Machine Learning and Intelligent Systems
Maria A. Zuluaga

Introduction

What's machine learning?

Learning from experience (data) to be able to: make predictions, find similarities, find patterns, discover knowledge, etc. There are 3 types of learning:

- Supervised: learn from labeled data
  - Regression: continuous functions
  - Classification: separate data
- Unsupervised: no target or label
- Reinforcement: take actions to maximize rewards

PART I: Supervised Learning

To predict y using x without knowing the true relationship between them: $y = f(\boldsymbol{x}) \in \mathbb{R}$.

Training set (input-output set): $\{(x_i, y_i), i = 1, \dots, N\}$

Goal: finding a good predictor g such as $\hat{y} = g(\boldsymbol{x}) \simeq y$ for unseen $(\boldsymbol{x}, y)$ pairs.

CHAPTER 1: Linear Models for Regression (Week 1)

A linear regressor is a linear function of x: $y = f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum w_j x_j$ where $\boldsymbol{w} = \{w_j\}$ are the parameters.

Model fitting: estimate the modal's parameters $\hat{y} = g(\boldsymbol{x}) = \widehat{w}_0 + \sum \widehat{w}_j x_j$

We introduce a **loss function** to quantify how well/bad $\hat{y}$ approximates y, e.g. $\ell(y, \hat{y}) = (y - \hat{y})^2$

We use the loss function to compute average loss over all the data $\mathcal{L}_w = \frac{1}{N}\sum \ell(y, \hat{y})$ and $\underset{\mathrm{w}}{\operatorname{argmin}}\, \mathcal{L}_w$

A discriminant interpretation: <u>least squares</u>

Optimization: Least squares is a method for estimating the unknown parameter w of a linear model by minimizing the sum of the squares of the differences between the observed output y_i and those predicted by the model $\hat{y}_i$: $\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}_{\boldsymbol{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N}\sum (y_i - \hat{y}_i)^2 = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N}\sum \big(y_i - (w_0 + w_1 x_i)\big)^2$ (Here $x \in \mathbb{R}^1$).

Let's introduce the matrix notation:

$\boldsymbol{X} = \begin{pmatrix} 1 & \cdots & x_{1D} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_{ND} \end{pmatrix}$, $\boldsymbol{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_D \end{pmatrix}$ and $\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$ where N is the number of samples and D the dim of each x_i.

Using those notations, we get $\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}_{\boldsymbol{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$ and $\frac{\partial \mathcal{L}_w}{\partial \boldsymbol{w}} = 0 \Rightarrow \widehat{\boldsymbol{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$.

Predictions: now we can use estimated $\widehat{\boldsymbol{w}}$ to predict new values: $\widehat{\boldsymbol{y}}_{\boldsymbol{new}} = \boldsymbol{X}_{\boldsymbol{new}}\widehat{\boldsymbol{w}}$ where $\boldsymbol{X}_{\boldsymbol{new}}$ unseen data.

"All models are wrong, but some are useful."

We can add polynomial features $\hat{y} = w_0 + w_1 x_1 + \cdots + w_n x^n$ and this is still considered to be linear model.

A probabilistic interpretation: maximum likelihood estimation

Additive error model: $y = f(x, \boldsymbol{w}) + \epsilon$, where the error ϵ:

- Can be positive or negative
- Is independent for each x
- Can be modeled as a continuous random variable that follows a Gaussian distribution

Density function of ϵ: $p(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right) \sim N(0, \sigma^2)$

Density function of y: $p(y_i|\boldsymbol{x_i}; \boldsymbol{w}, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2}{2\sigma^2}\right) \sim N(\boldsymbol{w}^T\boldsymbol{x_i}, \sigma^2)$

Likelihood function: $L(w, \sigma) = \prod p(y_i|\boldsymbol{x_i}; \boldsymbol{w}, \sigma^2)$ which is NOT a probability.

We maximize $L(w, \sigma)$ or $\log L(w, \sigma)$ because it's simpler.

$$\ell(\boldsymbol{w}, \sigma^2) = \log L(w, \sigma) = \log \prod p(y_i|\boldsymbol{x_i}; \boldsymbol{w}, \sigma^2) = \cdots = -N \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum (y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2$$

$\frac{\partial \log L}{\partial \boldsymbol{w}} = 0 \Rightarrow \widehat{\boldsymbol{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$ and $\frac{\partial \log L}{\partial \sigma} = 0 \Rightarrow \hat{\sigma}^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$

Prediction: to predict new values, we use $\widehat{\boldsymbol{y}}_{\boldsymbol{new}} = E[y_{new}|\boldsymbol{X}; \boldsymbol{w}, \sigma^2] = \widehat{\boldsymbol{w}}^T\boldsymbol{x}_{new}$.

σ measures the degree of uncertainty of our prediction.

CHAPTER2: Linear Models for Classification (Week 2)

Probability Reminders:

1- Sampling from a Gaussian with mean m and the adding a constant C is the same as sampling from a Gaussian with mean $m + C$.

2- Two random variables are independent if $p(A = a, B = b) = p(A = a)p(B = b)$.

3- Join probability: for two discrete random variables X and Y, $P(X = x, Y = y)$ is the probability that X has value x and Y has value y. For continuous random variables, $p(x, y)$ is the join density function (pdf).

4- Conditional probabilities: when variables are independent, we can work with conditioning: $p(y_i|x_i; w, \sigma^2)$. Which allow to decompose join probability (or pdf): $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$ (and $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$).

5- Conditional expectation: expectation is the average of a large number of independent realizations of a random variable: $\mathrm{E}[x] = \sum_x xP(x) = \int xp(x)dx$. Conditional expectation: expected value of y given $\boldsymbol{x}$: $\mathbb{E}[y|x] = \int yp(y|x)dy$.

6- Covariance: joint variability of two variables: $cov(X, Y) = \mathbb{E}\big[X - \mathbb{E}[X]\big]\mathbb{E}\big[Y - \mathbb{E}[Y]\big] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$

Classification

When the output is discrete, y represents labels or classes and we denote $\mathcal{C}$ the set of possible classes.

The main goal is to predict the class $y = c \in \mathcal{C}$ using $\boldsymbol{x}$.

The input space is divided into decision regions whose boundaries are called decision boundaries.

So how to represent classes:

- Binary case: $y \in \{0,1\}$
- Multiple classes: $1 - of - K$ coding scheme: $y = (0 \quad \cdots \quad 1 \quad \cdots \quad 0)^T$

Learning and Expected Loss

The new loss function in a $K \times K$ matrix with $K = |\mathcal{C}|$

Our goal is to minimize $\mathbb{E}[\boldsymbol{L}] = \sum_k \sum_j \int L_{kj} p(\boldsymbol{x}, \mathcal{C}_k) d\boldsymbol{x}$ by minimizing $\sum_k L_{kj} p(\mathcal{C}_k|\boldsymbol{x})$

Minimization problem: $\hat{y}(\boldsymbol{x}) = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \sum_k L_{kj} p(\mathcal{C}_k|\boldsymbol{x}) = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \big(1 - p(\mathcal{C}_c|\boldsymbol{x})\big) = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \big(p(\mathcal{C}_c|\boldsymbol{x})\big)$

Bayes classifier: we classify the most likely class using the conditional discrete distribution (the value of a feature x_i is independent of the value of any other feature $x_{k \neq i}$).

How does that work? (with an example from Wikipedia)

The goal is to compare, for a given sample $\boldsymbol{x}$, $p(\mathcal{C}_k|\boldsymbol{x}) = p(\mathcal{C}_k|x_1, \dots, x_n)$ for each class $\mathcal{C}_k$.

Under the independence assumptions:

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\mathcal{C}_k) p(\boldsymbol{x}|\mathcal{C}_k)}{p(\boldsymbol{x})} = \frac{p(\mathcal{C}_k) p(x_1|\mathcal{C}_k) \dots p(x_n|\mathcal{C}_k)}{\sum_k p(\mathcal{C}_k) p(\boldsymbol{x}|\mathcal{C}_k)} = \frac{p(\mathcal{C}_k) p(x_1|\mathcal{C}_k) \dots p(x_n|\mathcal{C}_k)}{\sum_k p(\mathcal{C}_k) p(x_1|\mathcal{C}_k) \dots p(x_n|\mathcal{C}_k)}$$

NB: the denominator is constant for a given sample $\boldsymbol{x}$, so it can be ignored.

Example: classify whether a given person is a male or a female based on the measured features. Using a Gaussian distribution assumption, we calculate mean and variance of every feature (height, weight, and foot size) for every class (male and female) then test for a given $\boldsymbol{x}$ by comparing posterior male and female:

$$p(male|x) = \frac{p(male) p(height|male) p(weight|male) p(foot\ size|male)}{evidence} =$$

$$\frac{0.5 \times \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(\frac{-(x_1 - \mu_1)^2}{2\sigma_1^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(\frac{-(x_2 - \mu_2)^2}{2\sigma_2^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_3^2}} \exp\left(\frac{-(x_3 - \mu_3)^2}{2\sigma_3^2}\right)}{evidence}$$

Methods of classification

1. **Linear regression**

For a binary class problem: $\boldsymbol{x} \in \mathcal{C}_1$ if $\hat{y}(\boldsymbol{x}) \geq 0$ else $\boldsymbol{x} \in \mathcal{C}_2$.

Where $\hat{y}(\boldsymbol{x}) = \widehat{w}_0 + \sum_{j=1}^{D} \widehat{w}_j x_j = \widehat{\boldsymbol{w}}^T \boldsymbol{x}$

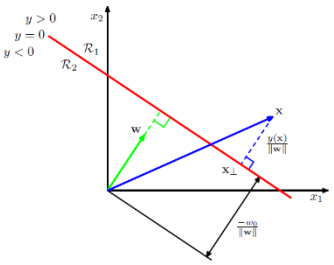
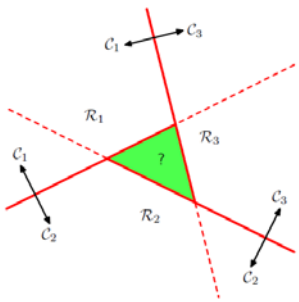


Figure: the decision surface (in red) is orthogonal to $\boldsymbol{w}$ and its displacement from the origin is controlled by w_0. If we consider an arbitrary point $x = x_\perp + r\frac{w}{\|w\|} \Rightarrow$ $w^T x + w_0 = w^T x_\perp + w_0 + r\frac{w^T w}{\|w\|}$ and finally $r = \frac{y(x)}{\|w\|}$ is the signed distance of x from the decision surface.

For a K-class problem: each category is coded via an indicator variable $\boldsymbol{y} = (y_1, \dots, y_K)^T$. Given N the number of total samples, we build the indicator response matrix $\boldsymbol{Y}$ of size $N \times K$.

Algorithm:

- Fit a linear model for each column of $\boldsymbol{Y}$.
- The weight matrix is given by $\widehat{\boldsymbol{W}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$ of dimension $(d+1) \times K$ (coefficient vector for each response column $\boldsymbol{y_k}$).
- A new observation x_{new} is classified by:
  - Computing the fitted output $f(\boldsymbol{x_{new}}) = \boldsymbol{x_{new}}\widehat{\boldsymbol{W}}$ (a K vector).
  - Predict the class corresponding to the highest score $\hat{y}(\boldsymbol{x_{new}}) = \underset{k}{\operatorname{argmax}}\, f(\boldsymbol{x_{new}})$.



Decision boundaries: for any two classes j and k, the decision boundary is a $(d-1)$-dimensional subset in $\mathbb{R}^d$ defined by the values of $\boldsymbol{x}$ satisfying $\boldsymbol{w}_k^T\boldsymbol{x} = \boldsymbol{w}_j^T\boldsymbol{x}$. We get a one-versus-one classifier with $K(K-1)/2$ boundaries.

This approach is problematic for $K > 2$ because we can get ambiguous regions in of the input space (figure).

2. Linear Discriminant Analysis

To use Bayes classifier, we have to estimate $p(\mathcal{C}_k|\boldsymbol{x})$. The LDA estimates $p(\boldsymbol{x}|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$ under two assumptions:

- Data within each class is normally distributed: $p(\boldsymbol{x}|\mathcal{C}_k) = \mathcal{N}(\mu_k, \Sigma)$.
- Each class has its own mean μ_k but all classes share a common covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$.

$$\mathcal{N}(\boldsymbol{\mu_k}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_k})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k})\right)$$

Formulation:

- Bayes classifier: $\hat{y}(\boldsymbol{x}) = \underset{\mathrm{k}}{\operatorname{argmax}}\, p(\mathcal{C}_k|\boldsymbol{x}) = \underset{\mathrm{k}}{\operatorname{argmax}}\, p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)$.
- Bayes theorem: $p(\mathcal{C}_k|\boldsymbol{x}) = \frac{\pi_k\mathcal{N}(\boldsymbol{\mu_k},\boldsymbol{\Sigma})}{p(\boldsymbol{x})}$ where $p(\boldsymbol{x}|\mathcal{C}_k) = \mathcal{N}(\boldsymbol{\mu_k}, \boldsymbol{\Sigma})$ and $p(\mathcal{C}_k) = \pi_k$
- We can then define the rule: $\hat{y}_{LDA}(x) = \underset{\mathrm{k}}{\operatorname{argmax}}\, \pi_k\mathcal{N}(\boldsymbol{\mu_k}, \boldsymbol{\Sigma})$
- Since $\log(-)$ is monotone, we can maximize $\delta_k(\boldsymbol{x}) = \log\pi_k\mathcal{N}(\boldsymbol{\mu_k}, \boldsymbol{\Sigma}) = \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu_k} - \frac{1}{2}\boldsymbol{\mu_k}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu_k} + \log\pi_k + constant$.

In practice, we don't know the Gaussian distribution parameters, but we can estimate them:

- $\hat{\pi}_k = \frac{N_k}{N}$: number of samples in each class divided by the total number of samples.
- $\boldsymbol{\mu_k} = \frac{1}{N_k}\sum \boldsymbol{x_i}$: the mean value of each class
- $\widehat{\boldsymbol{\Sigma}} = \frac{1}{N-K}\sum_k\sum_{y_i\in k}(\boldsymbol{x_i}-\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}})(\boldsymbol{x_i}-\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}})^T$: the covariance matrix (we normalize by $N-K$ to get unbiased estimator).

Finally, $\hat{\delta}_k(\boldsymbol{x}) = x^T\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}} - \frac{1}{2}\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}}^T\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}} + \log\hat{\pi}_k$: linear in $\boldsymbol{x}$, because it follows the form $\boldsymbol{a_k} + \boldsymbol{b_k^T}\boldsymbol{x}$.

The decision boundary between two classes k and j is given by $\{\boldsymbol{x}: \hat{\delta}_k(\boldsymbol{x}) = \hat{\delta}_j(\boldsymbol{x})\}$.

The Quadratic Discriminant Analysis removes the assumption of equal covariance among all classes, so the decision boundaries are determined by quadratic functions.

3. Logistic Regression

Let's introduce the log-odds of class k:

$$\log\frac{p(\mathcal{C}_k|\boldsymbol{x})}{p(\mathcal{C}_{\bar{k}}|\boldsymbol{x})} = \log\frac{\pi_k}{\pi_{\bar{k}}} - \frac{1}{2}(\boldsymbol{\mu_k}+\boldsymbol{\mu_{\bar{k}}})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu_k}-\boldsymbol{\mu_{\bar{k}}}) + \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu_k}-\boldsymbol{\mu_{\bar{k}}}) = w_0 + \boldsymbol{w}^T\boldsymbol{x}$$

So why not estimating w and w_0 directly rather than passing through $\hat{\mu}$, $\hat{\Sigma}$ and $\hat{\pi}$? → Logistic Regression.

A binary class problem: $p(\mathcal{C}_1|\boldsymbol{x}) = \frac{p(\mathcal{C}_1)p(\boldsymbol{x}|\mathcal{C}_1)}{p(\mathcal{C}_1)p(\boldsymbol{x}|\mathcal{C}_1)+p(\mathcal{C}_2)p(\boldsymbol{x}|\mathcal{C}_2)} = \frac{1}{1+\exp(-a)} = \sigma(a)$ where $a = \log\frac{p(\mathcal{C}_1)p(\boldsymbol{x}|\mathcal{C}_1)}{p(\mathcal{C}_2)p(\boldsymbol{x}|\mathcal{C}_2)}$

Finally, we have $p(\mathcal{C}_1|\boldsymbol{x}) = \sigma(w_0 + \boldsymbol{w}^T\boldsymbol{x})$. Now our goal is to estimate w_0 and w.

Given a training set $\{(\boldsymbol{x_i}, y_i), i = 1, \dots, N\}$ where $y_i \in \{0,1\}$, we will use maximum likelihood to fit our model:

$$L(\boldsymbol{w}) = \prod p(y_i|\boldsymbol{x_i};\boldsymbol{w}) = \prod \sigma(w_0+\boldsymbol{w}^T\boldsymbol{x})^{y_i}\big(1-\sigma(w_0+\boldsymbol{w}^T\boldsymbol{x})\big)^{1-y_i}$$

And the log-likelihood can be written:

$$\ell(\boldsymbol{w}) = \log L(\boldsymbol{w}) = \sum \log p(y_i|\boldsymbol{x_i};\boldsymbol{w}) = \sum y_i \log \sigma(w_0 + \boldsymbol{w}^T\boldsymbol{x}) + (1-y_i)\log\big(1-\sigma(w_0+\boldsymbol{w}^T\boldsymbol{x})\big)$$
$$= \sum y_i(w_0 + \boldsymbol{w}^T\boldsymbol{x}) - \log(1+\exp(w_0+\boldsymbol{w}^T\boldsymbol{x}))$$

Fitting and Predicting: the coefficients are estimated via Maximum Likelihood Estimation:

$$\widehat{\boldsymbol{w}_0}, \widehat{\boldsymbol{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum y_i(w_0 + \boldsymbol{w}^T\boldsymbol{x}) - \log(1+\exp(w_0+\boldsymbol{w}^T\boldsymbol{x}))$$

And given a new sample: $\hat{y}_{LogReg}(\boldsymbol{x}) = 1 \; if \; w_0 + \boldsymbol{w}^T\boldsymbol{x} > 0 \; else \; 0$.

A multiple-class problem: K-class

We chose a base class, let's say class K and we only need to specify the decision boundary between class j and the base class K.

$$\begin{cases} \log\frac{p(\mathcal{C}_1|\boldsymbol{x})}{p(\mathcal{C}_K|\boldsymbol{x})} = w_{10} + \boldsymbol{w_1^T}\boldsymbol{x} \\ \vdots \\ \log\frac{p(\mathcal{C}_{K-1}|\boldsymbol{x})}{p(\mathcal{C}_K|\boldsymbol{x})} = w_{(K-1)0} + \boldsymbol{w_{K-1}^T}\boldsymbol{x} \end{cases}$$, where $p(\mathcal{C}_k|\boldsymbol{x}) = \frac{\exp(w_{k0}+\boldsymbol{w_k^T}\boldsymbol{x})}{1+\sum_{l=1}^{K-1}\exp(w_{l0}+\boldsymbol{w_l^T}\boldsymbol{x})}$ and $p(\mathcal{C}_K|\boldsymbol{x}) = \frac{1}{1+\sum_{l=1}^{K-1}\exp(w_{l0}+\boldsymbol{w_l^T}\boldsymbol{x})}$.

Estimating $\widehat{\boldsymbol{w}}$ using Newton-Raphson algorithm

It's an iterative algorithm used to find a zero of a function f, i.e. to find θ such that $f(\theta) = 0$. So, we can apply it to $f = \frac{\partial \ell}{\partial \mathbf{w}}$ to find a point where the log-likelihood is maximal.

The update rule is $\theta \leftarrow \theta - \frac{f(\theta)}{f'(\theta)}$, so we will have $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{w}}\ell(\boldsymbol{w})$.

We will minimize the cross-entropy error function: $E(\boldsymbol{w}) = -\ell(\boldsymbol{w}) = -\sum y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$

First derivative: $\nabla_{\boldsymbol{w}}E(\boldsymbol{w}) = \frac{\partial E}{\partial \boldsymbol{w}} = \sum(\hat{y}_i - y_i)\boldsymbol{x_i}$ (using the fact that $\frac{d\sigma}{da} = \sigma(1-\sigma)$)

Second derivative: $\frac{\partial^2 E}{\partial \boldsymbol{w}\partial \boldsymbol{w}^T} = \sum \hat{y}_i(1-\hat{y}_i)\boldsymbol{x_i}\boldsymbol{x_i^T}$

Using matrix notations: $\partial E = \boldsymbol{X}^T(\hat{y}-y)$ and $\partial^2 E = \boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X}$ where $R_{ii} = \hat{y}_i(1-\hat{y}_i)$

We get $\boldsymbol{w}^{(new)} = \boldsymbol{w}^{(old)} - (\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X})^{-1}\boldsymbol{X}^T(\hat{y}-y) = (\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{R}\left(\boldsymbol{X}\boldsymbol{w}^{(old)} - \boldsymbol{R}^{-1}(\hat{y}-y)\right) = (\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{z}$

where $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}^{(old)} - \boldsymbol{R}^{-1}(\hat{y}-y)$ is an N-dimensional vector.

NB:

- At each iteration R and z change as they depend on w, we have to update them at each iteration.
- We know that the algorithm converges as the function f is concave.
- Overshooting can occur.
- Computationally expensive due to the Hessian matrix.

CHAPTER 3: Optimization (Week 3)

Unconstrained optimization problem: minimize $f(\theta)$, find θ^* such as $\forall\theta, f^* = f(\theta^*) \leq f(\theta)$.

We are assuming that f is differentiable.

- Analytically solvable. E.g. $\underset{\mathbf{w}}{\operatorname{argmin}}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) \rightarrow \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$.
- Using an iterative algorithm. E.g. $w^{(new)} = \operatorname{argmin}(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{w})^T\boldsymbol{R}(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{w})$
  - Can be descent: $f\left(\theta^{(k+1)}\right) < f\left(\theta^{(k)}\right)$
  - $f\left(\theta^{(k)}\right)$ converges but not necessarily to f^*
  - Stop when reaching maximum number or iterations or when $\| \nabla f\left(\theta^{(k)}\right) \| \leq \epsilon$
  - Non-heuristic: does not depend on $\theta^{(1)}$

Constrained optimization problem: minimize $f(\theta)$ subject to $g_i(\theta) = c_i$ and $h_j(\theta) \geq d_j$

Gradient-based methods
We assume that we have an unconstrained problem and f is continuous and convex.
Initialize $\theta^1 \in \mathbb{R}$, and while $\| \nabla f(\theta) \| > \epsilon$ do $\theta \leftarrow \theta - \alpha \nabla f(\theta)$.

CHAPTER 4: Neural Networks (Week 5)

The perceptron

Goal: find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.
Assumptions:

- Data is linearly separable
- Binary classification using labels $y \in \{-1,1\}$

Formulation: $y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{x})$ where $f(a) = \begin{cases} +1, a \geq 0 \\ -1, a < 0 \end{cases}$ and $x = \binom{x_i}{1}, w = \binom{w_i}{b}$

Patterns x_n in $\mathcal{C}_1$ will have $\boldsymbol{w}^T \boldsymbol{x_n} > 0$ and patterns x_n in $\mathcal{C}_2$ will have $\boldsymbol{w}^T \boldsymbol{x_n} < 0$. Using the fact that $y \in \{-1,1\}$, all patterns will satisfy $\boldsymbol{w}^T \boldsymbol{x_n} y_n > 0$.
The error function, also called perceptron criterion, is given by: $E_P(w) = -\sum_{n \in \mathcal{M}} \boldsymbol{w}^T \boldsymbol{x_n} y_n$ where $\mathcal{M}$ denotes the set of all misclassified patterns.
Finding the weights: we apply the stochastic gradient decent algorithm to this function:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \nabla E_P(\boldsymbol{w}) = \boldsymbol{w}^{(k)} + \alpha \boldsymbol{x_n} y_n$$

At every step k, we have to check every misclassified pattern $\boldsymbol{x_n}$ and update $\boldsymbol{w}^{(k)}$.
+ If data is linearly separable, then the perceptron guarantees a convergence to this solution.
− The number of steps to convergence might be large.
− The algorithm does not converge when data is not linearly separable (XOR function).

Neural Networks

Reminder
The linear models for regression and classification are based on linear combination of fixed nonlinear basis functions $\Phi_j(x)$ and take the form $y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum w_j \Phi_j(\boldsymbol{x})\right)$ where $f(.)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression.

Goal: making the basis functions $\Phi_j(x)$ depend on parameters and allow these parameters to be adjusted along with the coefficients $\{w_j\}$ during training.
Feed-forward Network Functions: this is a series of functional transformations aka Multilayer Perceptron

Step 1: construct M linear combinations of the input variables $x_1, \dots, x_D$ in the form

$$a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Step 2: transform each activation using a differentiable, nonlinear activation function to give

$$z_j = h(a_j)$$

Step 3: these values are again linearly combined to give output unit activations

$$a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

Step 4: the output unit activations are transformed using an activation function to give y_k

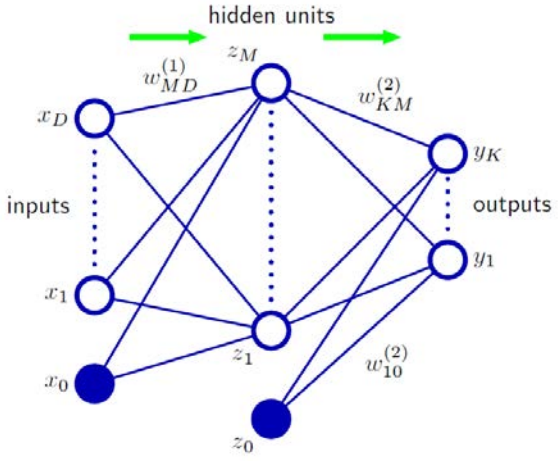
We can combine these various stages to give the overall network function that takes the form

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_j w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Thus, the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\boldsymbol{w}$ of adjustable parameters.

NB

- $i = 1, \dots, D$ index of dimension of the input $\boldsymbol{x}$
- $j = 1, \dots, M$ index of linear combinations (number of neurons in the first hidden layer)
- $k = 1, \dots, K$ index of linear combinations (number of neurons in the output layer)
- (1) and (2) indicate the layer index
- $w_{ji}^{(1)}$ are weights and $w_{j0}^{(1)}$ are biases
- Common activation functions: sigmoid $\sigma(a) = \frac{1}{1+e^{-a}}$, $\tanh a = \frac{e^a - e^{-a}}{e^a + e^{-a}}$, ReLu $h(a) = \max(0, a)$



The bias parameter can be absorbed into the set of weight parameters, so that the final function becomes

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=0} w_{kj}^{(2)} h\left(\sum_{i=0} w_{ji}^{(1)} x_i\right)\right) \Leftrightarrow y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_j w_j \Phi_j(\boldsymbol{x})\right)$$

Network Training: Backpropagation

We introduce the error function $E(w) = \sum_n E_n(w)$ and we will try to estimate $\nabla E_n(\boldsymbol{w})$.

Consider first a simple linear model $\hat{y}_k = \sum_i w_{ki} x_i$, together with an error function that, for a particular input pattern n, takes the form $E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$, where $\hat{y}_{nk} = \hat{y}_k(\boldsymbol{x_n}, \boldsymbol{w})$.

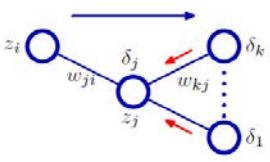
The gradient of this error function with respect to a weight w_{ji} is $\frac{\partial E_n}{\partial w_{ji}} = (w_{ji} x_{ni} - y_{nj}) x_{ni} = (\hat{y}_{nj} - y_{nj}) x_{ni}$ which can be interpreted as an error signal $\hat{y}_{nj} - y_{nj}$ associated to the output end of the link w_{ji} and the variable x_{ni}.

As we saw in feed-forward network, each unit computes a weighted sum of its inputs $a_j = \sum_i w_{ji} z_i$ which is then transformed by a nonlinear activation function $h(.)$ to give the activation $z_j = h(a_j) = h(\sum_i w_{ji} z_i)$.

Let's consider the derivative of the E_n with respect to a weight w_{ji} for every input pattern n

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

So, we have to calculate the value of δ_j for every hidden and output unit and then apply the previous formula.



→ For the output units $\delta_k = \frac{\partial E_n}{\partial a_k} = \hat{y}_{nk} - y_{nk} = \hat{y}_k - y_k$ (we omit n to simplify notations)

→ For the hidden units $\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j}\left(\sum_j w_{kj} h(a_j)\right) = h'(a_j)\sum_k w_{kj}\delta_k$

Error Backpropagation

1. Apply an input vector x_n to the network and forward propagate using $a_j = \sum_i w_{ji} z_i$ and $z_j = h(a_j)$
2. Evaluate the δ_k for all output units using $\delta_k = \hat{y}_k - y_k$
3. Backpropagate the δ's using $\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$ to obtain δ_j for each hidden unit in the network
4. Use $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$ to evaluate the required derivatives

Properties of MLP

- Universal Boolean functions: can compute any Boolean function (e.g. can implement XOR)
- Universal classification functions:
- Universal approximators: can approximate continuous functions on compact subsets of $\mathbb{R}^d$

Summary on MLP

- Advantages
  - Very general, can be applied in many situations
  - Powerful according to theory
  - Efficient according to practice
- Drawbacks
  - Training is often slow
  - Difficult choice of number of layers & neurons

CHAPTER 5: Relevant Machine Learning Topics – Part I (Week 7)

In this chapter, we will answer a few questions:

- Will our models perform well on testing data?
- In the available training data enough?
- Which model should be chosen?

Definitions

Generalization: ability of predictor to perform well on unseen data.

Model Selection: task of selecting a model from a set of candidate models given the data.

Bias-Variance Tradeoff

Training dataset $\mathcal{T} = \{(\boldsymbol{x_i}, y_i), i = 1, \dots, N\}$: an i.i.d. drawn from some distribution $P(X, Y)$

Expected label: $\bar{y}(x) = \int y p(y|x)\, \partial y = \mathbb{E}_{y|\boldsymbol{x}}[Y]$

Machine learning algorithm $\mathcal{A}$

Learning process: using $\mathcal{T}$ to learn a hypothesis (model/classifier). Formally: $h_{\mathcal{T}} = \mathcal{A}(\mathcal{T})$

Expected classifier: $\bar{h} = \mathbb{E}_{\mathcal{T} \sim P^N}[h_{\mathcal{T}}] = \int h_{\mathcal{T}}(\boldsymbol{x}) \Pr(\mathcal{T})\, \partial \mathcal{T}$

Expected test error: $\mathbb{E}_{(\boldsymbol{x},y)\sim P}[(h_{\mathcal{T}}(\boldsymbol{x}) - y)^2] = \int \int (h_{\mathcal{T}}(\boldsymbol{x}) - y)^2 \Pr(\boldsymbol{x}, y)\, \partial \boldsymbol{x} \partial y$

Expected test error of $\mathcal{A}$: evaluates the quality of ML algorithm $\mathcal{A}$ with respect to data distribution $P(X, Y)$

$$\mathbb{E}_{\substack{(x,y)\sim P \\ \mathcal{T}\sim P^N}}[(h_{\mathcal{T}}(\boldsymbol{x})-y)^2] = \int\int\int (h_{\mathcal{T}}(\boldsymbol{x})-y)^2 \Pr(\boldsymbol{x},y)\Pr(\mathcal{T})\,\partial\boldsymbol{x}\partial y\partial\mathcal{T}$$

$$\mathbb{E}_{x,y,\mathcal{T}}[(h_{\mathcal{T}}(\boldsymbol{x})-y)^2] = \mathbb{E}_{x,\mathcal{T}}\left[\left(h_{\mathcal{T}}(\boldsymbol{x})-\bar{h}(x)\right)^2\right] + \mathbb{E}_{x,y}[(\bar{y}(x)-y)^2] + \mathbb{E}_x\left[\bar{h}(x)-\bar{y}(x)\right]^2$$

- Variance: error caused from sensitivity to fluctuations in the training set. High variance can cause an algorithm to model random noise in the training set, rather than the intended outputs: overfitting.
- Squared Bias: error caused by the simplifying assumptions built into the model (e.g. the approximation of a nonlinear using a learning method for linear models): underfitting.
- Noise: this is the error associated to the data. No matter how good the model is, data will always have certain amount of noise that cannot be removed.

Regularization

Large sensitivity can lead to poor performance of the model: ideally $\boldsymbol{w}$ should not be too large.
We introduce a regularizer to control the size of $\boldsymbol{w}$: $R(\boldsymbol{w}) = \sum w_i^2$ or $R(\boldsymbol{w}) = \sum|w_i|$.
Now the predictor should also avoid being too sensitive using a regularization parameter $\lambda \geq 0$

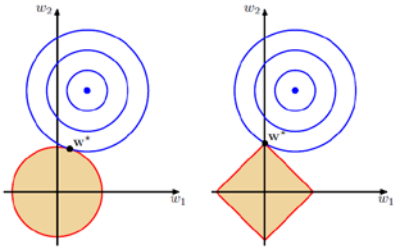
$$\mathcal{L}(w) + \lambda R(w) = \frac{1}{N}\sum \ell_w(y_i, x_i w) + \frac{\lambda}{N}\sum w_i^2$$

$$\underset{\mathrm{w}}{\operatorname{argmin}} \frac{1}{N}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) + \frac{\lambda}{N}\boldsymbol{w}^T\boldsymbol{w}$$

Which gives $\widehat{\boldsymbol{w}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

When to use regularization?

- If $D > N$, so it's impossible to invert $\boldsymbol{X}^T\boldsymbol{X}$ to get $\widehat{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
- To reduce variance: ridge regression shrinks to zero the size of the coefficients.
- To perform feature selection: Lasso can reduce variance and lead some weights to go to zero.



Validation

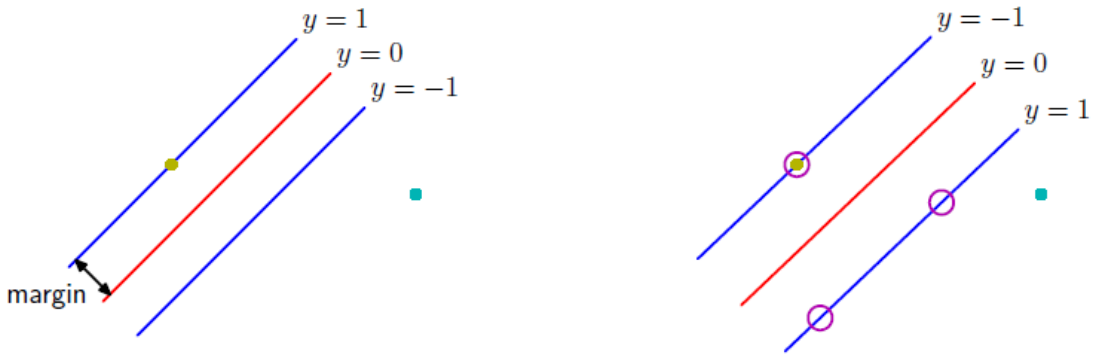
We split data into training data and testing data (e.g. 80/20 or 90/10) and apply the algorithm on the testing data. The testing error should be worse than the training one but acceptable.
Overfitting occurs when the model fits the data very well → Failure to generalize.
Underfitting occurs when the model cannot capture the structure of data → High training error.
Regularization to avoid overfitting, but how to choose parameters introduced by regularization? **Validation**.

K-fold Cross-validation:

1. The data is divided into K folds (a common value $K = 5$).
2. Fit a model to all data except the k^{th} fold.
3. Test on the k^{th} fold.
4. Average error across folds.
5. Use resulting average error of each model to judge.

CHAPTER 6: Support Vector Machines (Week 8)

SVM is a decision machine that makes prediction like the perceptron. But SVMs aim to find the boundary by maximizing the margin between classes.



Goal:

maximize the margin width (the distance between the decision boundary and the nearest points of each class). The location of this boundary is determined by a subset of data points, known as *support vectors*.

Some Linear Algebra

L: hyperplane defined by $f(x) = w_0 + \boldsymbol{w}^T\boldsymbol{x} = 0$

1. For any two points x_1 and $x_2 \in L, w^T(x_1 - x_2) = 0$ so $w^* = w/\|w\|$ is a vector normal to L
2. For any $x_0 \in L$, $w^T x_0 = -w_0$
3. The signed distance between any point x to L is the projection of $x - x_0$ into the normal vector $w^*.(x - x_0) = \frac{1}{\|w\|}.(w^T x - w^T x_0) = \frac{1}{\|w\|}.(w^T x + w_0) = \frac{y}{\|w\|}$

Formalization

- Training data: $\{\boldsymbol{x_i}, y_i\}$ for $i = 1, \dots, N$, $x_i \in \mathbb{R}^D$ and $y_i \in \{-1,1\}$
- y_i is the side of the perfect decision boundary the point x_i is on
- Our model: $\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$
- Correctly classified points: $y_i(\boldsymbol{w}^T\boldsymbol{x} + w_0) > 0$
- Distance of x_i of the perfect boundary L: $d(x_i, L) = \frac{y_i}{\|w\|}(\boldsymbol{w}^T\boldsymbol{x_i} + w_0)$

Aim?

$$\arg\max_{\mathrm{w},\mathrm{w}_0}\left\{\min_{\mathrm{i}} d(x_i, L)\right\} = \arg\max_{\mathrm{w},\mathrm{w}_0}\left\{\frac{1}{\|w\|}\min_{\mathrm{i}}[y_i(\boldsymbol{w}^T\boldsymbol{x_i} + w_0)]\right\}$$

If a set of points satisfies $\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0$ for a given w and w_0, then the same set of points will also satisfy $\alpha\boldsymbol{w}^T\boldsymbol{x} + \alpha w_0 = \hat{w}^T x + \hat{w}_0 = 0$.

→ The same boundary can be expressed in infinite ways.

We choose w and w_0 such that $y_{i^*}(\boldsymbol{w}^T\boldsymbol{x_{i^*}} + w_0) = 1$, so $d(\boldsymbol{x_{i^*}}, L) = \frac{1}{\|\boldsymbol{w}\|}$

All data points will satisfy $y_i(\boldsymbol{w}^T\boldsymbol{x_i} + w_0) \geq 1$

Aim? Maximize $\frac{1}{\|\boldsymbol{w}\|}$ or minimize $\|\boldsymbol{w}\|^2$

Putting all together,

$$\arg\min_{\mathrm{w},\mathrm{w}_0}\frac{1}{2}\|\boldsymbol{w}\|^2 \ s.t. y_i(\boldsymbol{w}^T\boldsymbol{x_i} + w_0) \geq 1$$

Lagrange multipliers

Maximizing $f\ s.t.\ g(x,y)=c$ means to find the level curve of f with greatest value intersecting the constraint curve. At this point, the two curves are tangent.

Since ∇f is perpendicular to its level curve and ∇g is perpendicular to the constraint curve, we want points where $\nabla f = \lambda \nabla g$.

Lagrange function $\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda g(x,y)$

Solve $\nabla \mathcal{L}(x,y,\lambda) = 0$ to find critical points which are candidates to be the max/min

Lagrange multipliers with multiple constraints

In general, if we want to minimize $f(x_1,\dots,x_n) = 0\ s.t.\ g_i(x_1,\dots,x_n) = 0$ for $i = 1,\dots,M$, we introduce $\mathcal{L}(x_1,\dots,x_n,\lambda_1,\dots,\lambda_M) = f(x_1,\dots,x_n) - \sum \lambda_k g_k(x_1,\dots,x_n)$, and solve

$$\nabla \mathcal{L}(x_1,\dots,x_n,\lambda_1,\dots,\lambda_M) = 0 \Leftrightarrow \begin{cases} \nabla f(x_1,\dots,x_n) - \sum \lambda_k \nabla g_k(x_1,\dots,x_n) = 0 \\ g_1(x_1,\dots,x_n) = \dots = g_M(x_1,\dots,x_n) = 0 \end{cases}$$

Using Lagrange multipliers to solve $\arg\min_{w,w_0} \frac{1}{2}\|\boldsymbol{w}\|^2\ \ s.t.\ y_i(\boldsymbol{w}^T\boldsymbol{x_i} + w_0) \geq 1$

Lagrange function $\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum \alpha_i\{y_i(\boldsymbol{w}^T\Phi(\boldsymbol{x_i}) + w_0) - 1\}$ where $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_n)^T$

$$\nabla_{w,w_0} f = \alpha \nabla_{\boldsymbol{w},\boldsymbol{w_0}} g \Leftrightarrow \begin{cases} \boldsymbol{w} = \sum \alpha_i y_i \Phi(\boldsymbol{x_i}) \\ 0 = \sum \alpha_i y_i \end{cases}$$

Dual representation: $\|\boldsymbol{w}\|^2 = \sum\_\alpha_i y_i \Phi(\boldsymbol{x_i}) \sum \alpha_j y_j \Phi(\boldsymbol{x_j}) = \sum\sum \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x_i})^{\mathrm{T}} \Phi(\boldsymbol{x_j})$

For the second term: $\sum \alpha_i\{y_i(\boldsymbol{w}^T\Phi(\boldsymbol{x_i}) + w_0) - 1\} = \sum \alpha_i y_i \boldsymbol{w}^T\Phi(\boldsymbol{x_i}) + \sum \alpha_i y_i w_0 - \sum \alpha_i$ and replace $\boldsymbol{w}$.

Finally, $\tilde{\mathcal{L}}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x_i})^{\mathrm{T}} \Phi(\boldsymbol{x_j})$

Constrained optimization should satisfy Karush–Kuhn–Tucker conditions:

1. $\alpha_i \geq 0$ ($\alpha_i > 0$ then x_i is a support vector, $\alpha_i = 0$ then x_i is outside the margin)
2. $y_i(\boldsymbol{w}^T\Phi(\boldsymbol{x_i}) + w_0) - 1 \geq 0$
3. $\alpha_i(y_i(\boldsymbol{w}^T\Phi(\boldsymbol{x_i}) + w_0) - 1) = 0$

Optimal boundary:

1. solve $\max_{\alpha} \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x_i})^{\mathrm{T}} \Phi(\boldsymbol{x_j})\ \ s.t.\ \alpha_i \geq 0\ and\ \sum \alpha_i y_i = 0$
2. Compute $\widehat{\boldsymbol{w}} = \sum \hat{\alpha}_i y_i \Phi(\boldsymbol{x_i})$
3. Compute w_0 using $\hat{\alpha}_{i_i}(y_i(\widehat{\boldsymbol{w}}^T\Phi(\boldsymbol{x_i}) + w_0) - 1) = 0$, if x_i is a support vector $y_i(\widehat{\boldsymbol{w}}^T\Phi(\boldsymbol{x_i}) + w_0) - 1 = 0 \Rightarrow w_0 = y_i - \widehat{\boldsymbol{w}}^T\Phi(\boldsymbol{x_i})$. It's better to average $w_0 = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} y_i - \widehat{\boldsymbol{w}}^T\Phi(\boldsymbol{x_i})$ where $\mathcal{S} = \{i, \alpha_i > 0\}$.
4. Using the expression obtained for $\boldsymbol{w}$, a new point can be classified using $\hat{y}(\boldsymbol{x_{test}}) = \widehat{\boldsymbol{w}}^T\boldsymbol{x_{test}} + w_0 = \sum \hat{\alpha}_i y_i \Phi(\boldsymbol{x_i})^{\mathrm{T}} \Phi(\boldsymbol{x_{test}}) + w_0$ whether it's positive ($\boldsymbol{x_{test}} \in \mathcal{C}_1$) or negative ($\boldsymbol{x_{test}} \in \mathcal{C}_2$).

CHAPTER 7: Kernels (Week 9)

SVM limitations: real data won't probably be linearly separable.

Possible solution: transform the data.

Sometimes, the decision boundary is not linear but it may become if the set is projected in higher dimension space.

Idea:

1. Define a transform ϕ from input space to feature space $x \to \phi(x)$
2. Solve a linear problem in the feature space
3. This gives a non-linear classifier in the input space

This will lead to calculate $\phi(x)$ for each sample which is not practical

Kernel: a function that can be expressed as a dot product in some feature space $K(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$

Example: $K(u,v) = (uv + c)^2 = \left(u^2, \sqrt{2c}u, c\right)^T \left(v^2, \sqrt{2c}v, c\right) = \phi(u)^T \phi(v)$ where $\phi: x \to \left(x^2, \sqrt{2c}x, c\right)$

Given K, is there a ϕ implied by the kernel?

If $\left(K(x_i, x_j)\right)_{1 \le i,j \le n}$ is a symmetric positive semi-defined matrix ($z^T K z \ge 0$ for every non-zero vector z) then K is a kernel (no need to compute ϕ)

Back to dual objective problem: we solve $\max_{\alpha} \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ and use $\sum \hat{\alpha}_i y_i K(x_i, x_{test}) + w_0$ for prediction.

Example of kernels:

- $d - th$ degree polynomial: $K(u,v) = (1 + u.v)^d$
- Radial basis: $K(u,v) = \exp(-\gamma \|u - v\|^2)$
- Neural Network: $K(u,v) = \tanh(\xi_1 (u.v) + \xi_2)$

SVM multi-class classification:

- 1 versus all: N classifiers
- 1 versus 1: $N(N-1)/2$ classifiers

CHAPTER 8: Relevant Machine Learning Topics – Part II (Week 9)

A **learning machine**: given an input x predicts the output $\hat{y} = \pm 1$ using weights α.

The power of the model. Tradeoff between:

- Powerful model: more complex but can overfit
- Less powerful: not going to overfit but restricted in what I can model

Some terms & definitions

Probability of misclassification (test error): $R(\alpha) = \mathbb{E}\left[\frac{1}{2}\left(y - f(x, \alpha)\right)\right]$

Fraction training set misclassified (training error): $R_{emp}(\alpha) = \frac{1}{N} \sum \frac{1}{2} [y_i - f(x_i, \alpha)]$

The Nasty Formula (aka VC dimension): to estimate the error of future data.

$$\Pr\left(R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{1}{N}\left[h\left(\log\frac{2N}{h}+1\right) - \log\frac{\eta}{4}\right]}\right) = 1-\eta$$

The VC dimension h is the maximum number of points that can be arranged so that f shatters them (classify points within the 2^N possibilities without training error).
We choose the model with minimum upper bound (high VC → more chance to overfit)

CHAPTER 9: Decision Trees (Week 10)

Motivation: $K-NN$ classifier is time consuming because for every testing point it should calculate the distance between this point and every labeled point. Can we improve it?
→ Decision Trees: learning the simplest decision tree is an NP-complete problem, so we will use a greedy approach: empty tree → split on next best feature → recurse → get a compact tree

Impurity functions

Greedy strategy: we keep splitting data to minimize an impurity function.

- Data $S = \{(x_i, y_i), y_i \in \{1, \dots, K\}\}$ where K is the number of classes
- $S_k \subseteq S$ where $S_k = \{(x_i, y_i), y_i = k\}$ so $S = S_1 \cup \dots \cup S_K$
- $p_k = \frac{|S_k|}{|S|}$

Gini impurity: $G(S) = \sum_k p_k(1-p_k)$. For $K=2$, $G(S) = 2p(1-p)$
Kullback-Lieber Divergence (or Relative Entropy) to compare two probability distributions:

$$KL(p||q) = \sum_k p_k \log\frac{p_k}{q}$$

For $q = q_k = \frac{1}{K}$, $KL\left(p||\frac{1}{K}\right) = \dots = \sum p_k \log p_k + \log K$. So we look for $\min_p -\sum p_k \log p_k$

Algo:
→ Compute the impurity function for every attribute
→ Split the set S using the minimum impurity function attribute (greedy approach)
→ Make a decision tree node containing that attribute
→ Recurse on subsets with remaining
When stopping? All data points in the subset have the same input
There are no more features to consider for the split

Regression Trees

CART: Classification And Regression Trees. We have another impurity function: $\mathcal{L}(S) = \frac{1}{|S|}\sum_{(x,y)\in S}(y-\bar{y}_S)^2$ where $\bar{y}_S$ is the average y in S.
We can have overfitting problems if the size of the tree (number of nodes) is very high.
How to fix? Find simple trees, fixed depth/early stopping, etc.

Parametric vs. Non-Parametric Algorithms:

- Parametric: has a constant number of parameters, independent of the number of samples
- Non-Parametric: Scales as a function of the training samples ($K-NN$)

What about decision trees?

- Parametric: if the tree depth is fixed or limited by a maximum
- Non-Parametric: if trained in full depth ($O(\log_2 n)$)

CHAPTER10: Ensembles (Week 10)

Goal: reducing variance without increasing bias. How? Averaging.

Weak law of large numbers: $\frac{1}{m}\sum_i x_i(x) \to \bar{x}$ as $m \to \infty$.

Apply that to classifiers:

1. m datasets available $D_1, \dots, D_m$ drawn from P^n
2. Train a classifier on each dataset and the average: $\hat{h} = \sum_i h_{D_i}(x) \to \bar{h}(x)$

The problem is that we can't have $m \to \infty$ (to have an infinite number of datasets)

Solutions:

Bagging (**B**ootstrap **agg**regat**ing**)

Take repeated bootstrap datasets from training set D: given a set D of size N, draw N sample with replacement.

Algo:

1. Create k bootstrap samples $D_1, \dots, D_k$
2. Train a classifier on each D_i
3. Classify new instance by majority vote or average

$+$ easy to implement, reduce variance by keeping bias unchanged

$-$ computationally more expensive, correlated training sets

Random Forest

Random Forests differ from Bagging in only one way: select a random subset of features at each split in the learning process.

Algo:

1. Create k bootstrap samples $D_1, \dots, D_k$
2. For each D_i, train a full decision tree with slight modifications:
    a. Before each split, randomly select $k < d$ features without replacement.
    b. Consider only these features for the split (increases variance of trees)
3. Classify new instance by majority vote or average

Tips

- Good choice for RF params: $k = \sqrt{d}$ and m as large as possible.
- Trees do not require preprocessing (different feature scales are ok)

PART II: Unsupervised Learning

No targets or Labels: Clustering, Density Estimation, Dimensionality Reduction.

Set of N observations $\boldsymbol{x_i}$ from a random d-vector X with joint density $P(X)$ (vs. $P(Y|X)$ in supervised).

Approaches:

- Clustering: determine whether $P(X)$ can be represented by simpler densities representing distinct types or classes of observation (K-mean, DBSCAN, Hierarchical Clustering).
- Dimensionality Reduction: determine whether variables can be considered as functions of a smaller set of latent variables (Principal component analysis, self-organizing maps, autoencoders, principal curves, multidimensional scaling).

CHAPTER 1: Clustering (week 12)

The task of grouping a set of objects in such a way that objects in the same group (a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

Key Elements: Similarity Criteria, Number of Groups, Features

K-means

Formalization:

- Dataset $\mathcal{D} = \{x_1, \dots, x_N\}$ where $x_n \in \mathbb{R}^D$
- Goal 1: partition the dataset into K clusters
- D-dimensional vector μ_k: a prototype associated to the k-th cluster.
- Goal 2: assign points to clusters and minimize the sum of square distances of each point to its closest mean vector by introducing $r_{ik} = 1$ if $x_i \in cluster\ k$.

$$J = \sum \sum r_{ik} \left|\left|x_i - \mu_k\right|\right|^2 = \sum J_k$$

$$\frac{\partial J}{\partial \mu_k} = 2 \sum r_{ik}(x_i - \mu_k) = 0 \Rightarrow \mu_k = \frac{\sum r_{ik} x_i}{\sum r_{ik}}$$

Algorithm

Initialize μ_k

Repeat

For each x_i:

$$r_{ik} = \begin{cases} 1, if\ k = \underset{j}{\operatorname{argmin}} \left|\left|x_i - \mu_k\right|\right| \\ 0, otherwise \end{cases}$$

For each μ_k:

$$\mu_k = \frac{\sum r_{ik} x_i}{\sum r_{ik}}$$

Until μ_k converges

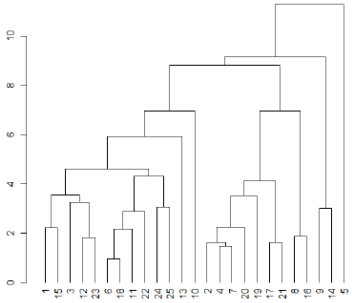
Hierarchical Clustering

K-means requires to know K

Results from K-means depend on the initialization

Bottom-up Hierarchical Clustering

1. Start with each point in its own
2. Identify the two closest clusters. Merge.
3. Repeat until all points are in a single cluster



Linkage: dissimilarity between two points/clusters (min, max, average)
At the beginning, each point represents a class, so the dissimilarity between classes is the distance between two points. After that, we need to think about dissimilarities between sets.
Linkages Types: complete (max), single (min), average, centroid.
→ The challenge: choice of similarity/dissimilarity measure (e.g. d, d^2) and the linkage.

CHAPTER 2: Dimensionality Reduction (week 12)

Principal Component Analysis (PCA) can be defined as the orthogonal projection of the data onto a lower Dimensional linear space.

Minimum-error formulation

Considering a complete orthonormal basis of vectors, i.e. $\{\boldsymbol{u_i}\}$ where $i = 1, \dots, D$ and $\boldsymbol{u_i^T}.\boldsymbol{u_j} = \delta_{ij}$
We can write $\boldsymbol{x_n} = \sum \alpha_{ni} u_i = \sum(\boldsymbol{x_n^T}.\boldsymbol{u_i})\boldsymbol{u_i}$
Goal: approximate each data point using a representation involving a restricted number $M < D$ of variables.
So $\tilde{\boldsymbol{x}}_{\boldsymbol{n}} = \sum_{i=1}^{M} z_{ni}\boldsymbol{u_i} + \sum_{i=M+1}^{D} b_i \boldsymbol{u_i}$, where $\{z_{ni}\}$ depend on the point and $\{b_i\}$ are the same for all data points.
$z_{ni} = \boldsymbol{x_n^T}.\boldsymbol{u_i}$ and $b_i = \bar{\boldsymbol{x}}^{\boldsymbol{T}}.\boldsymbol{u_i}$ are the optimum values.
Loss: $\mathcal{L}(\alpha) = \frac{1}{N}\sum\|\boldsymbol{x_n} - \tilde{\boldsymbol{x}}_{\boldsymbol{n}}\|^2 = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left(\boldsymbol{x_n^T}.\boldsymbol{u_i} - \bar{x}^T.u_i\right)^2 = \sum_{i=M+1}^{D}\boldsymbol{u_i^T}.\boldsymbol{S}.\boldsymbol{u_i}$ where $S = \frac{1}{N}\sum(x_n - \bar{x})(x_n - \bar{x})^T$: covariance matrix.

Algo:

1. Standardize data (mean to 0 and variance to 1)
2. Get the covariance matrix S
3. Compute eigenvectors and eigenvalues of S
4. Decreasingly sort eigenvalues
5. Compute new features $X.\theta$
6. Drop useless values