1 Linear Regression

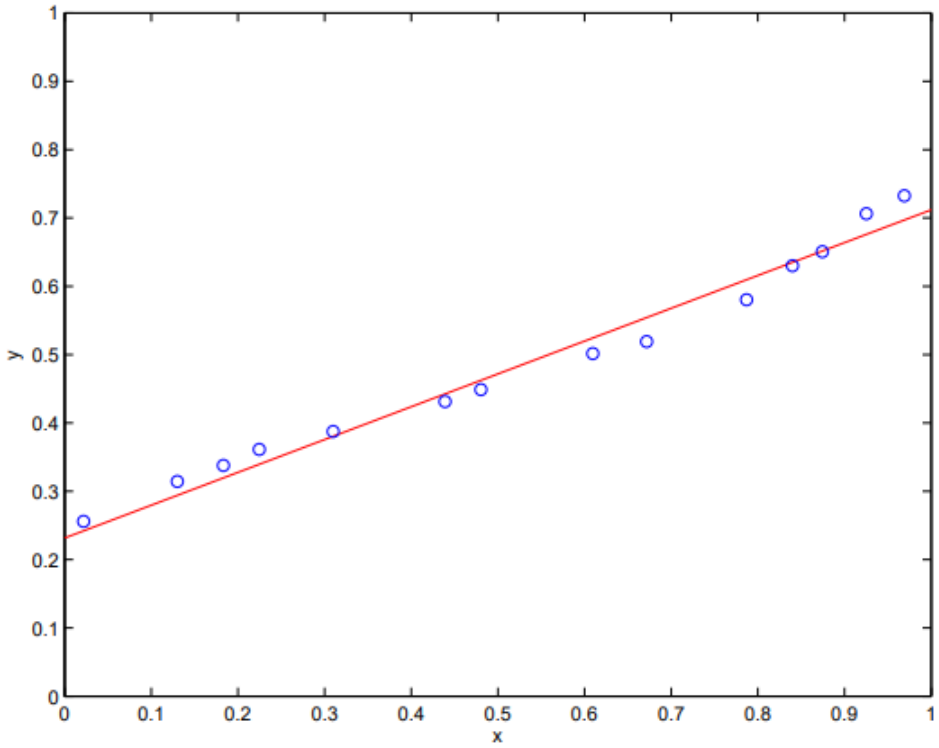
1.1 1 Dimensional Case $x \in R, y \in R$

1.1.1 Motivation

- Given $\{x_i\}_{i=1}^N, \{y_i\}_{i=1}^N$ we want to find a **linear** line

$$f(x) = w \cdot x + b$$

that fits the data best



1.1.2 Learning

- Learn the line by minimizing our **least squares Error Function**

$$E(w, b) = \Sigma(y_i - f(x_i))^2$$

With some work, $\frac{\partial E}{\partial w} = 0, \frac{\partial E}{\partial b} = 0 \implies$

$$b^* = \hat{y} - w\hat{x}$$

$$w^* = \frac{\Sigma(y_i - \hat{y})(x_i - \hat{x})}{\Sigma(x_i - \hat{x})^2}$$

* note that $\hat{x} =$ mean of $\{x_i\}$

1.2 Multi Dimensional Case $x \in R^N, y \in R^N$

1.2.1 Motivation

- Given $\{\boldsymbol{x_i}\}_{i=1}^N, \{\boldsymbol{y_i}\}_{i=1}^N$ we want to find a **linear** line

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

  that fits the data best

1.2.2 Learning

- Learn the line by minimizing our **least squares Error Function**

$$E(\boldsymbol{w}) = \Sigma(y_i - f(\boldsymbol{x}))^2$$

  With some work, $\frac{\partial E}{\partial w_i} = 0 \implies$

$$w^* = (X^T X)^{-1} X^T y$$

  where

$$X = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}$$

1.3 Prediction

- Simply evaluate

$$f(\boldsymbol{x_0}, \boldsymbol{w}^*)$$

  to predict the output

1.4 Notes

- Linear Regression is a good first model to try, but its linear and simple restrictions make it ineffective for complex data

2 Non linear Regression

2.1 Motivation

- Most data can not be modeled with a linear line. We need to model log, expo, ... relationships too
- In Linear regression, we have

$$f(x) = w \cdot x + b$$

  In Non linear regression, we apply a family of basis functions

$$f(x) = \Sigma w_i b_i(x) = w_0 b_0(x) + \cdots + w_k b_k(x) + b$$

2.2 Common of Basis

2.2.1 Polynomial

$$b_0(x) = 1 \quad b_1(x) = x \quad b_2(x) = x^2 \quad b_3(x) = x^3 \quad b_4(x) = x^4 \quad \dots$$

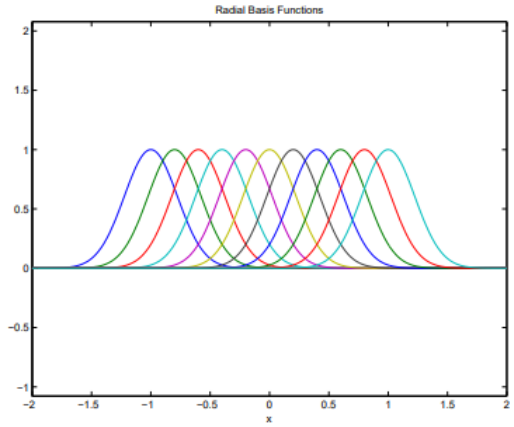
- using polynomial basis functions, we can model any **polynomial curve**

2.2.2 Radial Basis functions

$$b_k(x) = exp(-\frac{(x - c_k)^2}{2\sigma^2})$$

creates a normal curve with $\mu = c_k$ and $variance = \sigma^2$

- c_k, σ^2 are **Hyper-parameters** that must be decided ahead of time
- Its common for c_k to be at **every data point**
- Its common for σ^2 to be **median of nearest neighbour**
- * see more methods to pick parameters on page 12



2.3 Learning

- The Error function we want to minimize is the **sum of squared residual errors**

$$E(\boldsymbol{w}) = \Sigma_i(y_i - \Sigma_k w_k b_k(x))^2 = ||\boldsymbol{y} - B\boldsymbol{w}||^2$$

with some work, $\frac{\partial E}{\partial \boldsymbol{w}} = 0 \implies$

$$\boldsymbol{w}^* = (B^T B)B^{-1}\boldsymbol{y}$$

where

$$B = \begin{bmatrix} b_1(x_1) & \dots & b_K(x_1) \\ \vdots & \ddots & \vdots \\ b_1(x_N) & \dots & b_K(x_N) \end{bmatrix}, B \text{ is } N \times K$$

2.4 Prediction

- Simply evaluate

$$f(\boldsymbol{x_0}, \boldsymbol{w}^*)$$

to predict the output

2.5 Regularization (l2 / ridge)

- we might get fantastic training accuracy but poor test accuracy, this is due to over fitting the data, such that we fit the **noise** as well.

  We can solve this by adding an additional regularization term to our Error function, such that we penalize **non smoothness / high weight values**.

$$E(\boldsymbol{w}) = ||\boldsymbol{y} - B\boldsymbol{w}||^2 + \lambda||\boldsymbol{w}||^2$$

  $||w||^2 \longrightarrow \infty \implies$ more non smooth
  λ (hyper-parameter) controls how much we penalize

  - under-fitting: λ needs to be increased.
  - over-fitting: λ needs to be decreased.

- with some work, $\frac{\partial E}{\partial \boldsymbol{w}} = 0 \implies$

$$\boldsymbol{w}^* = (B^T B + \lambda \boldsymbol{I})B^{-1}\boldsymbol{y}$$

2.6 Notes

- Basis functions must not linearly independent of each other, or else a ∞ set of weights is valid

3 K-nearest Neighbours Regression

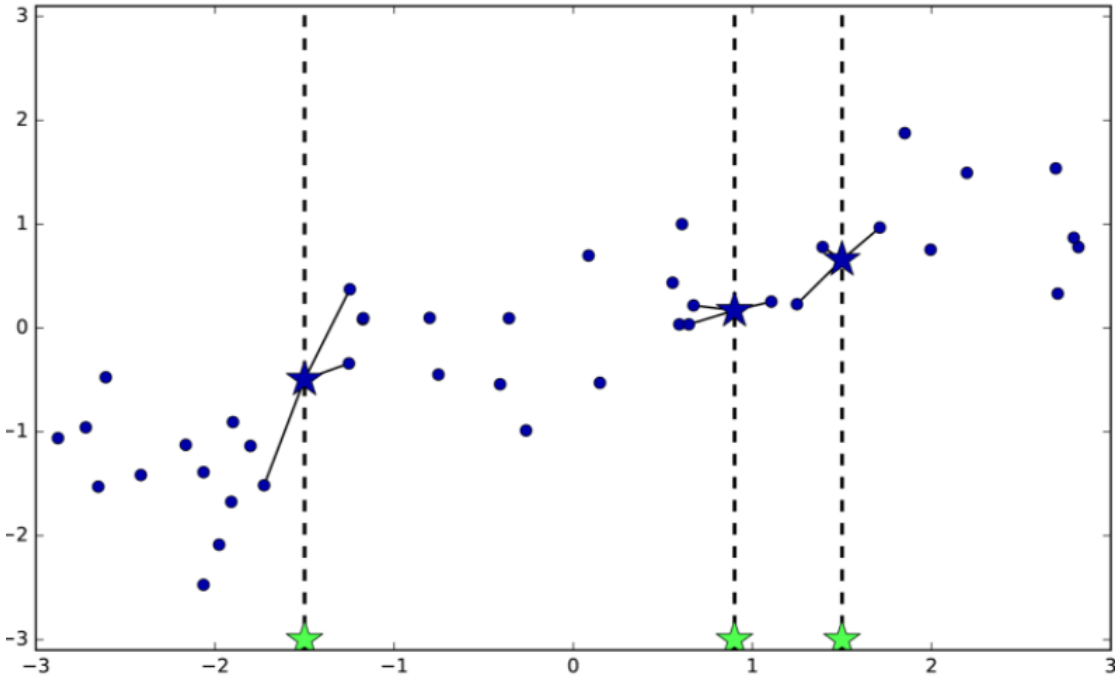
3.1 Motivation

- Why not cut out the middleman?
  Use the data as the model it self. **no training or hyper-parameters needed**.

3.2 Prediction

- let $N_k(x)$ be set of k neighbours closest to index x_0.

$$y(x_0) = \frac{1}{K}\Sigma_{i \in N_k(x)} y_i$$



3.3 Notes

- This requires us to keep the entire data set to predict
- Although no training is needed, each prediction is very **computationally heavy**

4 Probabilistic Parameter Estimation

4.1 Motivation

- Using probability and Bayes rule to estimate parameters.
  Opens up doors to model selection, confidence ...

4.2 Bayes Rule

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

- $P(\theta|D)$: **Posterior**, Probability of Parameters Given Data
- $P(D|\theta)$: **Likelihood**, Probability of Data Given Parameters
- $P(\theta)$: Prior, Probability of Parameters **(mostly uninformative)**
- $P(D)$: Evidence, Probability of Data **(mostly uninformative, just for normalization)**

4.3 Estimation Techniques

4.3.1 Maximum a posterior (MAP)

$$\theta_{MAP} = argmin_\theta(-ln(P(\theta|D))) \tag{1}$$
$$= argmin_\theta(-ln(\frac{P(y_{1:N}|x_{1:N}, \mathbf{w})p(\mathbf{w})}{P(y_{1:N}|x_{1:N})})) \tag{2}$$

- find the most probable theta based on **Posterior**.
  Negative Ln is taken for numerical stability

4.3.2 Maximum Likelihood (ML)

$$\theta_{ML} = argmin_\theta(-ln(P(D|\theta))) \tag{3}$$
$$= argmin_\theta(-ln(\Pi_i pdf_\theta(x_i))) \tag{4}$$
$$= argmin_\theta(-\Sigma_i ln(pdf_\theta(x_i))) \tag{5}$$

- find the most probable theta based on **Likelihood**.
  Negative Ln is taken for numerical stability
- Mostly ML is used as we can split it into a **product** of the **PDF**
- We simply set the derivative to 0 to find θ_{ML}

4.4 Notes

- See **Assignment 1 question 3 a, c** for worked example for **MAP and ML**
- In many cases, the posterior is simply the likelihood with a prior assumption. Usually, this prior is uninformative, and

$$Posterior = Likelihood$$

5 Classification

5.1 Concept

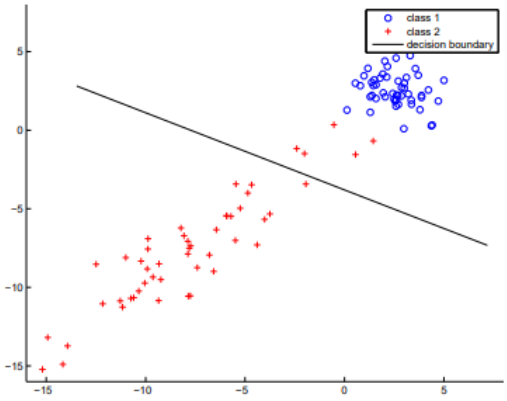
- for $\{x_i\}_{i=1}^N$, $y_i \in \{c_1, ..., c_k\}$
  in other words, regression has **continuous** put, Classification has **discrete** output

5.2 Objective

- The General Goal is to learn the Decision Boundary

$$a(\boldsymbol{x}) = 0$$

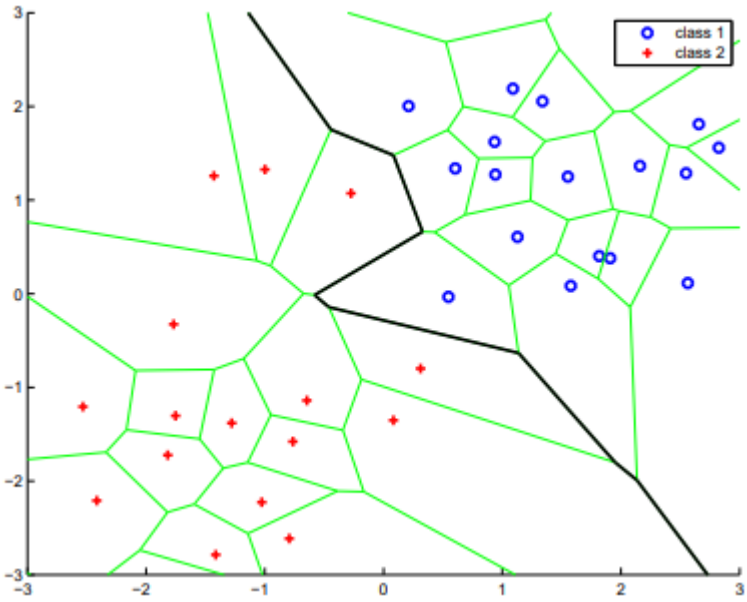
One side of the boundary is **class A**, other side is **class B**



6 k-Nearest Neighbors Classification

6.1 Motivation

- Use the idea of KNN
  Average **class of neighbours** is **me**



6.2 Prediction

$$y_0 = sgn(\Sigma_{i \in N_k(\boldsymbol{x_0})} y_i)$$

Since Class A is 1 and Class B is -1, we can simply take the **sign of the sum**
Furthermore, we can improve this prediction with a **weighted Gaussian average**

$$y_0 = sgn(\Sigma_{i \in N_k(\boldsymbol{x_0})} w(\boldsymbol{x_i}) \cdot y_i), \quad w(\boldsymbol{x_i}) = exp(-||x_i - x_0||^2/(2\sigma^2))$$

Note that σ^2 is an addition parameter

6.3 Notes

- Decision boundary is always **piece-wise linear**
- As we increase K, we **effectively smooth the decision boundary** and **generalize** better.
- Like KNN regression, we need to **retain** all **training** data,
  - storage and computation is expensive in higher dimensional features

7 Decision Trees

7.1 Motivation

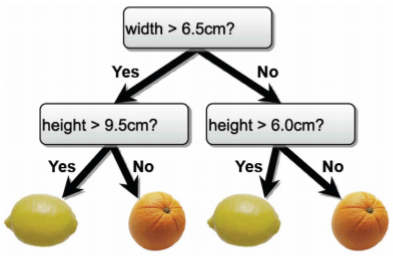
- A series of **questions** to ask our data, in the end we arrive at a answer / **class**
- we have N internal / split nodes, and $N-1$ terminal / leaf nodes
  At each split node, we perform a **binary test** to decide whether to continue down the **left node or right**

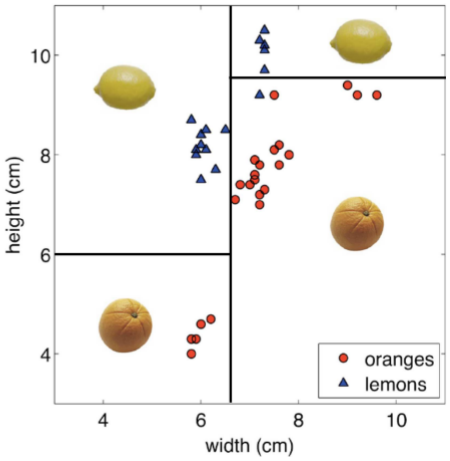
$$t_j(\boldsymbol{x}) : R^d \longrightarrow \{-1, 1\}$$

  Finally, we have terminal / leaf nodes to classify the input

$$P(y = c|j) = \frac{N_{j,c}}{N_j} = \frac{\text{Data in node } j \text{ with class } c}{\text{All data in node } j}$$

  Ideally, this probability is 1, meaning we are sure of 1 class





7.2 Learning

- Objective is to find all **internal and terminal nodes**.
  This task is NP-hard, we will **recursively generate down** the tree, making the best choice at each branch.
- First, we decide if the current node should be a leaf, if almost all points are same class, we can terminate.
- Secondly, In a internal node, we want the children nodes to be simpler **(non zero probability for a single class)**.

We measure this using **Information Gain**

$$IG(D_j, t_j) = H(D_j) - \frac{N_L}{N_j} H(D_L) - \frac{N_R}{N_j} H(D_R)$$

- $H(D_j)$: Entropy of Data in Node j. Decreases when a **single class is likely** / more information on class.

$$H = -\Sigma_{c=1} P(y = c) log(P(y = c))$$

  - $H = 0 \implies$ No uncertainty
  - $H = log_2(k) \implies$ Max uncertainty
- $\frac{N_{L,R}}{N_j}$: $\frac{\text{data points in L/R node}}{\text{Total data points in parent node}}$, This value is to normalize the Entropy
-

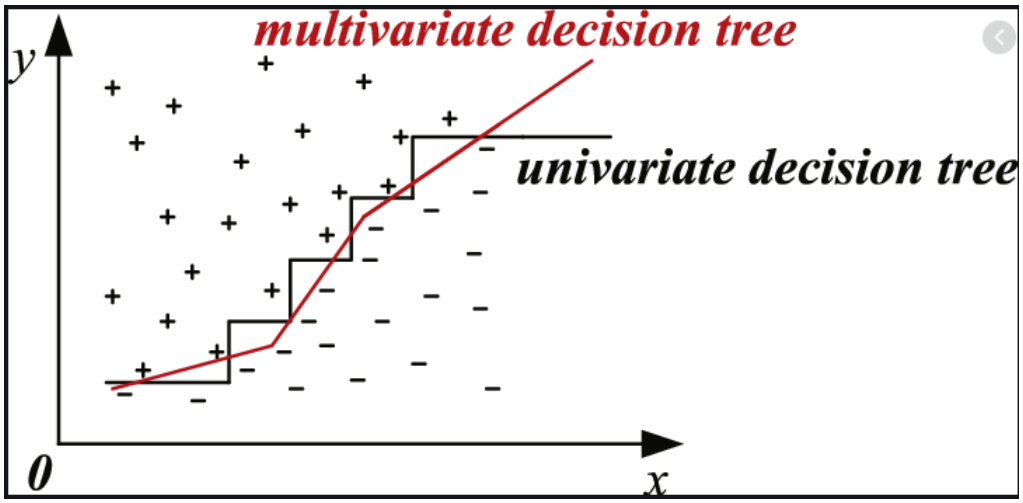
$$\text{Information Gain} = \text{Total Entropy - Left node entropy - right node entropy}$$

- $argmax_{t_j} IG(D_j, t_j)$ is found by trying every t_j. Given d dimensions, there are

$$d(N_j - 1)$$

possible candidates.

7.3 Uni-variate vs multivariate splits



- **Uni-variate splits** give a **stepping** Decision Boundary, with

$$d(N_j - 1)$$

candidates per node

- **multivariate splits** give a **non linear Decision** Boundary, with Exponential candidates
- * **Uni-variate wins** on training time alone.

7.4 Decision Forests

- Using Uni-variate splits are pretty bad predictors.
  We can improve using **many trees (a forest)**, each using a **subset of data** and a **subset of features**.
  We can then **average the predictions (Bagging)** of all the trees to a **single** prediction.

7.5 Prediction

- Simply run the point through the tree.

8 Class Conditionals

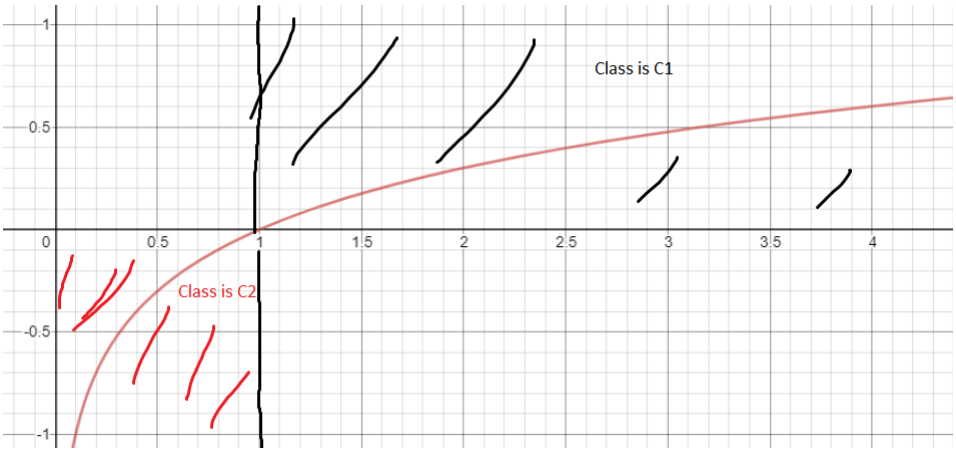
8.1 Motivation

- Lets assume the classes of data are generated by a **statistical model** (Gaussian, ...).
  Once we find these models, we can simple ask for the class with the **largest probability**

$$argmax_c(P(c|\boldsymbol{x}))$$

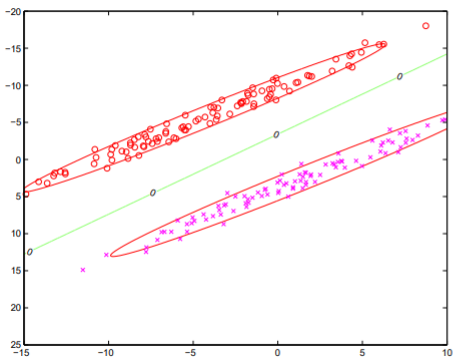
With 2 Classes,

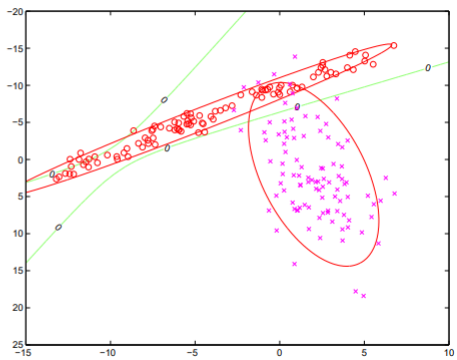
$$a(\boldsymbol{x}) = log(\frac{P(\boldsymbol{x}|c_1)P(c_1)}{P(\boldsymbol{x}|c_2)P(c_2)})$$



And the Decision boundary is simply

$$a(\boldsymbol{x}) = 0$$





8.2 Training

- For every class, we find its **Prior** and **Likelihood**

$$P(c_i) \quad P(\boldsymbol{x}|c_i)$$

Using MLE on the training set.

8.3 Prediction

- Simply evaluate

$$argmax_c(P(c|\boldsymbol{x}))$$

Assuming Gaussian, this simplifies to

$$a(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\mu_1)^T\Sigma_1^{-1}(\boldsymbol{x}-\mu_1) - \frac{1}{2}ln|\Sigma_1| \tag{6}$$

$$+\frac{1}{2}(\boldsymbol{x}-\mu_2)^T\Sigma_2^{-1}(\boldsymbol{x}-\mu_2) + \frac{1}{2}ln|\Sigma_2| \tag{7}$$

8.4 Notes

- This is a **Generative** model, as we can **generate** new data
- In the **Gaussian** case, we need alot of data to accurately train co-variance matrix Σ
  - Size of Σ is (quadratic w.r.t feature space)

$$d(d+1)/2$$

  - if cov are equal, the Decision boundary is **Planer**
  - else, The boundary is **conic section (quadratic)**
  - this **Cov** comes from the fact the **features** of the training data is **not IID**

9 Naive Bayes

9.1 Motivation

- The **Covariance** in CCM is so hard to calculate, lets set it to 0 by assuming the **features** of a data point are **IID**.
  So instead of estimating **1 d-dimensional multivariate density**, we estimate **d 1-dimensional uni-variate density**

  Luckily, each uni-variate Gaussian has **2 parameters** (μ, σ^2)
  We effectively reduce a $d \times d$ covariance matrix to $2d$ parameters, or a $d \times d$ **diagonal** matrix.
  * **Each class has d density functions, for each feature.**

9.2 Training

Suppose the features are **discrete (1, 0)**

- Since we deal with each feature of a vector, let

$$\boldsymbol{x} = F_{1:d}$$

To train the model, we just need to find all the uni-variate density functions.

$$P(c = j), \forall j \quad P(F_k = 1|c = j) \forall k, j$$

Since features are discrete,

$$P(c = j) = \frac{N_j}{N} = \frac{\text{number of data points with class j}}{\text{all data points}}$$

$$P(F_k = 1|c = j) = \frac{N_{kj}}{N_j} = \frac{\text{number of data points with class j with feature k}}{\text{number data points with class j}}$$

with some work,

$$\implies P(c = j|F_{1:d}) = \Pi_{k=1}^{d} P(F_k = 1|c = j)^{F_k}(1 - P(F_k = 1|c = j))^{1-F_k}$$

9.3 Prediction

- Find the

$$argmax_j P(c = j|F_{1:d})$$

9.4 Note

- Since we are effectively using the diagonal of the covariance matrix, Naive Bayes is less expressive than Class Conditional
- When we have many features with small data set, its possible a **feature never occurs on a class**, and therefor the likelihood is 0. We correct this problem by adding regularization.

$$P(c = j) = \frac{N_j + \beta}{N + K\beta}$$

$$P(F_k = 1|c = j) = \frac{N_{kj} + \alpha}{N_j + 2\alpha}$$

  - Now, **no previous occurrence** translates to **no knowledge**, instead of **impossible**

10 Logistic Regression

10.1 Motivation

- With some simple manipulation,

$$P(y = c_1|\boldsymbol{x}) = \frac{1}{1 + e^{-a(\boldsymbol{x})}} \qquad a(\boldsymbol{x}) = ln(\frac{P(\boldsymbol{x}|c_1)P(c_1))}{P(\boldsymbol{x}|c_2)P(c_2)}) \tag{8}$$

  Note that this function is a Sigmoid

  - $a(\boldsymbol{x}) \longrightarrow -\infty, P = 0$
  - $a(\boldsymbol{x}) \longrightarrow \infty, P = 1$
  - $a(\boldsymbol{x}) \longrightarrow 0, P = 0.5$

    The decision Boundary, is the **linear line**

$$a(\boldsymbol{x}) = 0$$

  If we **assume Gaussian Likelihoods** with same covariance, a becomes a **linear** function!

$$P(y = c_1|\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}} \tag{9}$$

  This is just binary class, for multi class, we should have a weight vector for each class
  * This is **the softmax function**, which converts K score value to probability

10.2 Learning

- The goal is to find the optimal weight

$$\boldsymbol{w}^*$$

  We minimize the **likelihood**

$$L = P(\{x_i, y_i\}|\boldsymbol{w})$$

  For Binary Class, the **Gradient** simplifies to

$$\frac{\partial(-ln(L))}{\partial \boldsymbol{w}} = -\Sigma_{i=1}^{N}(y_i - P(c_i|\boldsymbol{x}, \boldsymbol{w}))$$

  Unfortunately this gradient does **not have a close form**, we use **Gradient Decent** to find a good $\boldsymbol{w}^*$

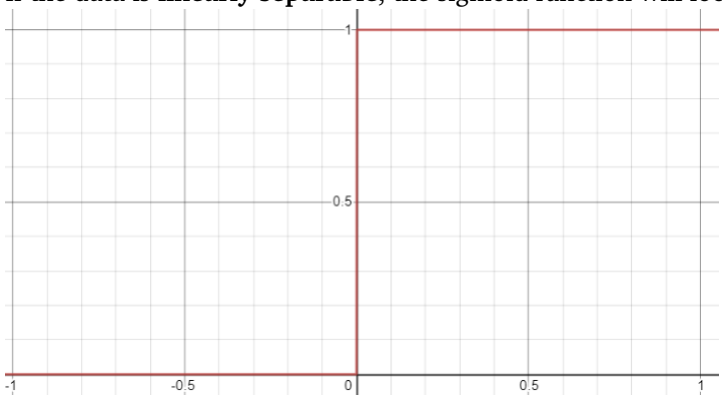
10.3 Prediction

- Simply find

$$argmax_{c_i}(P(y = c_i|\boldsymbol{x}))$$

10.4 Regularization

- If the data is **linearly separable**, the sigmoid function will look like a step function.



This will result in $\boldsymbol{w}^*$ getting large to the point of **instability**.

This can be solved by adding a weight decay prior on $\boldsymbol{w}$, such as adding

$$2\lambda \boldsymbol{w}$$

to the gradient

10.5 Notes

- Even if our Gaussian assumption is wrong, assuming Gaussian and learning to the best of our abilities will be good enough for classification.
- This whole time, we are learning

$$P(y = c_1|\boldsymbol{x})$$

thats why LR is Discriminative

- Remember that Logistic regression has a **linear decision boundary**, it can only fit as well as a linear planer could.

11 Bayesian Methods

11.1 Motivation

- In a frequentest approach, we only predict the output by maximizing one single correct value. We disregard any uncertainly.
- In Bayesian Methods, we always have bayes rule in mind, and try to incorporate distributions as much as possible.

Using bayes rule,

$$P(y_{new}|D, x_{new}) = \int P(y_{new}|\boldsymbol{w}, x_{new})P(\boldsymbol{w}|D)d\boldsymbol{w}$$

  - There is an interesting **balance** going on here, to achieve a high value, the **weight must be probable for training set (Prior)**, as well **perform well** on $\{x_{new}, y_{new}\}$
  - *** what we are really doing is, averaging all possible models, weighing each model according to their posterior probability**

This one probability distribution tells us everything we need to know, this model combines intrinsic uncertainty from $P(y_{new}|\boldsymbol{w}, x_{new})$, and uncertain of $\boldsymbol{w}$ from $P(\boldsymbol{w}|D)$.

However, in practice this integral is not possible to compute, we can user **Monte Carlo** sampling to compute a approximation

11.2 Prediction

- Knowing $P(y_{new}|D, x_{new})$, we can find
  - most likely prediction, i.e., $argmax_y P(y_{new}|D, x_{new})$
  - we can calculate variance and confidence in our prediction
  - we can sample a prediction, taking in account all uncertainty

11.3 Notes

- * see an example of Bayesian regression here page 73

12 Model Selection

12.1 Motivation

- How do we decide which model is right for the problem? We could run all of them one at a time and compare accuracies, but this is often impractical. We can use Bayesian Methods to select a optimal model by prioritizing simplicity of the model.

  We can Gauge a Models effectiveness using likelihood,

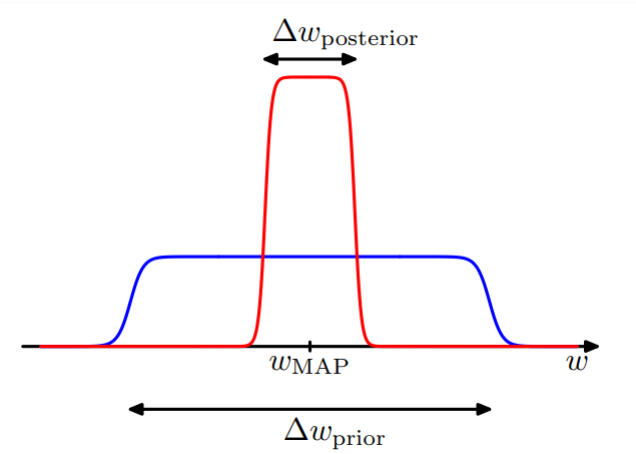
$$P(D|M_i) = \int P(D|\boldsymbol{w}, M_i)P(\boldsymbol{w}|M_i)d\boldsymbol{w}$$

  - Note that for a good likelihood, we need **fitting performance**, and parameters are **probable under the Model (Prior)**

  It turns out,

$$P(D|M_i) \approx P(D|w^{MAP}, M_i)P(w^{MAP}|M_i)\Delta w^{posterior} \quad \text{Works if we assume the peak is STRONG} \tag{10}$$

$$= P(D|w^{MAP}, M_i)\frac{\Delta w^{posterior}}{\Delta w^{prior}} \quad \text{Assume Prior is somewhat uniform} \tag{11}$$



Since $w^{posterior} < w^{prior}$, the best case is we minimize w^{prior},Consequently, this also implies a lower complexity.

*** By selecting a model with the highest marginal data likelihood we are automatically balancing model complexity with the ability of the model to capture the data. This can be seen as the mathematical realization of Occam's Razor.**

13 Generative vs Discriminative

There are 2 types of Models, **Generative** and **Discriminative**

13.1 Generative

13.1.1 Difference

- Attempts to model **complete probability** of training data

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

  Often, this is formulated as

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

- *** the model can be used to generate new data, to classify noise inputs, outliers ...**
  It knows priors ex. if a x value is unlikely
- The trade of is that the classification performance is lesser.
- We can use prior to inject model with more
  - Ex. Data is not **Gaussian** (GCC)

13.1.2 Examples

- Class Conditional
- Gaussian Class Conditional
- Naive Bayes

13.2 Discriminative

13.2.1 Difference

- Attempts to model / maximize

$$p(y|\boldsymbol{x})$$

  has no idea of distribution of $P(x)$ **(Priors)**

- * this model is **optimized** for **classification**, can't do anything else.
- More Generic and efficient, hard to specialize to a problem **(due to no prior)**

13.2.2 Examples

- Logistic Regression
- Tree / Forest
- K-NN

13.3 Note

- Logistic Regression vs GCC is a good example of Discriminative vs Generative.
  Thsey both have linear boundaries, but LR performs better on non Gaussian and in general, while GCC can generate data.

14 Dimension Reduction

14.1 Motivation

High dimensional Data is extremely hard to work with, here are some reasons

- Visualization : High dimensional data is very hard to visualize.(see disjointedness of data)
- Pre-processing : learning a high dimensional feature space is difficult since number of parameters grow, learning is slow and prone to over-fitting.
- Sparsity: data with many dimensions may be very sparse (most of the features are 0), this makes training very hard, we can condense the data using Dimension reduction techniques.

15 Principle Component Analysis

15.1 Motivation

- were given a set of data

$$\{\boldsymbol{y}_i\}$$

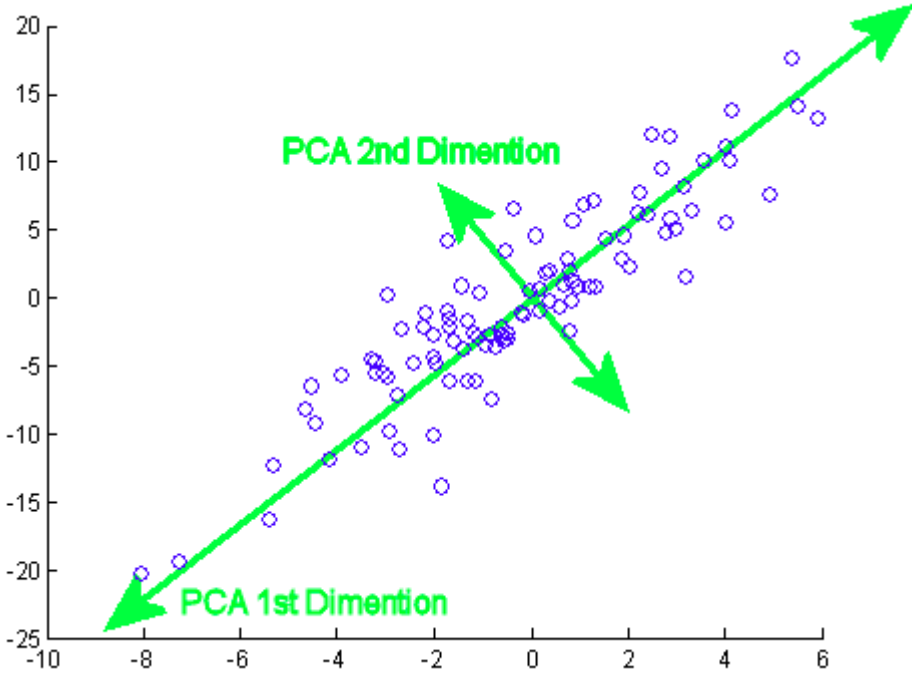
  we want to find a lower representation of this data

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

  where $\boldsymbol{x}$ is lower dimension than $\boldsymbol{y}$, and $\boldsymbol{W}$ is orthonormal. In this case $\boldsymbol{x}$ will be our new simpler data.

all we need to do is find this converter matrix $\boldsymbol{W}$.

What we are effectively doing, is assigning each column of W to be a new axis, a principle component.



15.2 Learning

It turns out, we can use the following algorithm to find everything

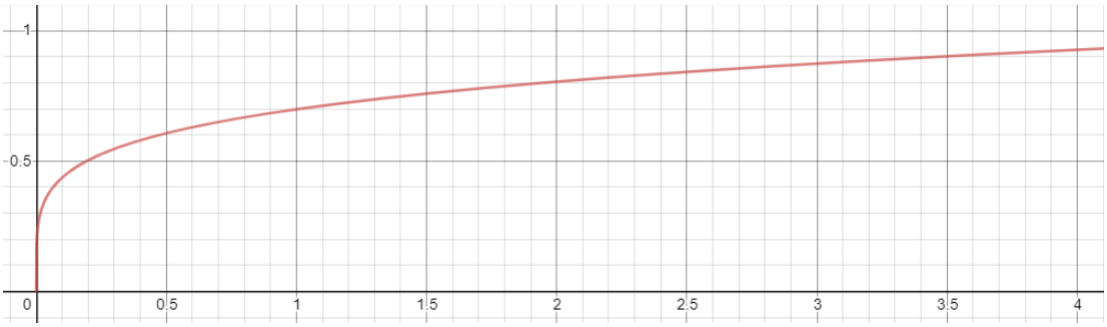
- Step 1 : let
$$\boldsymbol{b} = \frac{1}{N}\Sigma \boldsymbol{y_i}$$
- Step 2 : find covariance
$$C = \frac{1}{N}\Sigma(\boldsymbol{y_i} - \boldsymbol{b})(\boldsymbol{y_i} - \boldsymbol{b})^T$$
- Step 3 : find eigen decomposition of C
$$C = V\Lambda V, \quad \Lambda = diag(\lambda_1, ..., \lambda_d), \quad V = [V_1, ..., V_d]$$
- Step 4 : sort eigen value from biggest to smallest, then adjust eigen vectors based on the reordering
- Step 5 : choose lower dimension, c,
$$W = [V_1, ..., V_c]$$
- Step 6: Get all reduced data
$$\boldsymbol{x_i} = \boldsymbol{W}^T(\boldsymbol{y_i} - \boldsymbol{b})$$

15.3 How to choose c

- We want to select a good dimension to cast down too, such that its low, but we dont lose too much information.
  Turns our, eigen-vectors carry the **variance** of its eigen-vector, and we want to save as much variance

as possible.
Lets graph

$$\frac{\Sigma_{i=1}^{c}\lambda_i}{\Sigma_{i=1}^{d}\lambda_i} = \frac{\text{Variance of c dims}}{\text{total variance}}$$



This is why we sorted λ/variance from biggest to smallest, so we will always choose the eigen-vectors with most variance.

To choose a c, simply choose the smallest c that gives 95% of total variance!
In the above graph, 4 is a good c.

16 Probabilistic PCA

16.1 motivation

- We can also take an Bayesian approach to PCA, lets assume
  - $\boldsymbol{x} \sim N(0, \boldsymbol{I})$
  - $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} + \eta$, η is the uncertainty

  Then,

$$P(\boldsymbol{y}) = \int P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x})d\boldsymbol{x} \tag{12}$$

$$= \int G(\boldsymbol{y}; \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \sigma^2\boldsymbol{I})G(\boldsymbol{x}; 0, \boldsymbol{I}) \qquad \text{Implying } P(\boldsymbol{y}) \text{ is Gaussian} \tag{13}$$

with some work,

$$P(\boldsymbol{y}) \sim N(\boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}} + \sigma^2\boldsymbol{I})$$

Remember that the dimensionality of a Guassian distribution lies on the **dimensionality of its covariance matrix**, if we can shrink the covariance, we're effectively **reducing dimension**.

16.2 Learning

- We can learn this model by using Maximum Likelihood

$$L(\boldsymbol{W}, \boldsymbol{b}, \sigma^2) = -ln\Pi G(y_i; \boldsymbol{b}\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}} + \sigma^2\boldsymbol{I})$$

Unsurprisingly, the solution is very similar to learning ordinary PCA.

16.3 Notes

- as $\sigma \longrightarrow 0, PPCA \longrightarrow PCA$

- PPCA gives us a generative model, having learned the distribution of the data, we can easily sample more.

17 Clustering

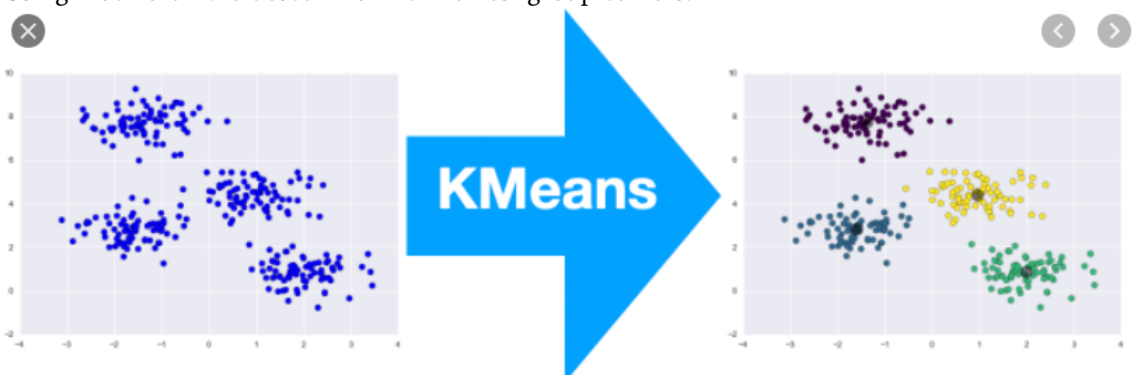
17.1 Motivation

- we have lots of data, we want to group them into clusters (topic, type ...)
- different than classification, in that we dont know the clusters before hand. The clusters themselves are learned.
- Clustering is unsupervised.

17.2 K-means

17.2.1 Motivation

- Find clusters by represented by the cluster centers.
- Using block ordinate descent to find the best group centers.



18 Gaussian Mixture Models

19 Techniques In Practice

19.1 Cross Validation

19.2 Gradient Descent

19.3 Monte Carlo Methods

19.4 Lagrange Multipliers