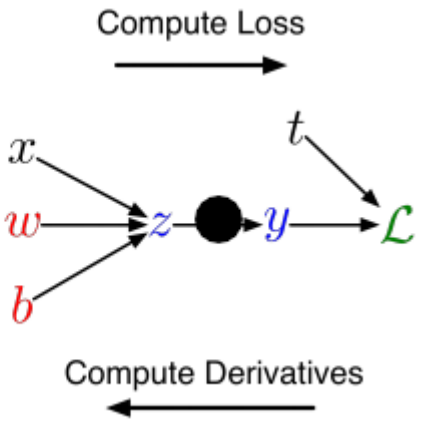
$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

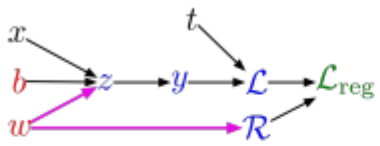
Computing the loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



Examples of forward and backward pass:

Example: univariate logistic least squares regression



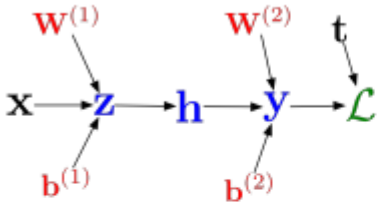
Forward pass:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$
$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\lambda$$
$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}$$
$$\overline{y} = \overline{\mathcal{L}}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = \overline{\mathcal{L}}\,(y - t)$$
$$\overline{z} = \overline{y}\frac{\mathrm{d}y}{\mathrm{d}z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} = \overline{z}\,x + \overline{\mathcal{R}}\,w$$
$$\overline{b} = \overline{z}\frac{\partial z}{\partial b} = \overline{z}$$

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Computational cost of forward pass:

- One add-multiply operation per weight
  - $$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

Computational cost of backward pass:

- Two add-multiply operations per weight
  - $$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$
    $$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

Note: backward pass is about as expensive as two forward passes
For multilayer perceptron: cost is linear in the number of layers, quadratic in number of units per layer
Note: backward prop used to train the overwhelming majority of neural nets today

- Algorithm much “fancier” than gradient descent use backprop to compute the gradients

Note: backprop is believed to be neurally implausible (no connection to how the brain works??)

LECTURE 5 AND LECTURE 6

Please note, I tried my best to explain these concepts, sorry if I made any mistakes

<u>Maximum Likelihood (In general):</u>

- Consider a scenario in which you have multiple distributions, and you observe some random variable X, maximum likelihood estimation is concerned with identifying the distribution which most *likely* produced this data point.
- Recall a likelihood function is defined as follows:
  - $$L(\theta) = f_\theta(x_1) * f_\theta(x_2) * ... * f_\theta(x_n) = \prod_{i=1}^{n} f_\theta(x_i)$$
  - Where F(x1) is essentially the distribution of X given parameter theta
  - Maximum likelihood estimation is concerned with maximizing this likelihood function.
- How do we maximize the likelihood function?
  - Note: Ln(x) is a 1-1 increasing function of X;
  - We can use this to simplify the likelihood function because it is easier to differentiate a sum rather than a product
  - $$l(\theta) = ln \prod_{i=1}^{n} f_\theta(x_i) = \sum_{i=1}^{n} ln f_\theta(x_i)$$
  - Now, we simply solve the following equation:
    - $$\frac{\partial l(\theta)}{\partial \theta} = 0 \text{ for } \theta$$
    - Then you check if the second derivative of the log-likelihood is less than zero (to ensure that it is indeed a maxima) and you can then conclude that theta is indeed a maximum likelihood estimate
    - Note: Bonner did not mention second derivatives; so on a test you can skip taking the second derivative (I think?)

<u>Another Person explaining MLE:</u>

↳ Maximum likelihood estimation:

↳ input data = $\{x^{(1)}, \ldots, x^{(N)}\}$ & outputs come from $y^{(i)} \sim p(y|x^{(i)}, w)$, with unknown parameter w.

$\Rightarrow D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$.

↳ likelihood function is $pr(D|w)$

↳ Maximum likelihood estimation (MLE):

↳ principle that suggests we have to find a parameter $\hat{w}$ that maximizes the likelihood

$$\hat{w} = \underset{w}{\text{argmax}}\ Pr(D|w)$$

↳ independent samples: likelihood function of samples D is product of likelihoods.

$$p(y^{(1)}, \ldots y^{(n)} | x^{(1)}, \ldots, x^{(N)}, w) = \prod_{i=1}^{N} P(y^{(i)} | x^{(i)}, w) = L(w)$$

↳ maximum likelihood is equivalent to minimizing negative log-likelihood.

$$l(w) = -\log L(w) = -\log \prod_{i=1}^{N} P(z^{(i)}|w) = -\sum_{i=1}^{n} \log P(z^{(i)}|w)$$

MLE is the base for this whole lecture

Two approaches to Classification:

Discriminative Approach:

Estimates the decision boundary directly from the training data.

Mapping INput to Output

Generative Approach:

Models the distribution of the inputs generated from the class.

- Recall, in Bayesian statistics, the parameter that we are estimating (θ) is considered a random variable. Parameters are believed to be random variables that follow some distribution.

- The distribution of a parameter is called the prior. We update this prior belief using our observed data. i.e., generally, the heights of students at UofT will be between 5 and 7 feet. Our goal is to update this prior belief using new data.

- The posterior is the probability of the parameter (θ) given the new observations.

How do we calculate this posterior distribution?

- In this Bayesian approach first, we calculate the Class Likelihood (The likelihood for the data observed, given the parameter) (The same as in MLE)

- $L(\boldsymbol{s}|\theta) = f(x_1, x_2, ..., x_n|\theta)$
- You then multiply this class likelihood by the prior and divide by the evidence.
- $$\underbrace{p(c|\mathbf{x})}_{\text{Pr. class given words}} = \frac{p(\mathbf{x}, c)}{p(\mathbf{x})} = \frac{\overbrace{p(\mathbf{x}|c)}^{\text{Pr. words given class}} \quad p(c)}{p(\mathbf{x})}$$
- $$\text{posterior} = \frac{\text{Class likelihood} \times \text{prior}}{\text{Evidence}}$$

Important Note: The posterior distribution is directly proportional to the (Class likelihood x prior), hence if you are trying to maximize the posterior (C), you don't really have to calculate "evidence"

Bayesian Parameter Estimation

A supplemental example from STA261:

Consider the following;

$(X_1, X_2, ..., X_n) \sim Bern(\theta)$ and $\theta \sim Unif[0, 1]$

Given this information, theta coming from the uniform distribution is your prior (previous information about the parameter).

The dataset coming from the Bernoulli, this data is used to create the class likelihood function.

prior:

$\pi(\theta) = 1\, I(0 \le \theta \le 1)$

$L(s|\theta) = \theta^{\sum x_i} (1-\theta)^{n - \sum x_i}$ <- this is the class likelihood function creating by multiplying n Bernoulli observations under the parameter theta.

Then, your posterior distribution is simply the following:

$\pi(\theta|s) \propto \theta^{\sum x_i} (1-\theta)^{n-\sum x_i} * 1\, I(0 \le \theta \le 1)$

Now let us look at Bonner's Coin Flip Example:

The distribution of the dataset is Bernoulli, hence the MLE is:

$$L(\theta) = p(\mathcal{D}|\theta) = \theta^{N_H}(1-\theta)^{N_T}$$

Now, we specify the prior.

Bonner states that we can utilize the beta distribution depending on the dataset:

$$p(\theta; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\,\theta^{a-1}(1-\theta)^{b-1}.$$

Where, the gamma function (the greyed-out thing) can be ignored due to proportionality.

$$p(\theta; a, b) \propto \theta^{a-1}(1-\theta)^{b-1}.$$

Finally, he computes the posterior distribution as follows:

$$\begin{aligned} p(\boldsymbol{\theta} \mid \mathcal{D}) &\propto p(\boldsymbol{\theta}) p(\mathcal{D} \mid \boldsymbol{\theta}) \\ &\propto \left[\theta^{a-1}(1-\theta)^{b-1}\right]\left[\theta^{N_H}(1-\theta)^{N_T}\right] \\ &= \theta^{a-1+N_H}(1-\theta)^{b-1+N_T}. \end{aligned}$$

Finally, he computes the posterior mean as follows.

This is just a beta distribution with parameters $N_H + a$ and $N_T + b$.
The posterior expectation of θ is:

$$\mathbb{E}[\theta \mid \mathcal{D}] = \frac{N_H + a}{N_H + N_T + a + b}$$

Which in this case our predicted parameter theta.

Benefits of this approach:
Deals better with data sparsity (explained below)

Cons:
This is an integration problem, there aren't really any comparable tools (like gradient descent) for Bayesian parameter estimation.

Naïve Bayes -> (A different type of estimation)
A method of estimation that uses the Naïve bayes assumption.
Note, while we use the words "Class Likelihood and Prior", here we simply use them to refer to the formulas. In Naïve Bayes they do not function the same way as in the Bayesian Parameter estimation we just did. Instead, the data is just used to make maximum likelihood estimates.

How does one calculate Class Likelihood and Prior?

Recall the following:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)}$$

$$P(B|A) = \frac{P(A \text{ and } B)}{P(A)}$$

Thus, the entire numerator of the posterior distribution formula is simply the second formula seen here.

Naïve Bayes Learning

Let C be a class (0,1).

Where: $p(c^{(i)}) = \pi^{c^{(i)}}(1-\pi)^{1-c^{(i)}}$.

- To compute $p(c|\mathbf{x})$ we need: $p(\mathbf{x}|c)$ and $p(c)$

  - In order to calculate them, we must make the **Naïve Bayes Assumption,** which essentially states that data points are *conditionally* independent if they are conditioned on a class label

  - Then, you simply calculate as follows:

$$p(c, x_1, \ldots, x_D) = p(c)p(x_1|c)\cdots p(x_D|c).$$

  - This is your numerator.

How else does this naïve bayes assumption help us?

Basically, you calculate the MLE for every parameter in the dataset given every class C, and this is your class likelihood function (for that specific class). Then for prediction you simply punch in the X value for your different posterior estimates based on the different classes and you choose the largest one.

(Calculating the Class Likelihood)

When you create the log likelihood of theta, under this naïve bayes class likelihood, it will separate into independent terms for each feature, hence you can optimize the parameters differently.

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(c^{(i)}, \mathbf{x}^{(i)}) = \sum_{i=1}^{N} \log \left\{ p(\mathbf{x}^{(i)} | c^{(i)}) p(c^{(i)}) \right\}$$

$$= \underbrace{\sum_{i=1}^{N} \log p(c^{(i)})}_{\substack{\text{Bernoulli log-likelihood} \\ \text{of labels}}} + \sum_{j=1}^{D} \underbrace{\sum_{i=1}^{N} \log p(x_j^{(i)} \mid c^{(i)})}_{\substack{\text{Bernoulli log-likelihood} \\ \text{for feature } x_j}}$$

Now, you simply maximize the Bernoulli log-likelihood of labels.

$$\sum_{i=1}^{N} \log p(c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \log \pi + \sum_{i=1}^{N} (1 - c^{(i)}) \log(1 - \pi)$$

In essence, you are simply calculating the MLE's for each parameter given the dataset.

- Obtain MLEs by setting derivatives to zero:

$$\hat{\pi} = \frac{\sum_i \mathbb{I}\{c^{(i)} = 1\}}{N} = \frac{\#\text{ spams in dataset}}{\text{total }\#\text{ samples}}$$

Where,

- Log-likelihood:

$$\sum_{i=1}^{N} \log p(x_j^{(i)} \mid c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \left\{ x_j^{(i)} \log \theta_{j1} + (1 - x_j^{(i)}) \log(1 - \theta_{j1}) \right\} + \sum_{i=1}^{N} (1 - c^{(i)}) \left\{ x_j^{(i)} \log \theta_{j0} + (1 - x_j^{(i)}) \log(1 - \theta_{j0}) \right\}$$

- Obtain MLEs by setting derivatives to zero:

$$\hat{\theta}_{jc} = \frac{\sum_i \mathbb{I}\{x_j^{(i)} = 1 \;\&\; c^{(i)} = c\}}{\sum_i \mathbb{I}\{c^{(i)} = c\}} \overset{\text{for } c=1}{=} \frac{\#\text{word } j \text{ appears in spams}}{\#\text{ spams in dataset}}$$

Then, to actually use this model to make predictions, you simply apply bayes rules and take the largest output based on the different classes:

$$p(c \mid \mathbf{x}) = \frac{p(c) p(\mathbf{x} \mid c)}{\sum_{c'} p(c') p(\mathbf{x} \mid c')} = \frac{p(c) \prod_{j=1}^{D} p(x_j \mid c)}{\sum_{c'} p(c') \prod_{j=1}^{D} p(x_j \mid c')}$$

$$p(c \mid \mathbf{x}) \propto p(c) \prod_{j=1}^{D} p(x_j \mid c)$$

Issues with this:
When you have data that for example, which simply contains heads; the MLE estimates for the parameter will lean towards heads. This is called data sparsity.

While we are using Bayes Rule, we are not using it in its full capacity to update our prior beliefs. This is what we did earlier in Bayesian Parameter Estimation.

Benefits:
This is simply a optimization problem, where we can simply apply gradient descent.

<u>Maximum A-posteriori Estimation (Different type of Estimation)</u>

The goal is to find the most likely parameter -> BUT UNDER THE POSTERIOR!
However, this is simply a subset (approximation) of the full Bayesian parameter estimation, since we are not actually getting the full posterior distribution, but rather finding one (expected) parameter.

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} &= \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{\theta} \mid \mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{\theta}, \mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{\theta})\, p(\mathcal{D} \mid \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \; \log p(\boldsymbol{\theta}) + \log p(\mathcal{D} \mid \boldsymbol{\theta})\end{aligned}$$

- Maximize by finding a critical point

$$0 = \frac{\mathrm{d}}{\mathrm{d}\theta} \log p(\theta, \mathcal{D}) = \frac{N_H + a - 1}{\theta} - \frac{N_T + b - 1}{1 - \theta}$$

- Solving for θ,

$$\hat{\theta}_{\mathrm{MAP}} = \frac{N_H + a - 1}{N_H + N_T + a + b - 2}$$

What happened?
We simply calculate our posterior distribution (as before) and we maximize it in terms of theta, by taking a derivative, setting equal to zero and solving.

Now in most cases you would take a second derivative to check if it is indeed a maximum, but Bonner did not mention this.

MLE VS MAP VS BAYESIAN
In most cases, (imo) full Bayesian will be the best because you are getting a distribution on the data.

Comparison of estimates in the coin flip example:

	Formula	$N_H = 2, N_T = 0$	$N_H = 55, N_T = 45$
$\hat{\theta}_{\text{ML}}$	$\frac{N_H}{N_H+N_T}$	1	$\frac{55}{100} = 0.55$
$\mathbb{E}[\theta\|\mathcal{D}]$	$\frac{N_H+a}{N_H+N_T+a+b}$	$\frac{4}{6} \approx 0.67$	$\frac{57}{104} \approx 0.548$
$\hat{\theta}_{\text{MAP}}$	$\frac{N_H+a-1}{N_H+N_T+a+b-2}$	$\frac{3}{4} = 0.75$	$\frac{56}{102} \approx 0.549$

$\hat{\theta}_{\text{MAP}}$ assigns nonzero probabilities as long as $a, b > 1$.

Gaussian Discriminant Analysis

$p(\mathbf{x}|t = k)$ may be very complex

$$p(x_1, \cdots, x_d, t) = p(x_1|x_2, \ldots, x_d, t) \cdots p(x_{d-1}|x_d, t)p(x_d|t)p(t)$$

- Naive Bayes used a conditional independence assumption to get

$$p(x_1, \cdots, x_d, t) = p(x_1|t) \cdots p(x_{d-1}|t)p(x_d|t)p(t)$$

What else could we do?

- Choose a simple distribution.

In this, we simply assume that X comes from a normal distribution, this is your new class likelihood function. And we find an MLE for the posterior distribution under this assumption.

$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, a Gaussian (or normal) distribution defined as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right]$$

where $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix $\boldsymbol{\Sigma}$.

Now, under this distribution, we simply calculate and maximize the log-likelihood function as before (This would be the class likelihood):

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log \prod_{i=1}^{N} \left[\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu}) \right\} \right]$$

$$= \sum_{i=1}^{N} \log \left[\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu}) \right\} \right]$$

$$= \sum_{i=1}^{N} \underbrace{-\log(2\pi)^{d/2}}_{\text{constant}} - \log|\boldsymbol{\Sigma}|^{1/2} - \frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu})$$

$$0 = \frac{\mathrm{d}\ell}{\mathrm{d}\boldsymbol{\mu}} = -\sum_{i=1}^{N} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}} \frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu})$$

$$= -\sum_{i=1}^{N} \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu}) = 0$$

Solving for this, gives us the mean of the sample data.

Again, in this case we choose the class with the highest posterior probability given class K.

GDA (GBC) decision boundary is based on class posterior $p(t_k|\mathbf{x})$.
We choose a class with the highest posterior probability, i.e.,
$\text{argmax}_k\, p(t_k|\mathbf{x})$.

However, to calculate this class K, we further take an MLE estimate on the posterior, using the class likelihood we defined above.

$$\begin{aligned} \log p(t_k|\mathbf{x}) &= \log \frac{p(\mathbf{x}|t_k)p(t_k)}{p(x)} = \log p(\mathbf{x}|t_k) + \log p(t_k) - \log p(\mathbf{x}) \\ &= -\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}_k^{-1}| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \\ &\quad + \log p(t_k) - \log p(\mathbf{x}) \end{aligned}$$

If you wish to calculate the decision boundary between two classes, you can do this:

$$\log p(t_k|\mathbf{x}) = \log p(t_l|\mathbf{x}).$$

(I don't think this would come up on exam personally) *

Note, This will form a quadratic Decision Boundary

However, after we calculate the MLE for the class posterior likelihood, you will end up with the following closed form:

$$\hat{\phi} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}\{t^{(n)}=1\}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{n=1}^{N}\mathbb{I}\{t^{(n)}=k\}\mathbf{x}^{(n)}}{\sum_{n=1}^{N}\mathbb{I}\{t^{(n)}=k\}}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{\sum_{n=1}^{N}\mathbb{I}\{t^{(n)}=k\}}\sum_{n=1}^{N}\mathbb{I}\{t^{(n)}=k\}(\mathbf{x}^{(n)}-\hat{\mu}_{t^{(n)}})(\mathbf{x}^{(n)}-\hat{\mu}_{t^{(n)}})^T$$

GDA vs Logistic Regression

- GDA is a generative model, LR is a discriminative model.
- GDA makes stronger modelling assumption that the class-conditional data is a multivariate Gaussian.
- If this is true, GDA is asymptotically efficient.
- But LR is more robust, less sensitive to incorrect modelling assumptions (what loss is it optimizing?)
- When these distributions are non-Gaussian (true almost always), LR usually beats GDA
- GDA can handle easily missing features

General Conclusion for Generative Models

- GDA has quadratic; LR has linear decision boundary
- With shared covariance, GDA leads to a model similar to logistic regression (but with different estimation procedure).
- Generative models:
  - Flexible models, easy to add/remove class.
  - Handle missing data naturally
  - More “natural” way to think about how data is generated.
- Tries to solve a hard problem in order to solve a easy problem.

Types of Classification

Discriminative approach: estimate parameters of decision boundary/class separator directly from labeled examples.

- ➔ "How do I separate these classes?"
- ➔ Learn p(t|**x**) directly

Generative approach: model the distribution of inputs generated from the class (Bayes classifier).

- ➔ What does each class "look" like?
- ➔ Build a model of p(**x**|t) (Baye's Rule)

$$\underbrace{p(c|\mathbf{x})}_{\text{Pr. class given words}} = \frac{p(\mathbf{x},c)}{p(\mathbf{x})} = \frac{\overbrace{p(\mathbf{x}|c)}^{\text{Pr. words given class}} \quad p(c)}{p(\mathbf{x})}$$

$$\text{posterior} = \frac{\text{Class likelihood} \times \text{prior}}{\text{Evidence}}$$

Naive Bayes

- ➔ Makes a strong assumption: probabilities are ***conditionally independent*** given a class.
- ➔ This simplifies a probability calculation to:

$$p(c, x_1, \ldots, x_D) = p(c)p(x_1|c)\cdots p(x_D|c).$$

Learning

Training time: Estimate parameters using maximum likelihood.
Test time: Use Bayes' theorem.

Prior

- ➔ Using a uniform distribution for the prior makes no assumption on the class, but a beta distribution can be used to add additional assumptions.

LECTURE 7
Principal Component Analysis

Unsupervised Learning: Motivating Examples

- Some examples of situations where you would use unsupervised learning:
    - You want to understand how a scientific field has changed over time. You want to take a large database of papers and model how the distribution of topics changes from year to year. UBt what are the topics?
    - You are a biologist studying animal behaviour, so you want to infer a high-level description of their behaviour from video. You don't know the set behaviour ahead of time.
    - You want to reduce your energy consumption, so you take a time series of your energy consumption over time, and try to break it down into separate components (when refrigerator, washing machine, etc. were operating or not)
- Common theme: You have some data, and you want to infer the structure underlying the data.
- This structure is *latent*, which means it is not observed.
- Example: Determine groups of people in an image of people lined up:
    - Based on clothing styles
    - Gender, age, etc.
- Example: Determine moving objects in videos.

Overview:

- Today we'll cover the first unsupervised learning algorithm for this course: *Principal Component Analysis (PCA)*
- PCA is a *dimensionality reduction* method.
- Dimensionality reduction: Mapping the data to a lower dimensional space.
    - Saving computation and memory
    - Reducing overfitting and achieve better generalization visualizing
- PCA is a linear model. It is useful for understanding many other similar algorithms.
    - Autoencoders
    - Matrix factorizations
- We use a lot of linear algebra in today's lecture.
    - Especially orthogonal matrices and eigendecompositions

Setup: Multivariate Inputs

- Setup: Given an i.i.d dataset D = $\{x^{(1)},\dots x^{(N)}\} \subset \mathrm{R}^{\mathrm{D}}$
- N instances/ Observations/ Examples

$$\mathbf{X} = \begin{bmatrix} [\mathbf{x}^{(1)}]^\top \\ [\mathbf{x}^{(2)}]^\top \\ \vdots \\ [\mathbf{x}^{(N)}]^\top \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_D^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_D^{(N)} \end{bmatrix}$$

- Mean: $\mathbb{E}[\mathbf{x}^{(i)}] = \boldsymbol{\mu} = [\mu_1, \cdots, \mu_D]^T \in \mathbb{R}^D$

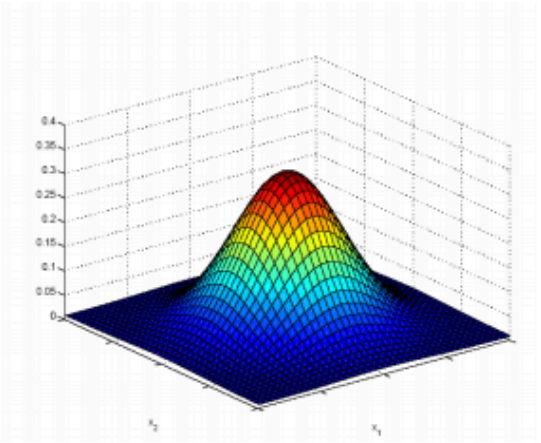
- Covariance:

$$\boldsymbol{\Sigma} = \text{Cov}(\mathbf{x}^{(i)}) = \mathbb{E}[(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^\top] = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_D^2 \end{bmatrix}$$

Multivariate Gaussian Model

- $\mathbf{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, a Gaussian (or normal) distribution defined as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right]$$



Mean and Covariance Estimators

- Observed data: D = $\{x^{(1)}, \ldots x^{(N)}\}$
- Recall that the MLS estimators for the mean μand ∑ under the multivariate Gaussian model is given by (previous lecture)

- Sample mean: $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}$ (quantifies (approximately) where data is located)
- Sample Covariance: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^{\top}$
  - Quantifies (approximately) how your data points are spread

Bivariate Normal

$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ $\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$ $\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.8 \\ 0.8 & 1 \end{pmatrix}$

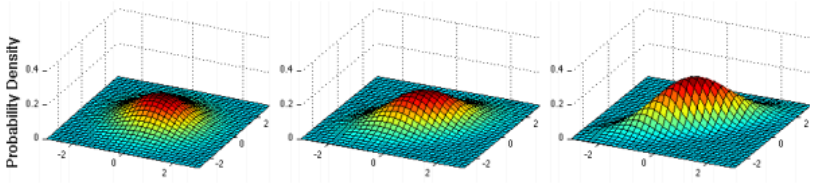


Figure: Probability density function

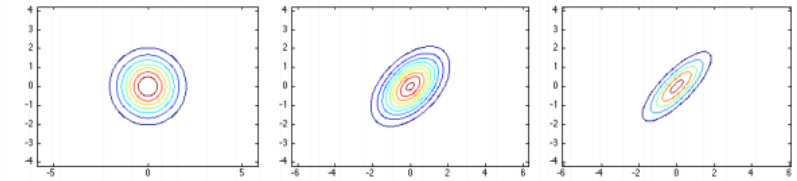
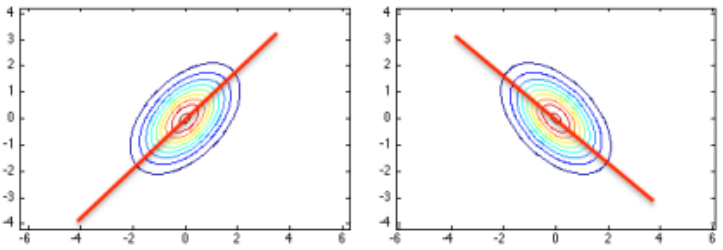
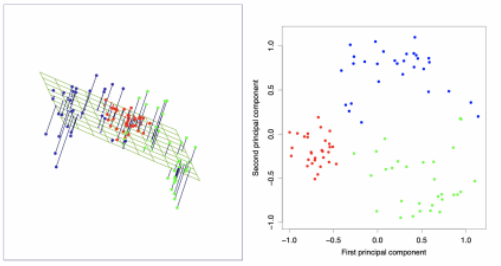


Figure: Contour plot of the pdf

Low Dimensional Representation

- Sometimes in practice, even though data is very high dimensional, its important features can be accurately captured in a low dimensional subspace.



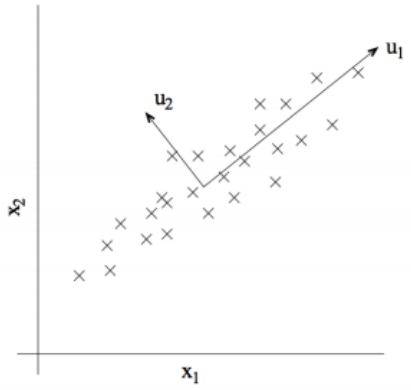


- Find a low dimensional representation of data
    - Computational benefits
    - Interpretability, visualization
    - Generalization

Projection onto a Subspace

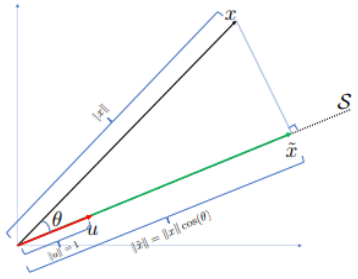
- D = $\{x^{(1)},\ldots x^{(N)}\} \subset R^D$
- Set μto the sample mean of the data, $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}$
- Goal: Find a K-dimensional linear subspace $S \subset R^D$ such that $x^{(n)}$ - μ is "well-represented" by its projection onto a K-dimensional S.
- Recall: The projection of a point x onto S is the point in S closest to x. More on this coming soon.

We are Looking for Directions



- For example, in a 2-dimensional problem, we are looking for the direction $\mathbf{u}_1$ along which the data is well represented.
- DIfferent interpretation of "well represented":
    - (1) Direction of highest variance
    - (2) Direction of minimum difference after projection
- It turns out they are the same.

Euclidean Projection

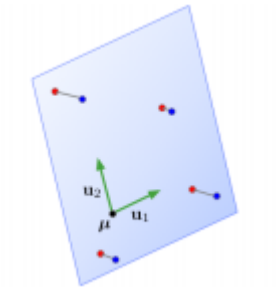


- Here, S is the line along the unit vector **u** (1- dimensional subspace)

    - **u** is a basis for S: any point in S can be written as ***z*u** for some ***z***.
- Projection of x on S is denoted by $Proj_s(x)$
- Recall: $x^T\mathbf{u} = ||x||\ ||u||\cos(\theta) = ||x||\cos(\theta)$
- $Proj_s(x) = x^T u \bullet u = ||x||\mathbf{u}$
  = length of proj • direction of proj

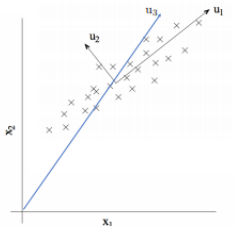
General Subspace

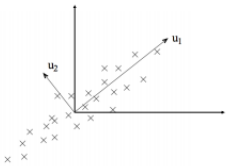
- In general, S is not one dimensional (i.e., line), but a (linear) subspace with a dimension K.
- In this case, we have K basis vectors $u_1, ..., u_K \in R^D$: any vector **y** in S can be written as $y = \sum_{i=1}^{K} z_i u_i$ for some $z_1, ..., z_K$



- Projection of $x \in R^D$ on this subspace is given by $Proj_S(x) = \sum_{i=1}^{K} z_i u_i$ where $z_i = \mathbf{x}^T\mathbf{u}_i$

First Step: Center Data





- Directions we compute will pass through origin, and should represent the direction of highest variance.
- We need to center our data since we don't want the location of data to influence our calculations. We are only interested in finding the direction of highest variance. This is independent from its mean.
- ⇒ We are **not** interested in $\mathbf{u_3}$ above, we are interested in $\mathbf{u_1}$

Projection onto a Subspace

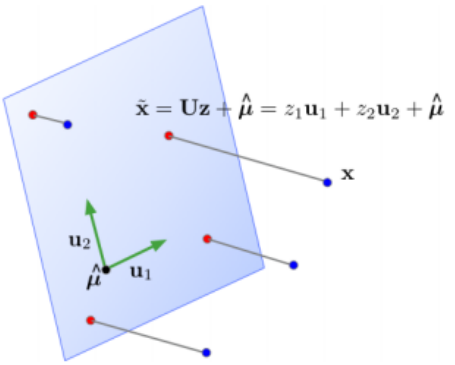
- Let $\{u_K\}_{k=1}^{K}$ be an orthonormal basis of the subspace S (a K-dimensional linear subspace of R^D
- Approximate each data point $\mathbf{x} \in R^D$ as:
    - 1. Center (subtract the mean)
    - 2. Project onto S

- 3. Add the mean back

$$\tilde{x} = \hat{\mu} + Proj_S\left(x - \hat{\mu}\right)$$
$$= \mu + \sum_{k=1}^{K} z_k u_k$$

- We also know: $z_k = u_k^T\left(x - \hat{\mu}\right)$
- Let $U \in R^{D*K}$ be a matrix with columns $\{u_K\}_{k=1}^{K}$
- Then $z = U^T\left(x - \hat{\mu}\right)$ (Note that $z \in R^K$)
- Also: $\tilde{x} = \hat{\mu} + Uz = \hat{\mu} + UU^T\left(x - \hat{\mu}\right)$ (Note that $\tilde{x} \in R^D$)
- Here UU^T is the projector onto S, and $U^T U = 1$

- Note that x and $\tilde{x}$ have the same dimensionality. That is, they are both in R^D.
- But $\tilde{x}$ lives in a low dimensional subspace in R^D
- Its low dimensional representation is $z \in R^K$



$$\mathbf{z} = \mathbf{U}^\top(\mathbf{x} - \hat{\boldsymbol{\mu}})$$

- In machine learning, $\tilde{x}$ is also called the *reconstruction* of x.
- **Z** is its *representation* or *code*

- If we have a K-dimensional subspace in a D-dimensional input space, then $x \in R^D$ and $z \in R^K$
- If the data points **x** all lie close to their reconstructions, then we can approximate distances, etc. in terms of the same operations on the code vectors **z**.
- If K << D, then it is much cheaper to work with **z** than **x**.
- A mapping to a space that is easier to manipulate or visualize is called a *representation*, and learning such a mapping is *representation learning*.
- Mapping data to a low-dimensional space is called *dimension reduction*.

Learning a Subspace

- How to choose a good subspace S?

- Need to choose D x K matrix U with orthonormal columns.
- Two criteria:
  - Minimize the reconstruction error: Find vectors in a subspace that are closest to data points.

$$min\ U\ \frac{1}{N}\sum_i ||x^{(i)} - \tilde{x}^{(i)}||^2$$

  - Maximize the *variance of reconstructions*: Fins a subspace where data has the most variability.

$$maxU\ \frac{1}{N}\sum_i ||\tilde{x}^{(i)} - \hat{\mu}||^2$$

- The data and its reconstruction has the same means (exercise)!
- These two criteria are equivalent!
  - We show that

$$\frac{1}{N}\sum_i ||x^{(i)} - \tilde{x}^{(i)}||^2 = const - \frac{1}{N}\sum_i ||\tilde{x}^{(i)} - \hat{\mu}||^2$$

- Recall that $\tilde{x}^{(i)} = \hat{\mu} + Uz^{(i)}$ and $z^{(i)} = U^T(x^{(i)} - \hat{\mu})$
- Observation 1:

$$\left|\left|\tilde{x}^{(i)} - \hat{\mu}\right|\right|^2 = \left(Uz^{(i)}\right)^T\left(Uz^{(i)}\right) = [z^{(i)}]^T U^T U z^{(i)} = [z^{(i)}]^T z^{(i)} = \left|\left|z^{(i)}\right|\right|^2$$

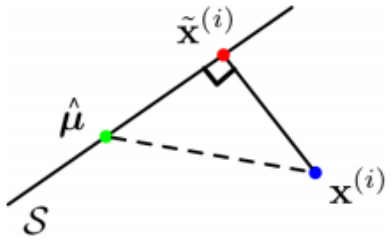
Norm of centered reconstruction is equal to the norm of representation.

Recall: For any two (compatible) matrices, A and B, $(AB)^T = (B^TA^T)$

Pythagorean Theorem

- Observation 1: $||\tilde{x}^{(i)} - \hat{\mu}||^2 = ||z^{(i)}||^2$
  - Variance of reconstructions is equal to variance of code vectors:
    - $\frac{1}{N}\sum_i ||x^{(i)} - \tilde{x}^{(i)}||^2 = \frac{1}{N}\sum_i ||z^{(i)}||^2$ (Exercise $\frac{1}{N}\sum_i z^{(i)} = 0$)

- Observation 2: Orthogonality of $\tilde{x}^{(i)} - \hat{\mu}$ and $\tilde{x}^{(i)} - x^{(i)}$.
  - (Two vectors a, b are orthogonal $\Leftrightarrow a^Tb = 0$)
  - Recall $\tilde{x}^{(i)} = \hat{\mu} + UU^T(x^{(i)} - \hat{\mu})$



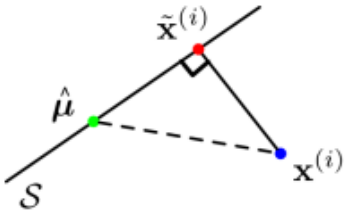
To show the orthogonality of $\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}$ and $\tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}$, observe that

$$
\begin{aligned}
&(\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}})^\top (\tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}) \\
&= (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top \mathbf{U}\mathbf{U}^\top (\hat{\boldsymbol{\mu}} - \mathbf{x}^{(i)} + \mathbf{U}\mathbf{U}^\top (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})) \\
&= (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top \mathbf{U}\mathbf{U}^\top (\hat{\boldsymbol{\mu}} - \mathbf{x}^{(i)}) + (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top \mathbf{U}\mathbf{U}^\top (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}) \\
&= 0
\end{aligned}
$$

Because of the orthogonality of $\tilde{x}^{(i)} - \hat{\mu}$ and $\tilde{x}^{(i)} - x^{(i)}$, we can use the Pythagorean theorem to conclude that

$$|| \tilde{x}^{(i)} - \hat{\mu} ||^2 + ||x^{(i)} - \tilde{x}^{(i)} ||^2 = ||x^{(i)} - \hat{\mu} ||^2$$

By averaging over the data, we obtain



$$
\underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\|^2}_{\text{projected variance}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2}_{\text{reconstruction error}}
$$

$$
= \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\|^2}_{\text{constant}}
$$

Therefore, projected variance = constant - reconstruction error.
Maximizing the variance is equivalent to minimizing the reconstruction error!

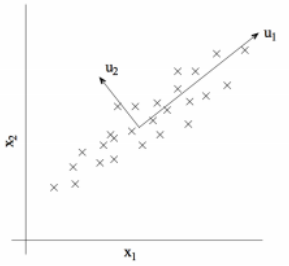
Principal Component Analysis
Choosing a subspace to maximize the projected variance, or minimize the reconstruction error, is called *principal component analysis (PCA)*.
Recall:

- Spectral Decomposition: a symmetric matrix **A** has a full set of eigenvectors, which can be chosen to be orthogonal. This gives a decomposition $A = Q\Lambda Q^T$, where Q is orthogonal and $\Lambda$is diagonal. The columns of Q are eigenvectors, and the diagonal entries $\lambda_j$of Λ are the corresponding eigenvalues.
- That is, symmetric matrices are diagonal in some basis.
- A symmetric matrix A is positive semidefinite iff each $\lambda_j \geq 0$.
- The matrix Q is an orthogonal matrix, i.e., it satisfies $Q^TQ = QQ^T = I$

- Consider the *empirical covariance matrix:*

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top$$

- Recall, Covariance matrices are symmetric and positive semidefinite
- The optimal PCA subspace is spanned by the top K eigenvectors of $\hat{\boldsymbol{\Sigma}}$
    - More precisely, choose the first K of any orthonormal eigenbasis for $\hat{\boldsymbol{\Sigma}}$
    - The general case is tricky, but we will show this for K = 1.



- These eigenvectors are called *principal components, analogous to the principal axes of an ellipse.*

Recap:

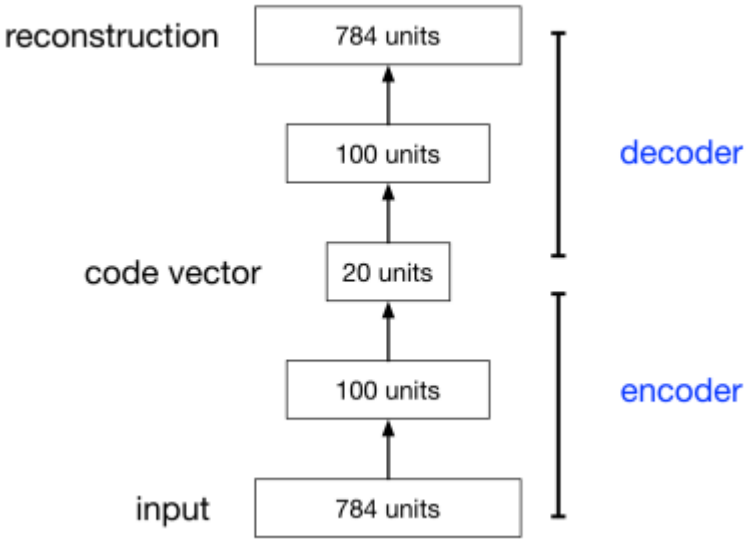
- Dimensionality reduction aims to find a low-dimensional representation of the data.
- PCA projects the data onto a subspace which maximizes the projected variance, or equivalently, minimizes the reconstruction error.
- The optimal subspace is given by the top eigenvectors of the empirical covariance matrix.
- PCA gives a set of decorrelated features.

Applying PCA to faces

- Consider running PCA on 2429 19x19 grayscale images (CBCL data)
- Can get good reconstructions with only 3 components
- PCA for pre-processins: can apply classifier to low-dimensional representation.
    - Original data is 361 dimensional
    - For face recognition PCA with 3 components obtains 79% accuracy on face/non-face discrimination on test data vs. 76.8% for a Gaussian mixture model (GMM) with 84 states.
- Can also be good for visualization.

Autoencoders

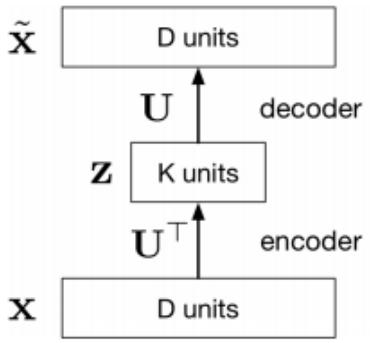
- An autoencoder is a feed-forward neural net whose job is to take an input **x** and predict **x**.
- To make this non-trivial, we need to add a *bottleneck layer* whose dimension is much smaller than the input.



Linear Autoencoders

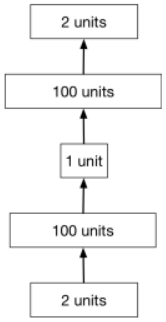
Why autoencoders?

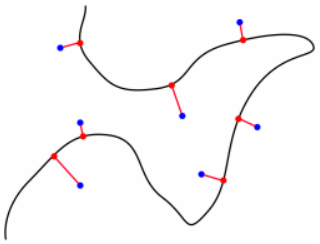
- Map high-dimensional data to two dimensions for visualization
- Learn abstract features in an unsupervised way so you can apply them to a supervised task
    - Unlabeled data can be much more plentiful than labeled data
- The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss.
    - $L(x,\tilde{x}) = || x - \tilde{x} ||^2$
- This network computes $\tilde{x} = W_2W_1$, which is a linear function.
- If $K \geq D$, we can choose W_2 and W_1 such that $W_2 W_1$ is the identity matrix. This isn't very interesting.
- But suppose $K < D$:
    - W_1 maps x to a K-dimensional space, so it is doing a dimensionality reduction.
- Observe that the output of the autoencoder must lie in a K-dimensional subspace spanned by the columns of W_2. This is because $\tilde{x} = W_2z$
- We saw that the best possible (min error) K-dimensional linear subspace in terms of reconstruction error is the PCA subspace.
- The autoencoder can achieve this by setting $W_1 = U^T$ and $W_2 = U$.
- Therefore, the optimal weights for a linear autoencoder are just the principal components



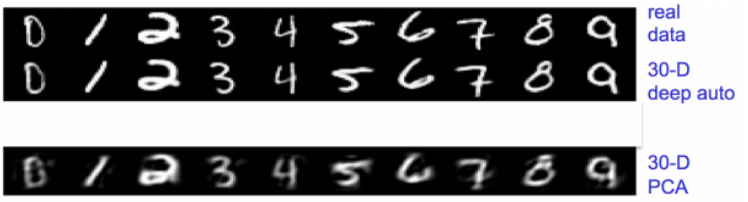
Nonlinear Autoencoders

- Deep nonlinear autoencoder learn to project the data, not onto a linear subspace, but onto a nonlinear *manifold*.
- This is a *nonlinear dimensionality reduction*.



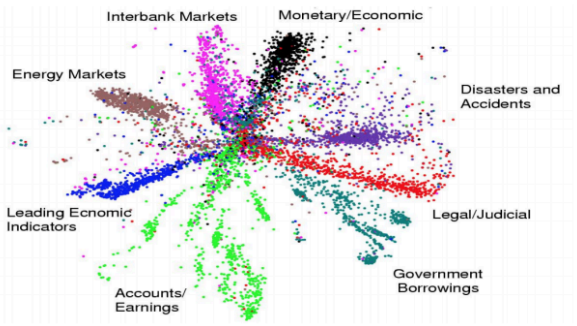


- Nonlinear autoencoders can learn more powerful codes fro a given dimensionality, compared with linear autoencoders (PCA)



Nonlinear Autoencoders

Here's a 2-dimensional autoencoder representation of newsgroup articles. They're colour-coded by topic, but the algorithm wasn't given the labels.



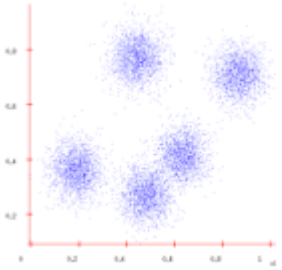
LECTURE 8
K-Means and EM Algorithm

Overview

- In last lecture we covered PCA, which was an unsupervised learning algorithm
    - Its main purpose was to reduce the dimension of the data
    - In practice, even though data is very high dimensional, it can be well represented in low dimensions
- This method relies on an assumption that data depends on some latent variables, which are not observed. Such models are called *latent variable models*.
    - For PCA, these corresponds to the code vectors (representation).
    - Today's lecture: K-means, a simple algorithm for *clustering*, i.e., grouping data points into clusters
    - Today's lecture: Reformulate clustering as a latent variable model, apply the EM algorithm

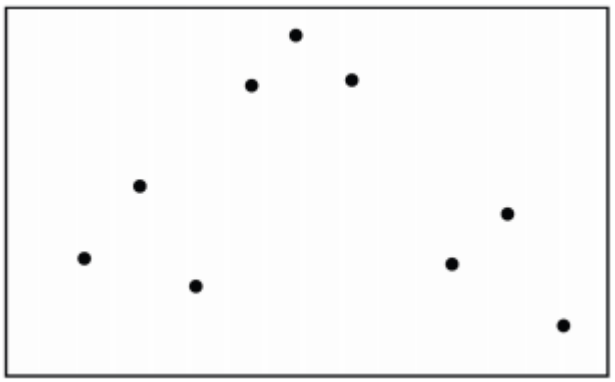
Clustering

- Sometimes the data form clusters, where samples within a cluster are similar to each other, and samples in different clusters are dissimilar:



- Such a distribution is *multimodal*, since it has *multiple nodes*, or regions of high probability mass.
- Grouping data points into clusters, *with no observed labels*, is called *clustering*. It is an unsupervised learning technique.
- Example: clustering machine learning papers based on topic (deep learning, Bayesian models, etc.)
    - But topics are never observed (unsupervised).

Clustering Problem



- Assume that the data points $\{x^{(1)},...,x^{(N)}\}$ live in an Euclidean space, i.e., $x^{(n)} \in R^D$.
- Assume that each data point belongs to one of K clusters.
- ASsume that the data points from same cluster are similar, ie.., close in Euclidean distance.
- How can we identify those clusters and the data points that belong to each cluster?

K-Means Objective

Let's formulate this as an optimization problem.

- *K-means Objective:* Find cluster centers $\{m_K\}_{k=1}^{K}$ and assignments $\{r^{(n)}\}_{n=1}^{N}$ to minimize the sum of squared distances of data points $\{x^{(n)}\}$ to their assigned cluster centres.
    - Data sample n = 1, …, N: $x^{(n)} \in R^D$ (observed)
    - Cluster centre k = 1,..., K: $m_k \in R^D$ (not observed)
    - Responsibilities: Cluster assignment for sample n: $r^{(n)} \in R^K$ 1-of-K encoding (not observed)
- Mathematically:

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} J\left(\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}\right) = \min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \left\|\mathbf{m}_k - \mathbf{x}^{(n)}\right\|^2$$

, where $r_k^{(n)} = I\{x^{(n)}$ is assigned to cluster k} e.g., $r^{(n)} = [0, \ldots, 1, \ldots, 0]^T$.

- Finding an optimal solution is an NP-hard problem!

K-Means Objective

- Optimization problem:

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K} r_k^{(n)} \left\|\mathbf{m}_k - \mathbf{x}^{(n)}\right\|^2}_{\text{distance between } \mathbf{x}^{(n)} \text{ and its assigned cluster centre}}$$

- Since $r_k^{(n)} = I\{x^{(n)}$ is assigned to cluster k} (e.g. $r^{(n)} = [0, \ldots, 1, \ldots, 0]^T$), the inner sum is over K terms but only one of them is non-zero.

- For example, if data point $x^{(n)}$ is assigned to cluster k = 3, then r^n = [0,0,1,0,...] and

$$\sum_{k=1}^{K} r_k^{(n)} \left\| \mathbf{m}_k - \mathbf{x}^{(n)} \right\|^2 = \left\| \mathbf{m}_3 - \mathbf{x}^{(n)} \right\|^2.$$

How to Optimize? Alternating Minimization
Optimization problem:

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} \left\| \mathbf{m}_k - \mathbf{x}^{(n)} \right\|^2$$

- Problem is hard when minimizing jointly over the parameters $\{m_k\}$, $\{r^{(n)}\}$.
- But if we fix one and minimize over the other, then it becomes easy.
- Doesn't guarantee the same solution!

Alternating Minimization (Optimizing Assignments)
Optimization Problem:

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} \left\| \mathbf{m}_k - \mathbf{x}^{(n)} \right\|^2$$

- Note:
  - If we fix the centres $\{m_k\}$, we can easily find the optimal assignments $\{r^{(n)}\}$ for each sample *n*

$$\min_{\mathbf{r}^{(n)}} \sum_{k=1}^{K} r_k^{(n)} \left\| \mathbf{m}_k - \mathbf{x}^{(n)} \right\|^2.$$

  - Assign each point to the cluster with the nearest centre
    - $r_k^{(n)}$ = {1 if k = arg $\min_j$ || $x^{(n)}$ - m_j ||^2, 0 otherwise}
  - E.g. if $x^{(n)}$ is assigned to cluster k, $r^{(n)}$ = $[0,0,...,1, \ldots, 0]^T$ (only kth entry is 1)

Alternating Minimization (Optimizing Centres)
- If we fix the assignments $\{r^{(n)}\}$, then we can easily find optimal centres $\{m_k\}$.
  - Set each cluster's centre to the average of its assigned data points: for l = 1, 2, ...K

$$0 = \frac{\partial}{\partial \mathbf{m}_l} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

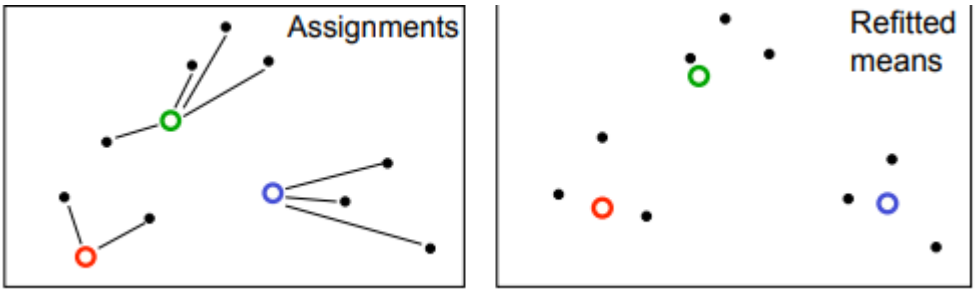
$$= 2 \sum_{n=1}^{N} r_l^{(n)} (\mathbf{m}_l - \mathbf{x}^{(n)}) \quad \Longrightarrow \quad \mathbf{m}_l = \frac{\sum_n r_l^{(n)} \mathbf{x}^{(n)}}{\sum_n r_l^{(n)}}$$

- Let's alternate between minimizing J($\{m_k\}$, $\{r^{(n)}\}$) with respect to $\{m_k\}$ and $\{r^{(n)}\}$
- This is called alternating minimization.

K - Means Algorithm

High level overview of algorithm:

- Initialization: randomly initialize cluster centres
- The algorithm iteratively alternates between two steps:
    - Assignment step: Assign each data point to the closest cluster
    - Refitting step: Move each cluster centre to the mean of the data assigned to it.



The K-Means Algorithm

- Initialization: Set K cluster means m1, …, m_K to random values
- Repeat until convergence (until assignments do not change)
    - Assignment: Optimize J with respect to {r}: Each data point $x^{(n)}$ assigned to nearest centre

$$\hat{k}^{(n)} = \arg\min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

    - And Responsibilities (1-hot or 1-of-K encoding)

$$r_k^{(n)} = \mathbb{I}\{\hat{k}^{(n)} = k\} \quad \text{for} \quad k = 1, .., K$$

- Refitting: optimize J with respect to {m}: Each centre is set to mean of data assigned to it

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}.$$

Questions about K-Means

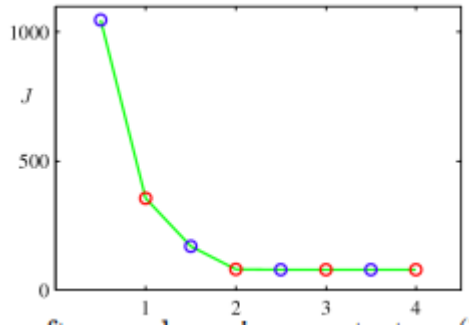
- Why does update set mk to mean of assigned points?
- What if we use da different distance measure?
- How can we choose the best distance?
- How to choose K?
- Will it converge?

Hard cases-unequal spreads, non-circular spreads, in-between points.

Why K-Means Converges

- K-means algorithm reduces the cost at each iteration
    - Whenevr asn assignment is changed, the sum squared distances J of data points from their assigned cluster centres is reduced.
    - Test for convergence: If the assignments do not change in the assignment step, we have converged (to at least a local minimum)

- This will always happen after a finite number of iterations, since the number of possible cluster assignments is finite.



- K-means cost function after each step (blue) and refitting step (red). The algorithm has converged after the third refitting step.

Local Minima

- The objective J is non-convex, (so coordinate descent on J is not guaranteed to converge to the global minimum).
- There is nothing to prevent k-means getting stuck at local minima
- We could try many random starting points.

Soft K-means

- Instead of making hard assignments of data points to clusters, we can make *soft assignments*. One cluster may have a responsibility of 0.7 for a data point and anothe rmay have a responsibility of 0.3.
  - Allows a cluster to use more information about hte data in the refitting step.
  - How do we decide on the soft assignments?
  - We already say this in multi-class classification:
    - 1-of-K encoding vs softmax assignments.

Soft K-Means Algorithm:

- Initialization: Set K means $\{m_k\}$ to random values
- Repeat until convergence (measured by how much J changes):
  - Assignment: Each data point n given soft "degree of assignment" to each cluster mean k, based on responsibilities

$$r_k^{(n)} = \frac{\exp[-\beta\|\mathbf{m}_k - \mathbf{x}^{(n)}\|^2]}{\sum_{j=1}^{K}\exp[-\beta\|\mathbf{m}_j - \mathbf{x}^{(n)}\|^2]}$$

$$\implies \mathbf{r}^{(n)} = \text{softmax}(-\beta\{\|\mathbf{m}_k - \mathbf{x}^{(n)}\|^2\}_{k=1}^{K})$$

  - Refitting: model parameters, means, are adjusted to match sample means of datapoints they are responsible for:

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)}\mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

Questions about soft K-means

Some remaining issues

- How to set β?
- Clusters with unequal width and weight?

These aren't straightforward to address with K-means. Instead we'll reformulate clustering using a generative model.

As $\beta \to \infty$, soft k-means becomes k-means! (Exercise!)

A Generative View of CLustering

- Next: probabilistic formulation of clustering
- We need a sensible measure of what it means to cluster the data well.
  - This makes it possible to judge different methods
  - It may help us decide on the number of clusters.
- An obvious approach is to imagine that the data was produced by a generative model
  - Then we adjust the model parameters using maximum likelihood i.e., to maximize the probability that iw ould produce exactly the data we observed.

The Generative Model:

- We'll be working with the following generative model for data D.
- Assume datapoint x is generated as follows:
  - Given a cluster *z* from {1,... K} such that p(z=k) = π_k
  - p(x| z = k) = $N(x|\mu_k, I)$

Clusters from the Generative Model

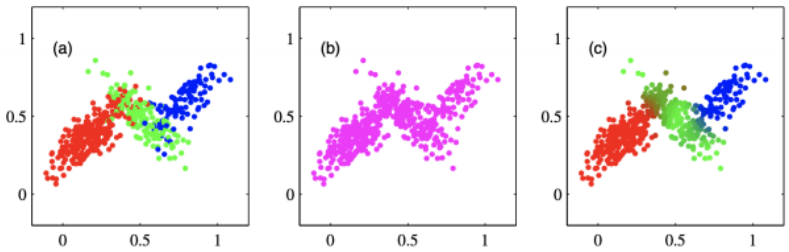
- This defines joint distribution p(z,x) = p(z)p(x|z) with parameters $\{\pi_k, \mu_k\}_{k=1}^{K}$
- The marginal of x is given by p(x) = $\sum_z p(z, x)$
- p(z = k|x) can be computed using Bayes rule

$$p(z = k|\mathbf{x}) = \frac{p(\mathbf{x} \mid z = k)p(z = k)}{p(\mathbf{x})}.$$

This tells us the probability that x comes from the kth cluster.

The Generative Model

- 500 points drawn from a mixture of 3 Gaussians



a) Samples from $p(\mathbf{x} \mid z)$ b) Samples from the marginal $p(\mathbf{x})$ c) Responsibilities $p(z \mid \mathbf{x})$

Maximum Likelihood with Latent Variables

- How should we choose the parameters $\{\pi_k, \mu_k\}_{k=1}^{K}$?
- Maximum likelihood principle: choose parameters to maximize likelihood of *observed data*
- We don't observe the cluster assignments z, we only see the data x
- Gvien data D = $\{x^{(n)}\}_{n=1}^{N}$, choose parameters that maximize:

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)})$$

We can find p(x) by marginalizing out z:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(z = k, \mathbf{x}) = \sum_{k=1}^{K} p(z = k) p(\mathbf{x} | z = k)$$

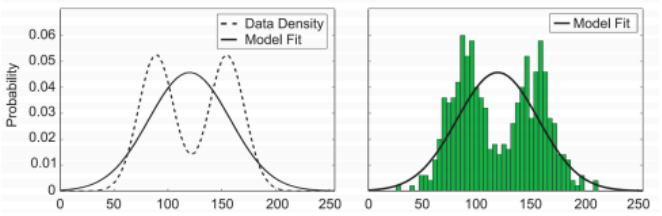
Gaussian Mixture Model (GMM)
What is p(x)?

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(z = k) p(\mathbf{x} | z = k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \mathbf{I})$$

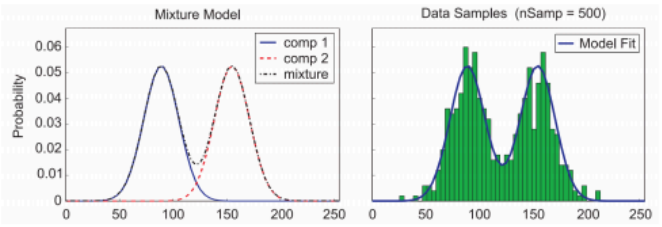
- This distribution is an example of *a Gaussian Mixture Model (GMM)* and π_k are known as the mixing coefficients.
- In general, we would have difference covariance for each cluster, i.e., p(x| z = k) = $N(x|\mu_k, I)$. For this lecture, we assume $\sum_k$ = I for simplicity.
- If we allow arbitrary covariance matrices, GMMs are universal approximators of densities (if you have enough Gaussians). Even diagonal GMMs are universal approximators.

Visualizing a Mixture of Gaussians - 1D Gaussians

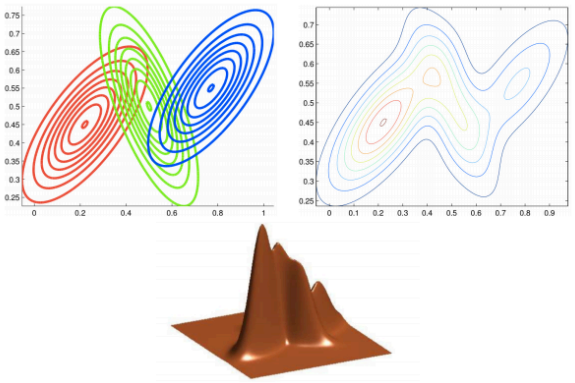
- If you fit one Gaussian distribution to data:



- Now, we are trying to fit a GMM with $K = 2$:



Visualizing a Mixture of Gaussians - 2D Gaussians



Fitting GMMs: Maximum Likelihood

Maximum likelihood objective:

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \boldsymbol{\mu}_k, \mathbf{I}) \right)$$

- How would you optimize this with respect to parameters $\{\pi_k, \mu_k\}$?
    - No closed-form solution when we set derivatives to 0
    - Difficult because sum inside the log
- One option: gradient descent → can we do better?
- Cna we have closed-form update?

Maximum Likelihood

- Observation: If we know $z^{(n)}$ for every $x^{(n)}$, (i.e., our dataset was $D_{Complete}$ = $\{(z^{(n)}, x^{(n)})\}_{n=1}^{N}$, the maximum likelihood problem is easy:

$$\begin{aligned}\log p(\mathcal{D}_{\text{complete}}) &= \sum_{n=1}^{N} \log p(z^{(n)}, \mathbf{x}^{(n)}) \\ &= \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}|z^{(n)}) + \log p(z^{(n)}) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{I}\{z^{(n)} = k\} \left(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k\right)\end{aligned}$$

$$\log p(\mathcal{D}_{\text{complete}}) = \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{I}\{z^{(n)} = k\} \left(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k\right)$$

- We have been optimizing something similar for Naive bayes classifiers
- By maximizing $\log p(\mathcal{D}_{\text{complete}})$, we would get this:

$$\begin{aligned}\hat{\boldsymbol{\mu}}_k &= \frac{\sum_{n=1}^{N} \mathbb{I}\{z^{(n)} = k\}\, \mathbf{x}^{(n)}}{\sum_{n=1}^{N} \mathbb{I}\{z^{(n)} = k\}} = \text{class means} \\ \hat{\pi}_k &= \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\{z^{(n)} = k\} = \text{class proportions}\end{aligned}$$

- We haven't observed the cluster assignments $z^{(n)}$, but we can compute $p(z^{(n)}|x^{(n)})$ using Bayes rule
- Conditional probability (using Bayes rule) of z given x

$$\begin{aligned}p(z = k|\mathbf{x}) &= \frac{p(z = k)p(\mathbf{x}|z = k)}{p(\mathbf{x})} \\ &= \frac{p(z = k)p(\mathbf{x}|z = k)}{\sum_{j=1}^{K} p(z = j)p(\mathbf{x}|z = j)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{I})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{I})}\end{aligned}$$

$$\log p(\mathcal{D}_{\text{complete}}) = \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{I}\{z^{(n)} = k\}(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k)$$

- We don't know the cluster assignments $\mathbb{I}\{z^{(n)}=k\}$ (they are our latent variables), but we know their expectation $\mathbb{E}[\mathbb{I}\{z^{(n)}=k\} \mid \mathbf{x}^{(n)}] = p(z^{(n)}=k|\mathbf{x}^{(n)})$.
- If we plug in $r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)})$ for $\mathbb{I}\{z^{(n)} = k\}$, we get:

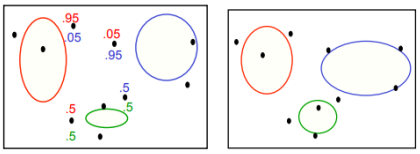
$$\sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k)$$

- This is still easy to optimize! Solution is similar to what we have seen:

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{n=1}^{N} r_k^{(n)}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} r_k^{(n)}} \qquad \hat{\pi}_k = \frac{\sum_{n=1}^{N} r_k^{(n)}}{N}$$

- Note: this only works if we treat $r_k^{(n)} = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_j, \mathbf{I})}$ as fixed.

How can we fit a mixture of Gaussians?

- This motivated the *Expectation-Maximization Algorithm*, which alternates between two steps:
  - 1. *E-Step:* Computer the posterior probabilities $r^{(n)}_k = p(z^{(n)} = k \mid x^{(n)})$ given our current model, i.e., how much do we think a cluster is responsible for generating a datapoint.
  - 2. M-step: Use the equations on the last slide to update the parameters, assuming $r^{(n)}_k$ are held fixed - change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.
  - 

- Initialize the means $\hat{\boldsymbol{\mu}}_k$ and mixing coefficients $\hat{\pi}_k$
- Iterate until convergence:
  - E-step: Evaluate the responsibilities $r_k^{(n)}$ given current parameters

$$r_k^{(n)} = p(z^{(n)}=k|\mathbf{x}^{(n)}) = \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I})}{\sum_{j=1}^K \hat{\pi}_j \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_j, \mathbf{I})} = \frac{\hat{\pi}_k \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_k\|^2\}}{\sum_{j=1}^K \hat{\pi}_j \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_j\|^2\}}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k}\sum_{n=1}^N r_k^{(n)}\mathbf{x}^{(n)}$$

$$\hat{\pi}_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N r_k^{(n)}$$

  - Evaluate log likelihood and check for convergence

$$\log p(\mathcal{D}) = \sum_{n=1}^N \log\left(\sum_{k=1}^K \hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I})\right)$$

What Just Happened: A Review

- The maximum likelihood objective $\sum_{n=1}^{N} log\, p(x^{(n)})$ was hard to optimize.
- The complete data likelihood objective was easy to optimize:

$$\sum_{n=1}^N \log p(z^{(n)}, \mathbf{x}^{(n)}) = \sum_{n=1}^N \sum_{k=1}^K \mathbb{I}\{z^{(n)} = k\}(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k)$$

- We don't know $z^{(n)}$'s (they are latent) so we replaced $I\{z^{(n)} = k\}$ with responsibilities $r_k^{(n)} = p(z^{(n)} = k \mid x^{(n)})$.
- That is, we replaced $I\{z^{(n)} = k\}$ with its expectation under $p(z^{(n)} \mid x^{(n)})$. E-Step
- We ended up with the expected complete data log-likelihood:

$$\sum_{n=1}^N \mathbb{E}_{p(z^{(n)}|\mathbf{x}^{(n)})}[\log p(z^{(n)}, \mathbf{x}^{(n)})] = \sum_{n=1}^N \sum_{k=1}^K r_k^{(n)}\left(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k\right)$$

Which we maximized over parameters $\{\pi_k, \mu_k\}_k$ (M-step)

- The EM algorithm alternates between
  - The E-step: computing the $r_k^{(n)} = p(z^{(n)} = k \mid x^{(n)})$ (ie. expectations $E[I\{z^{(n)} = k\}|x^{(n)}]$) given the current model parameters π_k, μ_k
  - The M-step: update the model parameters π_k, μ_k to optimize the expected complete data log-likelihood.

Relation to K-Means

- The K-means algorithm:

    - 1. Assignment Step: Assign each data point tot ehc closest cluster
    - 2. Refitting Step: move each cluster center tot he average of the data assigned to it.
- The EM Algorithm:
    - 1. E-step: Compute the posterior probability over z given our current model
    - 2. M-Step: Maximize the probability that it would generate the data it is currently responsible for.
    - Can you find the similarities between the soft k-Means algorithm and EM algorithm with shared covariance $\frac{1}{\beta}I$?
    - Both rely on alternating optimization methods and can suffer from bad local optima.

Further Discussion

- We assumed that the covariance of each Gaussian I was to simplify the math. This assumption can be removed, allowing clusters to have different spatial spreads. The resulting algorithm is still very simple.
- Possible problems with maximum likelihood objective:
    - Singularities: Arbitrarily large likelihood when a Gaussian explains a single point with variance shrinking to zero
    - Non-convex
- EM is more general than what was covered in this lecture. Here, EM algorithm is used to find the optimal parameters under the GMMS.

GMM Recap

- A probabilistic view of clustering. Each cluster corresponds to a different Gaussian.
- Model using latent variables.
- General approach, can replace Gaussian with other distributions (continuous or discrete)
- More generally, mixture models are very powerful models, i.e., universal distribution approximators.
- Optimization is done using the EM algorithm.

Lecture 12) Decision Trees:

- Depending on question: the root of decision tree should hold the most information regarding what the question asked:
- Ex:
    - Question: should i pick a cat or dog as a pet
        - Then from their its yes or no questions

Animal

Cat Dog

Internal nodes : test attributes
Branching is determined by attribute value
Leaf nodes are outputs (predictions)

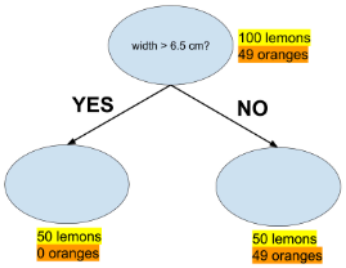
Discrete Trees:
-boolean and can be expressed in truth table
Continuous Trees:
-Can approximate any function arbitrarily closely

Classification Tree:
-Discrete output
Regression Tree:
-continuous output

Greedy Heuristic:
1) Start with empty tree
2) Split on “best attribute” which means get the one with most value depending on question
3) Recursively do the bs on subpartitions
4) We stop when we successfully have reached a conclusion that we need

How to choose “best”:
- Loss: misclassification error
- Accuracy Gain is L(R) = $\frac{|R_1|L(R_1)+|R_2|L(R_2)}{|R_1|+|R_2|}$

Example of AG:



Is this split good? Zero accuracy gain

$$L(R) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|} = \frac{49}{149} - \frac{50 \times 0 + 99 \times \frac{49}{99}}{149} = 0$$

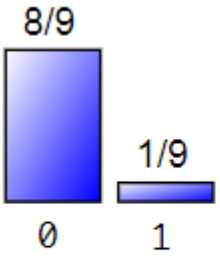
But we have reduced our uncertainty about whether a fruit is a lemon!

How da fuck do you calculate uncertainty?

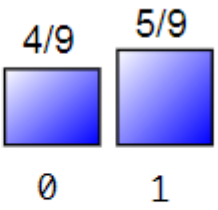
Entropy: measure of expected surprise or how uncertain we are drawing that value

$$H(X) = -\mathbb{E}_{X \sim p}[\log_2 p(X)] = -\sum_{x \in X} p(x) \log_2 p(x)$$

Example:



$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2}$$



$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

Entropy unit is **BITS,**

Fair coin flip is 1 bit of entropy

High Entropy

- Variable is like a uniform distribution
- Flat histogram
- Values sampled are less predictable

Low Entropy

- Distribution of variable has peaks and valleys
- Histogram has lows and highs
- Values sampled from it are more predictable

Calculation of joint distribution:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 p(x,y)$$

Condition entropy:

$$H(Y|X = \text{raining}) = -\sum_{y \in Y} p(y|\text{raining}) \log_2 p(y|\text{raining})$$

Discrete Case: Conditional Entropy:

- H is alway non negative
- Many more on slide 23

INFORMATION GAIN (IG)

- **How much info gain in Y due to X (mutual info of Y and X)**

- If X is completely uninformative about Y : IG(Y|X) = 0
- If X is completely informative about Y:IG(Y|X)=H(Y)

Decision Tree Algorithm:

- Simple, greedy, recursive approach, builds up tree node-by-node
- Start with empty decision tree and complete training set
    - Split on the most informative attribute, partitioning dataset
    - Recurse on subpartitions
    -
- Possible termination condition: end if all examples in current subpartition share the same class

Good Tree?

- Not too small: need to handle important but possibly subtle distinctions in data
- Not too big:
    - Computational efficiency (avoid redundant, spurious attributes)
    - Avoid over-fitting training examples
    - Human interpretability“

- Occam's Razor": find the simplest hypothesis that fits the observations
    - Useful principle, but hard to formalize (how to define simplicity?
- We desire small trees with informative nodes near the root

Problems:

- You have exponentially less data at lower levels
- A large tree can overfit the data
- Greedy algorithms don't necessarily yield the global optimum
- Mistakes at top-level propagate down tree

Advantages of decision trees over k-NN

- Good with discrete attributes
- Easily deals with missing values (just treat as another value)
- Robust to scale of inputs; only depends on ordering
- Good when there are lots of attributes, but only a few are
- important
- Fast at test time
- More interpretable

Advantages of k-NN over decision trees

- Able to handle attributes/features that interact in complex ways
- Can incorporate interesting distance measures, e.g., shape contexts.