EN.601.475 Machine Learning

Contents

1 Introduction to Machine Learning

1.1 Recitation: Probability and Statistics

1.1.1 Basic Probability

The basic terminology are:

1. ***Sample Space*** $(\Omega,\ S)$: Set of possible outcomes.
2. ***Event Space*** $(\mathcal{F},\ E)$: Collection of subsets of outcomes, $\mathcal{F} = 2^{\Omega}$.
3. ***Probability Measure*** $(P,\ \pi)$: the probability of an event.
4. ***Probability Space***: A triple $(\Omega, \mathcal{F}, P)$.

Rearranging the chain rule of probability $P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$ yields Bayes' Rule.

Bayes' Rule

Suppose $A, B \in \mathcal{F}$,

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

where B_i are a partition of Ω.

Events are independent if the occurrence of one does not affect the probability of occurrence of the other.

1. Two events A and B are called ***independent*** if $P(A \cap B) = P(A)P(B)$, or equivalently, $P(B|A) = P(B)$ for $P(A) > 0$.
2. Two events A and B are called ***conditionally independent given C*** when $P(A \cap B|C) = P(A|C)P(B|C)$, or equivalently, $P(B|A, C) = P(B|C)$.

1.1.2 Distributions

A ***random variable*** is a function $X : \Omega \to \mathbb{R}^d$. The probability measure associated with the random variable is characterized by its ***cumulative distribution function (CDF)***: $F_X(x) = P(X \leq x)$.

- If X is discrete, it can be characterized by ***probability mass function (PMF)***:

$$f_X(x) = P(X = x)$$

- If X is continuous, it can be characterized by its ***probability density function (PDF)***:

$$f_X(x) = \frac{d}{dx} F_X(x)$$

  The probability of an interval (a, b) is given by $P(a < X < b) = \int_a^b f_X(x)\, dx$.

A ***joint distribution***'s PMF or PDF can be written as $f_{X,Y}(x, y)$. Two random variables are ***independent*** when joint PDF factorizes: $f_{X,Y}(x, y) = f_X(x) f_Y(y)$.

The marginalizing is constructed by $f_X(x) = \int_y f_{X,Y}(x, y)\, dy$ and the conditioning is given by $f_{X|Y}(x, y) = f_{X,Y}(x, y)/f_Y(y)$.

Expectation The ***expectation*** of a distribution describe the "mean" of a distribution, and it is given by

$$\mathbb{E}(x) := \begin{cases} \sum_x x f_X(x), & X \text{ is discrete} \\ \int_{-\infty}^{\infty} x f_X(x)\, dx, & X \text{ is continuous} \end{cases} \tag{1.1}$$

The expectation is linear, namely $\mathbb{E}(aX + bY + c) = a\mathbb{E}(X) + b\mathbb{E}(Y) + c$, and note that $\mathbb{E}(\mathbb{E}X) = \mathbb{E}X$.

Variance The ***variance*** of the distribution of a distribution describe how "spread out" a distribution is, and it is given by

$$\mathrm{Var}(X) := \mathbb{E}(X - \mathbb{E}X) = \mathbb{E}(X^2) - (\mathbb{E}X)^2 \tag{1.2}$$

The variance is not linear, but the following holds: $\mathrm{Var}(aX + c) = a^2\mathrm{Var}(X)$.

$$\begin{aligned} \mathrm{Var}(X+Y) &= \mathbb{E}(X - \mathbb{E}X)^2 + \mathbb{E}(Y - \mathbb{E}Y)^2 + 2\mathbb{E}[(X - \mathbb{E}X)(Y - \mathbb{E}Y)] \\ &= \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X, Y) \end{aligned}$$

where $\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}X)(Y - \mathbb{E}Y)]$. $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$ when X, Y are independent.

Entropy ***Entropy*** is a measure of uniformity in a distribution given by

$$H(X) = -\sum_x f_X(x) \log_2 f_X(x) = -\mathbb{E}(\log_2 f_X(x)) \tag{1.3}$$

1.1.3 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

Suppose $X_1, \cdots, X_n$ are i.i.d. (independent and identically distributed), let $\bar{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$, $\mathbb{E}X_i = \mu$, and $\mathrm{Var}(X_i) = \sigma^2$. Then we have $E\bar{X}_n = \mu$ and $\mathrm{Var}(\bar{X}_n) = \sigma^2/n$.

Chebyshev's Inequality states that

$$P(|\bar{X}_n - \mu| \geq \varepsilon) = \frac{\sigma^2}{n\varepsilon^2} \to 0$$

for any fixed ε, as $n \to \infty$.

Laws of Large Numbers

Suppose $X_1, \cdots, X_n$ are independent and identically distributed.

Weak law of large numbers: for all $\varepsilon > 0$, there is a n such that $|\bar{X}_n - \mu| < \varepsilon$, namely,

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| < \varepsilon) = 1$$

Strong law of large numbers: The mean converges to μ as n increases, namely,

$$P\left(\lim_{n\to\infty} \bar{X}_n = \mu\right) = 1$$

Central Limit Theorem

The distribution of $\bar{X}_n$ converges weakly to a Gaussian,

$$\lim_{n\to\infty} F_{\bar{X}_n}(x) = \phi\left(\frac{x-\mu}{\sqrt{n}\sigma}\right)$$

1.2 Introduction to Machine Learning

Summary

- Loss (square loss, 0/1 loss)
- Risk (Bayes risk, empirical risk), Bayes Optimal Rule

1.2.1 Supervised Learning Task

Machine Learning is the design and analysis of algorithms that improve their *performance* at some *task* with *experience.*

Task Given $X \in \mathcal{X}$ in the input domain, predict $Y \in \mathcal{Y}$ in the output domain, namely construct the ***prediction rule*** $f : \mathcal{X} \to \mathcal{Y}$ in the hypothesis class.

Performance Performance describes how well does the algorithm do on average

1. for a test data X drawn at random, and
2. for a set of training data and labels $\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$ drawn at random.

Performance is represented by the loss function.

Performance: Loss $\text{loss}(Y, f(X))$ is a measure of closeness between true label Y and prediction $f(X)$ (aka $\widehat{y}$); that is, loss function is a measure of the performance of f associated with the event. In classification, the loss function is usually ***0/1 loss***, given by

$$\text{loss}(Y, f(X)) = I(f(X) \neq Y) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases} \tag{1.4}$$

where I is called the indicator function. In regression, we use ***square loss***, which is given by

$$\text{loss}(Y, f(X)) = (f(X) - Y)^2 \tag{1.5}$$

Performance: Risk The risk function describes the performance of f on population $(X, Y) \sim \mathcal{P}_{XY}$ $(\mathcal{D}_n)$,

$$\text{Risk } R(f) \equiv \mathbb{E}_{XY}[\text{loss}(Y, f(X))] \tag{1.6}$$

The risk function of 0/1 loss is

$$\text{Risk}(f) = \mathbb{E}_{XY}\left[I(f(X) \neq Y)\right] = P(f(X) \neq Y)$$

and the risk function of square loss is

$$\text{Risk}(f) = \mathbb{E}\left[(f(x) - Y)^2\right]$$

which is also known as MSE (mean squared error).

Machine Learning Algorithm The learning algorithm takes the training data $\{(X_i, Y_i)\}_{i=1}^n$ and output a prediction function $\widehat{f}_n$ to optimize the performance.

1.2.2 Bayes Optimal Rule

Bayes Optimal Rule Ideal goal: construct $f^*: \mathcal{X} \to \mathcal{Y}$ such that the ***Bayes Risk*** $R^* = \min_{f \in \mathcal{F}} R(f)$ is obtained (i.e., $R^* \leq R(f)$ for all f), namely f such that the best possible performance is achieved. That is, construct f^* such that

$$f^* = \arg\min_f \mathbb{E}_{XY}[\text{loss}(Y, f(X))].$$

However, note that optimal rule is not computable, because P_{XY} is unknown.

Empirical Risk Minimizer Practical goal: given $\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$, learn prediction rule $\widehat{f}_n : \mathcal{X} \to \mathcal{Y}$. The ***Expected Risk*** (***Generalization Error***) of f is given by

$$\widehat{R}(f) = \mathbb{E}_{D_n}\left[R(f_n)\right] \equiv \mathbb{E}_{D_n}\left[\mathbb{E}_{XY}\left[\text{loss}(Y, f_n(X))\right]\right] = \frac{1}{n}\sum_{i=1}^{n}\left[\text{loss}(Y_i, f(X_i))\right] \tag{1.7}$$

When n is large, the law of large numbers suggests that

$$\lim_{n\to\infty} \widehat{R}(f) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}\left[\text{loss}(Y_i, f(X_i))\right] \longrightarrow \mathbb{E}_{XY}\left[\text{loss}(Y, f(X))\right] = R(f)$$

Therefore, we often use the ***Empirical Risk Minimizer***, the prediction that introduced the smallest empirical risk with the given experience,

$$\widehat{f}_n = \underset{f\in\mathcal{F}}{\text{argmin}}\, \widehat{R}(f) = \underset{f\in\mathcal{F}}{\text{argmin}}\, \frac{1}{n}\sum_{i=1}^{n}\left[\text{loss}(Y_i, f(X_i))\right] \tag{1.8}$$

1.2.3 Learning Algorithm Evaluation

Consistency The excess risk is given by

$$\text{Excess Risk} = \mathbb{E}_{D_n}\left[R(\widehat{f}_n)\right] - R(f^*)$$

Consistent algorithms have excess risk $\to 0$ as $n \to 0$.

Performance Evaluation One approach to evaluate the performance of an algorithm is to split available data into two sets $\{(X_i, Y_i)\}_{i=1}^n$, $\{(X_i', Y_i')\}_{i=1}^n$. Train $\widehat{f}_n$ using the learning algorithm with the training data. Validate result using the validation data. the The ***Validation Error***, an estimate of generalization error, is given by

$$\frac{1}{n}\sum_{i=1}^{n}\left[\text{loss}(Y_i', \widehat{f}_n(X_i'))\right]$$

Better estimates: k-fold cross-validation, hold one out validation.

1.2.4 Approach to Solve Machine Learning Problems

1. Consider the goal $\to$ definition of task **T**
2. Consider the nature of available (or potential) experience **E**
3. Choose type of output **O** to learn
4. Choose the Performance measure **P** (error/loss function)

5. Choose a representation for the input **X**
6. Choose a set of possible solutions **H** (hypothesis space)
7. Choose or design a learning algorithm

2 Regression

2.1 Linear Regression

Summary

- Linear regression: general form, loss, and risk
- Least square estimation: ERM, matrix closed form

2.1.1 Regression Introduction

Formal Setup

- Input data space $\mathcal{X}$, output (label, target) space $\mathcal{Y}$
- Unknown probability distribution $P(\cdot, \cdot)$ over $\mathcal{X} \times \mathcal{Y}$
- Given a set of labeled examples $(\mathbf{x_i}, y_i)$, $(i = 1, \cdots, N)$, sampled i.i.d. from p; $\mathbf{x_i} \in \mathcal{X}, y_i \in \mathcal{Y}$
- Goal: for any future $\mathbf{x}$, accurately predict y (drawn according to p); that is, learn a mapping $f : \mathcal{X} \to \mathcal{Y}$

Types of Supervised Problems Goal: learn $f : \mathcal{X} \to \mathcal{Y}$. We will consider two types of f based on the nature of $\mathcal{Y}$:

1. ***Regression***: $\mathcal{Y} = \mathbb{R}$, learn a (continuous) function f
2. ***Classification***: $\mathcal{Y} = \{1, \cdots, C\}$, learn a separator between classes

2.1.2 Linear Regression

Linear Fitting Linear Regression fits a linear function to an observed set of points $X = [x_1, \cdots, x_N]$, with associated labels $Y = [y_1, \cdots, y_N]$.

Least squares (LSQ) fitting criterion: find the function that minimizes sum (or average) of square distances between actual y's in the training set and predicted ones.

Linear Functions General form:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d = \mathbf{w} \cdot \mathbf{x} \tag{2.1}$$

where $\mathbf{x}$ is augmented so that $x_0 \equiv 1$. The linear functions are hyperplane in general, d-D case.

Loss: Empirical and Expected Let $\mathbf{x_i}$ denotes the i-th data point in $\mathcal{X}$ (column vector representing the i-th row), or $\mathbf{x_i} = [x_{i1}\ x_{i2}\ \cdots\ x_{in}]^T$, and $\mathbf{X}$ denotes the $N \times (d+1)$ matrix where i-th row is $\mathbf{x_i}^T$.

A ***loss function*** $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ maps prediction to cost, given true value: $l(\widehat{y}, y)$ defines the penalty paid for predicting $\widehat{y}$ when the true value is y. Standard choice for regression: square loss

$$l(\widehat{y}, y) = (\widehat{y} - y)^2,$$

it penalize larger mistakes more harshly.

The ***empirical loss*** of function $y(\mathbf{x}; \mathbf{w})$ on a set $\mathbf{X}$ is

$$L(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{n} l\left(f(\mathbf{x_i}; \mathbf{w}), y_i\right) = \frac{1}{N} \sum_{i=1}^{n} (y_i - f(\mathbf{x_i}; \mathbf{w}))^2,$$

and the ultimate goal is to minimize the expected loss, aka risk, ***risk***:

$$R(\mathbf{w}) = \mathbb{E}_{(\mathbf{x_0}, y_0) \sim p(\mathbf{x}, y)} \left[l(f(\mathbf{x_0}; \mathbf{w}), y_0) \right].$$

Empirical risk minimization (ERM) approach: the empirical loss serves as a proxy for the risk when the training set is representative of $p(\mathbf{x}, y)$.

2.1.3 Empirical Risk Minimization (ERM)

Two steps:

1. Select a restricted class $\mathcal{H}$ of hypotheses $f : \mathcal{X} \to \mathcal{Y}$. Note that linear functions are parameterized by $\mathbf{w}$.
2. Select a hypothesis $f^* \in \mathcal{H}$ based on training set (X, Y).

Least Squares: Estimation We need to minimize L w.r.t $\mathbf{w}$,

$$L(\mathbf{w}) = L(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{n} (y_i - \mathbf{w} \cdot \mathbf{x_i})^2 .$$

A necessary condition to minimize L is $\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, \mathbf{X}, \mathbf{y}) = 0$, i.e., partial derivative w.r.t. w_j,

$$\frac{\partial}{\partial w_j} L(\mathbf{w}) = -\frac{2}{N} \sum_{i=1} (y_i - \mathbf{w} \cdot \mathbf{x_i}) \, x_{ij} \tag{2.2}$$

must be 0 for all features j. Therefore, when $\mathbf{w}$ minimizes L,

1. errors have zero mean (by taking $j = 0$) and
2. errors are uncorrelated with the data.

Least Squares: Matrix Form We first need to augment $\mathbf{X}$ so that $\mathbf{X}_{\cdot,0} = 1$. The empirical loss in matrix form is

$$\begin{aligned} L(\mathbf{w}, \mathbf{X}, \mathbf{y}) &= \frac{1}{N} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{1}{N} (\mathbf{y}^T - \mathbf{w}^T \mathbf{X}^T)(\mathbf{y} - \mathbf{X}\mathbf{w}) \end{aligned}$$

Note that

$$\frac{\partial \mathbf{a}^T \mathbf{b}}{\partial \mathbf{a}} = \frac{\partial \mathbf{b}^T \mathbf{a}}{\partial \mathbf{a}} = \mathbf{b}, \quad \frac{\partial \mathbf{a}^T \mathbf{B} \mathbf{a}}{\partial \mathbf{a}} = 2\mathbf{B}\mathbf{a}.$$

Therefore,

$$\begin{aligned} \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{N} \frac{\partial}{\partial \mathbf{w}} \left[\mathbf{y}^T \mathbf{y} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} \right] \\ &= \frac{1}{N} \left[0 - \mathbf{X}^T \mathbf{y} - (\mathbf{y}^T \mathbf{X})^T + 2\mathbf{X}^T \mathbf{X} \mathbf{w} \right] \\ &= -\frac{2}{N} (\mathbf{X}^T \mathbf{y} - \mathbf{X}^T \mathbf{X} \mathbf{w}) \end{aligned}$$

Let $\partial L(\mathbf{w}) / \partial \mathbf{w} = 0$, we obtain the Normal equation $\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \mathbf{w}$. Assuming the inverse of $\mathbf{X}^T \mathbf{X}$ exists, the optimal parameters $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.3}$$

and $\mathbf{X}^\dagger \triangleq (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is called the Moore-Penrose pseudoinverse of $\mathbf{X}$. The prediction is given by $\widehat{y}_0 = \mathbf{w}^{*T}\mathbf{x}_0 = \mathbf{y}^T\mathbf{X}^{\dagger T}\mathbf{x}_0$.

Linear regression in Python:

```
X[:, 0] = 1; X[:, 1::] = x
w = np.dot(np.linalg.pinv(X), y) # regression param. approx.
y_hat = np.dot(X, w)             # prediction
```

2.2 Error Analysis in Linear Regression

Summary

- Approximation and estimation error
- Gaussian noise model, Square loss
- Generalized linear regression

2.2.1 Error Analysis

Best Unrestricted Predictor

Suppose the model class $\mathcal{H}$ is not restricted. Suppose $f : \mathcal{X} \to \mathbb{R}$. The risk for a single test point $\mathbf{x}_0$ is

$$\begin{aligned} R(f) &= \mathbb{E}_{(\mathbf{x}_0,y_0)\sim p(\mathbf{x},y)}\left[(f(\mathbf{x}_0)-y_0)^2\right] \\ &= \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x})}\left[\mathbb{E}_{y_0\sim p(y|\mathbf{x})}\left[(f(\mathbf{x}_0)-y_0)^2\middle|\,\mathbf{x}_0\right]\right] \\ &= \int_{\mathbf{x}_0}\left\{\mathbb{E}_{y_0\sim p(y|\mathbf{x})}\left[(f(\mathbf{x}_0)-y_0)^2\middle|\,\mathbf{x}_0\right]\right\}p(\mathbf{x}_0)d\mathbf{x}_0 \end{aligned}$$

and so the inner conditional expectation for each $\mathbf{x}_0$ need to be minimized,

$$\begin{aligned} \frac{\partial}{\partial f(\mathbf{x})}\mathbb{E}_{y_0\sim p(y|\mathbf{x})}\left[(f(\mathbf{x}_0)-y_0)^2\middle|\,\mathbf{x}_0\right] &= 2\,\mathbb{E}_{y_0\sim p(y|\mathbf{x})}\left[f(\mathbf{x}_0)-y_0 \mid \mathbf{x}_0\right] \\ &= 2\left(f(\mathbf{x}_0)-\mathbb{E}_{p(y|\mathbf{x})}\left[y_0|\mathbf{x}_0\right]\right) \end{aligned}$$

We can minimize the expected loss by setting f to the conditional expectation of y for each $\mathbf{x}$:

$$\widehat{f}(\mathbf{x}_0) = \mathbb{E}_{y_0\sim p(y|\mathbf{x})}\left[y_0|\mathbf{x}_0\right]$$

If we know the distribution $p(y|\mathbf{x})$, we can find the best unrestricted predictor, by taking for each $\mathbf{x}_0$ the expectation $\widehat{y}(\mathbf{x}_0) = f(\mathbf{x}_0) = \mathbb{E}_{y\sim p(y|\mathbf{x})}[y|\mathbf{x}_0]$. Based on the knowledge we have, there are some common approaches:

- Generative approach: estimate $p(\mathbf{x}, y)$, and normalize to find the conditional density $p(y|\mathbf{x})$.
- Discriminative approach: estimate the conditional density $p(y|\mathbf{x})$ dirctly form the data
- Non-probabilistic approach: fit $f(\mathbf{x})$ directly to the data.

Decomposition of Error in Linear Regression

Let $\widehat{\mathbf{w}} \in \mathcal{H}$ denotes the LSQ estimates from training data, and $\mathbf{w}^* \in \mathcal{H}$ denotes the optimal linear regression parameters (generally unknown). Notice that $y - \widehat{\mathbf{w}}\cdot\mathbf{x} = (y - \mathbf{w}^*\cdot\mathbf{x}) + (\mathbf{w}^*\cdot\mathbf{x} - \widehat{\mathbf{w}}\cdot\mathbf{x})$,

$$\begin{aligned} R(\widehat{\mathbf{w}}) = \mathbb{E}_{p(\mathbf{x},y)}\left[(y-\widehat{\mathbf{w}}\cdot\mathbf{x})^2\right] &= \mathbb{E}_{p(\mathbf{x},y)}\left[(y-\mathbf{w}^*\cdot\mathbf{x})^2\right] + \mathbb{E}_{p(\mathbf{x},y)}\left[(\mathbf{w}^*\cdot\mathbf{x}-\widehat{\mathbf{w}}\cdot\mathbf{x})^2\right] \\ &\quad + 2\,\mathbb{E}_{p(\mathbf{x},y)}\left[(y-\mathbf{w}^*\cdot\mathbf{x})(\mathbf{w}^*\cdot\mathbf{x}-\widehat{\mathbf{w}}\cdot\mathbf{x})\right] \end{aligned}$$

Note that the last term in the first equality vanishes since the expectation of prediction error (residual)is uncorrelated with any linear function of $\mathbf{x}$ (Section 2.1.3 Equation 2.1.2, Necessary condition 2 to minimize L), in particular, with $(\mathbf{w}^*\cdot\mathbf{x} - \widehat{\mathbf{w}}\cdot\mathbf{x})$. Therefore, we obtain

$$R(\widehat{\mathbf{w}}) = E_{p(\mathbf{x},y)}\left[(y - \mathbf{w}^* \cdot \mathbf{x})^2\right] + \mathbb{E}_{p(\mathbf{x},y)}\left[(\mathbf{w}^* \cdot \mathbf{x} - \widehat{\mathbf{w}} \cdot \mathbf{x})^2\right] \tag{2.4}$$

Approximation error (= bias^2) $E_{p(\mathbf{x},y)}\left[(y - \mathbf{w}^* \cdot \mathbf{x})^2\right]$ measures inherent limitation of the chosen hypothesis class (linear function). This error will remain even with infinite training data.

Estimation error (= variance) $\mathbb{E}_{p(\mathbf{x},y)}\left[(\mathbf{w}^* \cdot \mathbf{x} - \widehat{\mathbf{w}} \cdot \mathbf{x})^2\right]$ measures how close to the optimal $\mathbf{w}^*$ is $\widehat{\mathbf{w}}$ estimated from (finite) training data. For a consistent estimation procedure, $\lim_{N\to\infty} \widehat{\mathbf{w}} = \mathbf{w}^*$, so the estimation error decreases to 0.

2.2.2 Gaussian Noise Model (Square loss)

Noise Explicitly model the randomness in the data gives $y = f(\mathbf{x};\mathbf{w}) + \nu$ where the noise ν accounts for everything not captured by f. The noise may include a meaning component. Under this model, the best predictor is

$$f^*(x) = \mathbb{E}_{p(y\,|\,\mathbf{x})}\left[f(\mathbf{x};\mathbf{w}) + \nu\,|\,\mathbf{x}\right] = f(\mathbf{x};\mathbf{w}) + \mathbb{E}_{p(v)}[\nu]$$

Typically, $\mathbb{E}_{p(v)}[\nu] = 0$ (white noise / Gaussian noise).

Gaussian Noise Model Suppose $y = f(\mathbf{x};\mathbf{w}) + \nu$ where $v \sim \mathcal{N}(\nu; 0, \sigma^2)$. Given the input $\mathbf{x}$, the label y is a random variable

$$p(y\,|\,\mathbf{x};\mathbf{w},\sigma) = \mathcal{N}(y; f(\mathbf{x},\mathbf{w}), \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y - f(\mathbf{x};\mathbf{w}))^2}{2\sigma^2}\right) \tag{2.5}$$

The ***likelihood*** of the parameters $\mathbf{w}$ given the observed data $X = [x_1, \cdots, x_N]$, $Y = [y_1, \cdots, y_N]^T$ is defined as $p(Y\,|\,X;\mathbf{w},\sigma)$, namely the probability of observing these ys for the given $\mathbf{x}$s, under model parameterized by $\mathbf{w}$ and σ. Under the assumption that data are i.i.d., the likelihood is given by

$$p(Y\,|\,X;\mathbf{w},\sigma) = \prod_{i=1}^{N} p(y_i\,|\,\mathbf{x}_i,\mathbf{w},\sigma)$$

Maximizing the likelihood is equivalent to maximizing log-likelihood,

$$\log p(Y\,|\,X;\mathbf{w},\sigma) = \sum_{i=1}^{N} \log p(y\,|\,\mathbf{x};\mathbf{w},\sigma) = \sum_{i=1}^{N}\left[-\frac{(y_i - f(\mathbf{x}_i;\mathbf{w}))^2}{2\sigma^2} - \log\sqrt{2\pi}\sigma\right]$$

$$\log p(Y\,|\,X;\mathbf{w},\sigma) = -N\log\sqrt{2\pi}\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\mathbf{w}))^2 \tag{2.6}$$

Note that $-N\log\sqrt{2\pi}\sigma$ and $-1/2\sigma^2$ are independent of $\mathbf{w}$. Thus, we can define a new function, ***log-loss*** - negative conditional log-probability of the training data,

$$L(f(\mathbf{x};\mathbf{w}), y) = -\log p(y\,|\,\mathbf{x};\mathbf{w},\sigma) \qquad \text{and} \qquad L(f(\mathbf{X};\mathbf{w}), \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N}\log p(y_i\,|\,\mathbf{x}_i,\mathbf{w})$$

and so maximizing log-likelihood is equivalent to minimizing log-loss, which is equivalent to minimizing squared loss. Under Gaussian noise model,

$$w^* = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} L(f(\mathbf{x};\mathbf{w}), y) = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i;\mathbf{w}))^2$$

$\star$ In conclusion, maximizing the likelihood under Gaussian Noise Model is equivalent to *minimizing square loss.*

2.2.3 Generalized Linear Regression

Suppose the input space is $\mathcal{X} \subseteq \mathbb{R}^d$ and the feature space is $\mathcal{Z} \subseteq \mathbb{R}^m$. The basis function $\phi(\mathbf{x}) : \mathcal{X} \to \mathcal{Z}$ transforms the original data, so we can perform a linear model on transformed data as a more complex model. A general extension of the linear regression model is

$$f(\mathbf{x};\mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + \cdots + w_m\phi_m(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x})$$

where $\mathbf{w} \in \mathbb{R}^m$.

2.3 Regularization

Summary

- Shrinkage method: ridge and lasso regularization

2.3.1 Overfitting and Complexity

Overfitting If a model overfits (too sensitive to data), it will be unstable, removing part of the data will change the fit significantly. More complex model (smaller "degree of freedom") overfits more than simple model.

Cross-Validation To reduce the risk of overfitting, we can hold out part of the data, called validation (val) set. We will fit the model to the rest and then test on the heldout data. ***k-fold cross-validation***: partition data into k roughly equal parts; train on all but j-th part, and test on j-th part.

Shrinkage Method We should penalize complexity to restrict the complexity.

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{N} \log p(\text{data}_i; \mathbf{w}) - \text{penalty}(\mathbf{w}) \right]$$

2.3.2 Regularization

Ridge Regression Shrinkage methods impose penalty on the size of $\mathbf{w}$. Using L_2 norm (Euclidean norm) penalization:

$$\mathbf{w}^*_{\text{ridge}} = \arg\max_{\mathbf{w}} \left[\sum_{i=1}^{N} \log p(\text{data}_i; \mathbf{w}) - \lambda \|\mathbf{w}\|^2 \right]$$

This is ***ridge regression***, λ is the regularization parameter.

$$\mathbf{w}^*_{\text{ridge}} = \arg\min_{\mathbf{w}} \left[\sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 + \lambda \sum_{j=1}^{m} w_j^2 \right] \tag{2.7}$$

For simplicity, $\mathbf{w} = [w_1, \cdots, w_m]$, then the closed form solution is

$$\widehat{\mathbf{w}}^*_{ridge} = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

note that the offset w_0 is usually not included in the regularization.

Careful: the solution is not invariant to scaling, so we should normalize input before solving to address this issue. For example, we can scale the data so that all features have the equal norm.

Lasso Regression The L_1-penalized (L_1 norm $\|\mathbf{w}\| = \sum_{j=1}^{m} |w_j|$) maximum likelihood under Gaussian noise model is

$$\mathbf{w}^*_{\text{ridge}} = \arg\min_{\mathbf{w}} \left[\sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 + \lambda \sum_{j=1}^{m} |w_j| \right] \tag{2.8}$$

This is ***Lasso Regression*** (least absolute shrinkage and selection operator). Lasso regression has no closed form (need numerical optimization tools).

Optimization We can view the optimization problem as minimizing

$$\min_{\mathbf{w}} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 \quad \text{subject to} \quad \|w_j\|^p \le t$$

The correspondence $\lambda \Rightarrow t$ can be shown using Lagrange multipliers. In 2D, the constrain for lasso is a square (with vertices on x and y axis) where the constrain for ridge is a circle. An equivalent formulation for L_p regularization is

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}:\sum_{j=1}^{m} |w_j|^p \le \beta} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2.$$

For $p > 1$, no sparsity is achieved, and for $p < 1$, the problem is non-convex. Choice of λ: if the λ is too small, overfitting will occur; conversely, underfitting occurs if λ is too large.

2.4 Recitation: MLE, MAP

Maximum Likelihood Estimation (MLE) Suppose we have N data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and the probability distribution function is $p(\mathbf{x}; \theta)$, or $p(y \mid \mathbf{x}; \theta)$. The plausibility of given data, fixing θ, is measured by the likelihood function. The likelihood function and log-likelihood are given by

$$L(\theta) = p(\mathbf{X}; \theta) = \prod_{i=1}^n p(\mathbf{x}_i; \theta), \qquad l(\theta) = \log P(\mathbf{X}; \theta) = \sum_{i=1}^n \log p(\mathbf{x}_i; \theta)$$

Maximum likelihood principle suggests we should pick θ that maximizes the likelihood (ML), equivalently, the log likelihood (by monotonicity)

$$\widehat{\theta}_{ML} = \operatorname*{argmax}_{\mu} \, p(\mathbf{X}; \theta) = \operatorname*{argmax}_{\mu} \, \log p(\mathbf{X}; \theta).$$

However, note that MLE doesn't work well if the sample size is small.

Maximum a Posteriori (MAP) In MLE, the parameter is viewed as a fixed unknown value, whereas in MAP (Bayesian), the oaramteter is viewed as a random variable. By Bayes rule,

$$p(\theta | \mathbf{X}) = \frac{p(\mathbf{X}|\theta) \, p(\theta)}{p(\mathbf{X})}$$

The maximum a-posteriori (MAP) estimate is defined as

$$\widehat{\theta}_{MAP} = \operatorname*{argmax}_{\mu} \, p(\theta | \mathbf{X}) = \operatorname*{argmax}_{\mu} \, p(\mathbf{X}|\theta) \, p(\theta)$$

since $p(\mathbf{X})$ is independent of θ. Choice of prior, $p(\theta)$:

- Bayesian approach - try to reflect our belief about θ,
- Utilitarian approach - choose a prior which is computationally convenient.

2.5 Gradient Descent

Summary

- Gradient descent algorithm

Gradient Descent The idea of gradient descent: start at a position and make steps in the direction of maximal altitude decease (the opposite direction of gradient), or equivalently, gradient descent on the convex loss $\log p(y \mid \mathbf{x}; \mathbf{w})$.

Gradient Descent Algorithm

1. Iteration counter $t = 0$.
2. Initialize $\mathbf{w}^{(t)}$ (to zero or small random vector).
3. For $t = 1, \cdots$, compute the gradient $\nabla f\left(\mathbf{X}, \mathbf{y}; \mathbf{w}^{(t-1)}\right)$ and update the model

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta \nabla f\left(\mathbf{X}, \mathbf{y}; \mathbf{w}^{(t-1)}\right) \tag{2.9}$$

   where the learning rate η controls the step size.

Gradient descent convergence usually relies on various properties of the objective function (strong convexity, smoothness, etc.).

3 Classification: Logistic Regression

3.1 Logistic Regression

Summary

- Loss, Surrogate Loss
- Logistic Regression MLE (Sigmoid Function, Gradient Descent)

3.1.1 Introduction to Classification

Linear Classifier (Binary Case) The standard form of linear classifier is given by

$$\widehat{y} = h(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w} \cdot \mathbf{x})$$

where $\mathbf{w}$ is the direction and w_0 is the location of the boundary. The decision boundary is orthogonal to $\mathbf{w}$, and the scaling on $\mathbf{w}$ changes the confidence but not the boundary. Classifying using regression effectively reduces the data dimension to 1.

Loss and Risk The expected loss (0/1) loss for classifier $h : \mathcal{X} \to \mathcal{Y}$ is

$$L(h(\mathbf{x}), y) = \begin{cases} 0 & \text{if } h(\mathbf{x}) = y \\ 1 & \text{if } h(\mathbf{x}) \neq y \end{cases}$$

The risk of a C-way classifier h is

$$R(h) = \mathbb{E}_{\mathbf{x},y}\left[L(h(\mathbf{x}), y)\right] = \int_{\mathbf{x}} \left[\sum_{c=1}^{C} L(h(\mathbf{x}), c)\, p(y = c \,|\, \mathbf{x})\right] p(\mathbf{x})\, d\mathbf{x},$$

so it is sufficient to minimize the conditional risk for any $\mathbf{x}$,

$$R(h \,|\, \mathbf{x}) = \sum_{c=1}^{C} L(h(\mathbf{x}), c)\, p(y = c \,|\, \mathbf{x}) = 1 - p(y = h(\mathbf{x}) \,|\, \mathbf{x}).$$

To minimize conditional risk given $\mathbf{x}$, the classifier must decide $h(\mathbf{x}) = \underset{c}{\text{argmax}}\; p(y = c \,|\, \mathbf{x})$.

Log-odds Ratio The optimal rule $h(\mathbf{x}) = c^* = \text{argmax}_c\, p(y = c \,|\, \mathbf{x})$ is equivalent to,

$$h(\mathbf{x}) = c^* \quad \Leftrightarrow \quad \frac{p(y = c^* \,|\, \mathbf{x})}{p(y = c \,|\, \mathbf{x})} \geq 1 \quad \Leftrightarrow \quad \log \frac{p(y = c^* \,|\, \mathbf{x})}{p(y = c \,|\, \mathbf{x})} \geq 0.$$

for all class c, since the probability of getting c^* is the greatest among all classes.

3.1.2 Logistic Regression MLE

Sigmoid Function The decision boundary can be modeled as $\log\left[p(y = 1 \,|\, \mathbf{x}) / p(y = 0 \,|\, \mathbf{x})\right] = 0$ for the binary case, or

$$\frac{p(y = 1 \,|\, \mathbf{x})}{1 - p(y = 1 \,|\, \mathbf{x})} = \exp(w_0 + \mathbf{w} \cdot \mathbf{x}) = 1$$

$$p(y = 1 \,|\, \mathbf{x}) = \frac{1}{1 + \exp(-w_0 - \mathbf{w} \cdot \mathbf{x})} = \frac{1}{2}$$

The logistic function is thus given by

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

which is a monotonic function, where $\sigma(0) = 1/2$ and $0 \leq \sigma(x) \leq 1$ for all x. the logistic function σ can turn the predicted value by regression into the likelihood.

The decision boundary is when $w_0 + \mathbf{w} \cdot \mathbf{x} = 0$, i.e., $p(y = 1 \,|\, \mathbf{x}) = \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}) = 1/2$. The decision boundary depends on the parameters: the offset w_0 will move the decision boundary, and $\mathbf{w}$ will change the orientation of the boundary and its slope.

Surrogate Loss and Maximum Likelihood Estimation The likelihood function is given by

$$\begin{aligned} p(y_i \,|\, \mathbf{x}; \mathbf{w}) &= \begin{cases} \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i) & \text{if } y_i = 1 \\ 1 - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i) & \text{if } y_i = 0 \end{cases} \\ &= \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i)^{y_i} (1 - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i))^{1-y_i} \end{aligned}$$

The log-likelihood of $\mathbf{w}$ is

$$\log p(Y \,|\, \mathbf{X}; \mathbf{w}) = \sum_{i=1}^{N} y_i \log \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i) + (1 - y_i) \log (1 - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x})) \tag{3.2}$$

Note that

$$\frac{\partial}{\partial z} \sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(z)(1 - \sigma(z)),$$

to maximize the likelihood,

$$\frac{\partial}{\partial w_0} \log p(Y \,|\, \mathbf{X}; \mathbf{w}) = \sum_{i=1}^{N} (y_i - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i)) = 0 \tag{3.3a}$$

$$\frac{\partial}{\partial w_j} \log p(Y \,|\, \mathbf{X}; \mathbf{w}) = \sum_{i=1}^{N} (y_i - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i)) x_{ij} = 0. \tag{3.3b}$$

We can treat $y_i - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i)$ as the prediction error of the model.

Note that instead of minimizing the 0/1 loss, we minimizes the ***log-loss***. This is a ***surrogate loss***.

3.1.3 Logistic Regression Gradient Descents

Gradient Ascents The gradient ascent algorithm for logistic regression is

$$\mathbf{w} := \mathbf{w} + \eta \sum_{i=1}^{N} (y_i - \sigma(w_0 + \mathbf{w} \cdot \mathbf{x}_i)) \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix} \tag{3.4}$$

Newton-Raphson The Newton-Raphson algorithm approximates the local shape of $\log p$ as a quadratic function.

$$\mathbf{w} \to \mathbf{w} + \mathbf{H}^{-1} \frac{\partial}{\partial \mathbf{w}} \log p(\mathbf{X}; \mathbf{w}),$$

where $\mathbf{H}$ is the ***Hessian*** matrix of second derivatives

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \log p}{\partial w_0^2} & \frac{\partial^2 \log p}{\partial w_0 w_1} & \cdots & \frac{\partial^2 \log p}{\partial w_0 w_d} \\ \frac{\partial^2 \log p}{\partial w_1 w_0} & \frac{\partial^2 \log p}{\partial w_1^2} & \cdots & \frac{\partial^2 \log p}{\partial w_1 w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \log p}{\partial w_d w_0} & \frac{\partial^2 \log p}{\partial w_d w_1} & \cdots & \frac{\partial^2 \log p}{\partial w_d^2} \end{bmatrix}$$

3.2 Recitation: Convex Optimization

Problem Setup f_0 is the objective function, and f_i $(i = 1, \cdots, m)$ are the constraint functions. The standard optimization formulation is

$$\begin{aligned} &\text{minimize}_w && f_0(\mathbf{w}) \\ &\text{subject to} && f_i(\mathbf{w}) \leq 0,\ i = 1, \cdots, m \end{aligned}$$

where $f_i(\mathbf{w})$ are all convex. General optimization problems are difficult to solve; however, certain problem classes can be solved efficiently and reliably.

Convex Sets A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for all $\lambda \in [0, 1]$ and $w_1, w_2 \in \mathbb{R}^d$, $f(\lambda w_1 + (1-\lambda)w_2) \leq \lambda f(w_1) + (1-\lambda)f(w_2)$. Equivalently, $f(y) \geq f(x) + \nabla f(x)^T(y-x)$, the first order condition.

Optimality Criterion If f_0 is differentiable then w is optimal if and only if it is feasible (w satisfies all constraints) and

$$\nabla f_0(w)^T(w_0 - w) \geq 0 \quad \text{for all feasible } w_0$$

For unconstrained problem, $\nabla f_0(\mathbf{w}) = 0$.

First Order Method: Gradient Descent Choose a starting point $\mathbf{w}_0$ and the desired tolerance ϵ. Repeat until $\|\nabla f(\mathbf{w}_t)\| \leq \epsilon$ is satisfied,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$

Note that the gradient descent is based on first order Taylor approximation $f(\mathbf{w}) = f(\mathbf{w}_1) + \nabla f(\mathbf{w}_1)(\mathbf{w} - \mathbf{w}_1) + O(\|\mathbf{w} - \mathbf{w}_1\|^2)$.

Stochastic Gradient Descent: Compute gradient at each iteration is expensive. SGD picks i and use $\nabla f_i(\mathbf{w})$ (stochastic gradient) instead of $\nabla F(\mathbf{w})$ to perform gradient descent. Small step size will guarantee the convergence to a local minimum.

Gradient Descent with Momentum: SGD has trouble navigating ravines, momentum is a method that helps accelerate SGD in the relevant direction nd dampens oscillations.

$$\begin{aligned} \Delta\mathbf{w}_t &= \gamma \Delta\mathbf{w}_{t-1} + \eta_t \nabla f(\mathbf{w}_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \Delta\mathbf{w}_t \end{aligned}$$

Second Order Methods: Hessian Hessian incorporates the local geometry of curvature of the error surface. For a function $f : \mathbb{R}^d \to \mathbb{R}$, the hessian matrix H is defined as

$$H_{ij} = \frac{\partial^2 f}{\partial w_i \partial w_j}$$

The hessian appears as a coefficient of the quadratic term in the Taylor expansion of f:

$$\widehat{f}(\mathbf{w} + \mathbf{v}) = f(\mathbf{w}) + \nabla f(\mathbf{w})^T \mathbf{v} + \frac{1}{2}\mathbf{v}^T H \mathbf{v}$$

Newton's Method: compute

$$w_{t+1} = w_t + \Delta w_{nt} = w_t - H^{-1} f(w_t)$$

until $\nabla f(w_t)^T H^{-1} \nabla f(w_1) \leq \varepsilon$. Newton's method and batch gradient can be expensive for large data size, stochastic gradient descent can be efficient.

3.3 SGD, Softmax

Summary

- Stochastic gradient descent
- Regularized logistic regression
- Softmax Model

3.3.1 Stochastic Gradient Descent (SGD)

SGD Intuition The disadvantage of gradient descent is computing gradient on all N examples is expensive. The intuition of SGD: assuming the gradient on the entire set is similar to the gradient on a single example ($\nabla L(w) \approx N \nabla L_i(w)$), then perform gradient descent on a single data each time.

SGD Algorithm Present examples $(\mathbf{x}_i, y)$ one at a time, modify $\mathbf{w}$ slightly to increase the log probability of observed y_i: (assuming $w_0 = 0$)

$$\mathbf{w} := \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log p(y_i \mid \mathbf{x}_i; \mathbf{w}) = \mathbf{w} + \eta \left(y_i - \sigma(\mathbf{w}^T \mathbf{x}_i) \right) \mathbf{x}_i \tag{3.5}$$

where η is the learning rate. ***Epoch*** (full pass through data) contains N updates instead of one. It is a good practice to shuffle the data.

3.3.2 Maximum a posteriori (MAP)

Overfitting with Logistic Regression We can get the same decision boundary with an infinite number of settings for $\mathbf{w}$ (since scaling $\mathbf{w}$ does not affects the decision boundary). Suppose the data are separable by $w_0 + \alpha \mathbf{w} \cdot \mathbf{x}$, with $\alpha \to \infty$, there is a continuum of w_0 that reach perfect separation, which meanwhile, reduce the likelihood to approaching 0.

MAP Estimation Intuition: we may have some belief about the value of $\mathbf{w}$, regardless of the data. (A prior of small $\mathbf{w}$ can serves as the regularization of logistic regression.)

A possible prior that captures that belief is $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, \sigma \mathbf{I})$. The objective becomes, instead of the log-likelihood $p(Y \mid \mathbf{X}; \mathbf{w})$, the ***log-posterior***:

$$\begin{aligned} \log p(Y \mid \mathbf{X}, \mathbf{w}; \sigma) &= \log p(Y \mid \mathbf{X}; \mathbf{w}) + \log p(\mathbf{w}; \sigma) \\ &= \sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i; \mathbf{w}) - \frac{1}{2\sigma^2} \sum_{j=1}^{d} w_j^2 + \text{const}(\mathbf{w}) \end{aligned}$$

where setting σ^2 affects the penalty on $\|\mathbf{w}\|$. Larger σ (smaller penalty) results in denser contours, and ML solution is attained when $\sigma \to \infty$.

3.3.3 Softmax

Logistic regression computes a score $f(\mathbf{x}; \mathbf{w}) = \mathbf{w} \cdot \phi(\mathbf{x})$, which is converted to posterior

$$p(y = 1 \mid \mathbf{x}) = \frac{\exp f(\mathbf{x}; \mathbf{w})}{1 + \exp f(\mathbf{x}; \mathbf{w})}.$$

The softmax model address the multi-class classification. Suppose we have C classes, and C scores $f_c(\mathbf{x}, \mathbf{W}) = \mathbf{w}_c \cdot \phi(\mathbf{x})$. To get posteriors from scores, exponentiate and normalize

$$p(y = c \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \phi(\mathbf{x}))}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \phi(\mathbf{x}))} \tag{3.6}$$

The posteriors are invariant to shifting scores. To address the common problem of overflow in $\exp(\mathbf{w}_c \cdot \phi(\mathbf{x}))$, we can subtract the exponent by $a = \max_c \mathbf{w}_c \cdot \phi(\mathbf{x})$ (so the max score is 0) and then exponentiate and normalize. Note that softmax is not needed for predictions.

4 Decision Trees

4.1 Classification and Regression Trees

Summary

- Regression Trees, Classification Trees
- Tree Pruning: weakest link pruning

4.1.1 Regression Trees

Partition The goal for decision trees is to deciding non-overlapping partition of the space into hyper-rectangles, and described the partition via a tree. Each leaf is associated with a fixed prediction value.

Decision: A point is placed in a region by moving it down the tree, applying the decision rule in each node.

Regression Predictor Model corresponding to a tree with M leaves; leaf $m \Rightarrow$ region $R_m \Rightarrow$ value f_m. The predictor is given by

$$f(\mathbf{x}) = \sum_{m=1}^{M} f_m \cdot 1_{\{\mathbf{x}\in R_m\}}$$

where $1_{\{\cdot\}}$ is the indicator function. To minimize the squared loss

$$L(Y \mid \mathbf{X}; f) = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 = \sum_{m=1}^{M} \sum_{x_i \in R_m} (f_m - y_i)^2,$$

we want to minimize $J(f_m) = \sum_{x_i\in R_m} (f_m - y_i)^2$ for each m. By setting derivative of $J(f_m)$ to 0, we obtain

$$f_m = \frac{1}{|\mathcal{I}_m|} \sum_{i\in\mathcal{I}_m} y_i \tag{4.1}$$

where $\mathcal{I}_m = \{i \mid \mathbf{x}_i \in R_m\}$; i.e., f_m is the average label of the training points in R_m.

Regression Tree Construction The goal is to find $R_1, \cdots, R_M$ to minimize $\sum_{i=1}^{N} (f_m \cdot 1_{\{\mathbf{x}_i\in R_m\}} - y_i)^2$, which is not computationally tractable. Greedy algorithm: consider a split at s along j-th feature,

$$R_L(j, s) = \{\mathbf{x} \mid \phi_j(\mathbf{x}) \le s\}, \quad R_U(j, s) = \{\mathbf{x} \mid \phi_j(\mathbf{x}) \ge s\}$$

The cost of the split, assigning $R_L \to f_L$, $R_U \to f_U$, is the sum of errors in two child trees:

$$\min_{f_L} \sum_{\mathbf{x}_i\in R_L} (y_i - f_L)^2 + \min_{f_U} \sum_{\mathbf{x}_i\in R_U} (y_i - f_U)^2$$

Therefore, we need to find j, s such that the cost is minimized. Since for any j, s, the f_L, f_U are the averages of the labels in induced R_L, R_U. We can exhaustively evaluate all distinct j, s pairs, and the complexity is $O(Nd)$.

4.1.2 Tree Pruning

Tree Pruning The decision tree can be easily over-fitted, gaining zero training errors. One way to limits model complexity is define notion of gain (reduction in loss) and don't split is the gain is small; however, this method can be short-sighted. Alternatively, we can grow a large tree T_0, then prune to $T \subset T_0$.

Let $|T|$ denotes the number of leaves, $N_m = |\mathcal{I}_m|$ is the size of a leaf, f_m is the value in the leaf, and $Q_m(T) = \frac{1}{N_m}\sum_{i\in\mathcal{I}_m}(y_i - f_m)^2$ be the leaf error. The cost-complexity criterion of tree $T \subset T_0$:

$$C_\lambda(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda|T| \tag{4.2}$$

note that the *goodness* $\sum_{m=1}^{|T|} N_m Q_m(T)$ is the overall square loss, and $\lambda|T|$ is the *complexity.* If $\lambda = 0$, then $T = T_0$; if $\lambda \to \infty$, T is a single node. For a given $\lambda \geq 0$, there exists a unique tree $T_\lambda = \operatorname{argmin}_T C_\lambda(T)$.

Weakest link pruning Intuition: keep collapsing the internal nodes that produce the smallest increase in cost $C(T) = \sum_m N_m Q_m(T)$, going from T_0 to a single node. For each nodes t, there is a λ_t such that

$$C(T') + \lambda_t|T'| = C(T) + \lambda_t|T| \quad \Leftrightarrow \quad \lambda_t = \frac{C(T) - C(T')}{|T'| - |T|}$$

where T' is the tree when collapsing t. That is, collapsing the node t will results in a gain in $C_\lambda(T)$ whenever $\lambda > \lambda_t$. The pruning algorithm is:

1. Starts with T_0. Find $\lambda_1 = \min(\lambda_t)$, gives T_1.
2. Repeat the procedure until T_R is a single node. Find the sequence of $\{(\lambda_i, T_i)\}_{i=1}^R$ where λ_i is strictly increasing.

For any λ, the tree T_i where $i = \operatorname{argmax}_i\{i \mid \lambda > \lambda_i\}$ is the optimal tree under λ.

4.1.3 Classification Trees

Classification Trees The fraction of example from class c in R_m: $\widehat{p}_{m,c} = \frac{1}{N_m}\sum_{i\in\mathcal{I}_m} 1_{y_i=c}$. Similarly, we want to minimize 0/1 loss, per leaf:

$$\operatorname*{argmin}_{\widehat{y}_m} \sum_{i\in\mathcal{I}_m} 1_{\{y_i \neq \widehat{y}_m\}},$$

and the solution is $\widehat{y}_m = \operatorname{argmax}_c \widehat{p}_{m,c}$.

Leaf Impurity In classification, the leaf impurity (square loss as an analogy in regression) can be measured by the misclassification rate or ***Gini Index***. The Gini index of leaf m in tree T:

$$Q_m(T) = \sum_{c=1}^{C} \widehat{p}_{m,c}(1 - \widehat{p}_{m,c}) \tag{4.3}$$

Common practice: use Gini to grow the tree and misclassification rate to prune.

4.1.4 Trees: Summary

The approach described above to regularize tree building is called CART (classification and regression trees). The advantages includes that CART is interpretable, can deal with non-numerical features naturally, and is naturally multi-class and non-linear. Limitations are hard split (non-smooth regression), limited to axis-aligned splits, and often have high variance despite regularization.

4.2 Recitation: Matrix Calculus

Derivative of Scalar The derivative of a scalar with respect to a vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is defined as

$$\frac{dy}{d\mathbf{x}} = \begin{bmatrix} \dfrac{dy}{dx_1} \\ \vdots \\ \dfrac{dy_1}{dx_n} \end{bmatrix} \in \mathbb{R}^{n\times 1}$$

The derivative of a scalar with respect to a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ is defined as

$$\frac{dy}{d\mathbf{X}} = \begin{bmatrix} \dfrac{dy}{d\mathbf{x}_1} & \vdots & \dfrac{dy}{d\mathbf{x}_n} \end{bmatrix} = \begin{bmatrix} \dfrac{dy}{dx_{11}} & \cdots & \dfrac{dy}{dx_{n1}} \\ \vdots & \ddots & \vdots \\ \dfrac{dy}{dx_{m1}} & \cdots & \dfrac{dy}{dx_{mn}} \end{bmatrix} \in \mathbb{R}^{m\times n}$$

Derivative of Vector Suppose $\mathbf{y} \in \mathbb{R}^{n\times 1}$. The derivative of $\mathbf{y}$ with respect to a scalar

$$\frac{d\mathbf{y}}{dx} = \begin{bmatrix} \dfrac{dy_1}{dx} & \cdots & \dfrac{dy_n}{dx} \end{bmatrix} \in \mathbb{R}^{1\times n}$$

The derivative of $\mathbf{y}$ with respect to a vector $x \in \mathbb{R}^{m\times 1}$

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{bmatrix} \nabla y_1(\mathbf{x}) & \cdots & \nabla y_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \dfrac{dy_1}{dx_1} & \cdots & \dfrac{dy_n}{dx_1} \\ \vdots & \ddots & \vdots \\ \dfrac{dy_1}{dx_m} & \cdots & \dfrac{dy_n}{dx_m} \end{bmatrix} \in \mathbb{R}^{m\times n}$$

5 Ensemble Methods

5.1 Boosting

Summary

- AdaBoost

5.1.1 Boosting Introduction

Combining Classifiers The general formulation to combine classifiers $h_1, \cdots, h_m : \mathcal{X} \to \{\pm 1\}$ is the function $H : \mathcal{X} \to [-1, 1]$ defined as

$$H(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \cdots + \alpha_m h_m(\mathbf{x})$$

where α_j is the vote assigned to classifier h_j, and it should be higher for more reliable classifiers. The prediction is made by $\widehat{y}(\mathbf{x}) = \text{sign } H(\mathbf{x})$.

Greedy Assembly Suppose the family of classifiers $\mathcal{H}$ is parameterized by θ. The intuition is

- Set the initial classifier θ_1 to minimize the training error (using surrogate for the 0/1 loss).
- Set the following classifiers $\theta_2, \cdots$ to minimize the (surrogate) loss of the combination $\sum_{i=1}^{N} L(H(\mathbf{x}_i), y_i)$.

Surrogate Loss: Exponential Loss ***Exponential loss***, another surrogate loss, is given by

$$L(H(\mathbf{x}), y) = e^{-y \cdot H(\mathbf{x})}, \qquad L(H, X) = \sum_{i=1}^{N} e^{-y_i \cdot H(\mathbf{x}_i)}$$

Note that it assigns loss to the data even the prediction is correct.

5.1.2 AdaBoost

Intuition The intuition of greedy algorithm for $m = 1, \cdots, M$ is

1. Maintain weights (weight distribution of the data points) $W_i^{(m)}$, initially all $1/N$
2. Pick a weak classifier h_m, minimizing error ϵ_m weighted by $W^{(m-1)}$
3. Set the vote (α_m) for h_m
4. Update weights W_i^m based on mistakes of h_m on α_m

The final (strong) classifier is $\text{sign}(\sum_m \alpha_m h_m)$.

Optimizing Weak Learner Suppose h_i are weak classifiers ($\mathcal{Y} = \{\pm 1\}$) and denote $H_m(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \cdots + \alpha_m h_m(\mathbf{x})$. Suppose we add $\alpha_m \cdot h_m(\mathbf{x})$ to H_{m-1},

$$L(H_m, X) = \sum_{i=1}^{N} e^{-y_i H_{m-1}(\mathbf{x}_i)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

Define the weight as the loss after $m-1$ iterations $W_i^{(m-1)} = e^{-y_i H_{m-1}(\mathbf{x}_i)}$, which captures the history of classification of $\mathbf{x}_i$ by H_{m-1}. Optimization choose $\alpha_m, h_m = h(\mathbf{x}; \theta_m)$ that minimize the (weighted) exponential loss at iteration m.

The ensemble loss is

$$L(H_m, X) = \sum_{i=1}^{N} W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)} = e^{-\alpha_m} \sum_{i:y_i = h_m(\mathbf{x}_i)} W_i^{(m-1)} + e^{\alpha_m} \sum_{i:y_i \neq h_m(\mathbf{x}_i)} W_i^{(m-1)}$$

For any $\alpha_m > 0$, $e^{-\alpha_m} < e^{\alpha_m}$, so minimizing L is equivalent to minimizing training error weighted by $W^{(m-1)}$.

Normalizing the weights gives

$$W_i^{(m-1)} = \frac{\exp\left[-y_i H_{m-1}(\mathbf{x}_i)\right]}{\sum_{j=1}^{N} \exp\left[-y_j H_{m-1}(\mathbf{x}_j)\right]} \tag{5.1}$$

Optimizing Votes The weighted error of h_m is

$$\epsilon_m = \sum\nolimits_{i:y_i \neq h_m(\mathbf{x}_i)} W_i^{(m-1)}$$

Note that $L(h_m; \mathbf{X}) = e^{-\alpha_m}(1-\epsilon_m) + e^{\alpha_m}\epsilon_m$ thus $\frac{\partial L}{\partial \alpha_m} = -e^{-\alpha_m}(1-\epsilon_m) + e^{\alpha_m}\epsilon_m$. Given h_m and its ϵ_m, the optimal α_m that minimized the exponential loss is

$$\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m} \tag{5.2}$$

as long as $\epsilon_m < 1/2$, $\alpha_m > 0$.

5.1.3 AdaBoost: Algorithm

1. Initialize weights $W_i^{(0)} = 1/N$.
2. Iterator for $m = 1, \cdots, M$:
   - Find weak classifier h_m that attains weighted error

     $$\epsilon = \frac{1}{2}\left(1 - \sum_{i=1}^{N} W_i^{(m-1)} y_i h_m(\mathbf{x}_i)\right) < \frac{1}{2}$$

   - Calculate the vote α_m

     $$\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$$

   - Update the weights and normalize so that $\sum_i W_i^{(m)} = 1$:

     $$W_i^{(m)} = \frac{1}{Z} W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

     where Z is the normalization constant $Z = \sum_i W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$.
3. The combined classifier: sign $\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$

5.1.4 More on Boosting

AdaBoost Behavior Training error: The training error in m-th iteration is bounded above by $\prod_{j=1}^{m} \sqrt{2\epsilon_m(1-\epsilon_m)}$. Test error: the test can error can still decrease after training error is flat (even zero).

Typical behavior: training error of H goes up and votes α_m goes down, and weighted error ϵ_m goes up. We can regularize via (1) early stopping or (2) regularization of weak learners.

Boosting the Margin: Suppose the classifiers are $h_m : \mathbf{x} \to \pm 1$. Define the ***margin*** of an example as

$$\gamma(\mathbf{x}_i) = y_i \cdot \frac{\alpha_1 h_1(\mathbf{x}_i) + \cdots \alpha_m h_m(\mathbf{x}_i)}{\alpha_1 + \cdots + \alpha_m}$$

$\gamma(\mathbf{x}_i) \in [-1, 1]$, positive if and only if sign $H(\mathbf{x}_i) = y_i$; this is a measure of confidence in the correct decision. Iterations of AdaBoost increase the margin of training examples even the training error is flat.

Boosting: Additive Regression View Additive form for a classifier: $H(\mathbf{x}) = \sum_{m \in \mathcal{M}} \alpha_m h_m(\mathbf{x})$ for some countable set $\mathcal{M}$. Boosting is a greedy algorithm for fitting $\boldsymbol{\alpha}$, under sparsity constraint $\|\boldsymbol{\alpha}\|_0 \leq M$. Another view is coordinate descent in the space of H.

Variations of Boosting Variations include:

- Different surrogate loss function.
- Confidence rated version: $h(\mathbf{x}) \in [-1, 1]$ instead of $\{\pm 1\}$.
- FloatBoost: after each round (having added a weak classifier), see if removal of a previously added classifier is helpful.
- Totally corrective AdaBoost:update the α for all weak classifiers one done.

Boosting Stumps Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \cdots, \phi_d(\mathbf{x})]^T$, the decision stump is a simple classifier with linear and axis parallel decision boundary

$$\theta_{j,r} = \begin{cases} 1 & \text{if } \phi_j(\mathbf{x}) \geq r \\ -1 & \text{if } \phi_j(\mathbf{x}) < r \end{cases}$$

Boosting the decision stumps can yields an accurate model.

Bias-Variance Tradeoff Denote $H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$. The complexity is determined by the weak classifiers h_m and their number M. Typically we should prefer simple weak classifiers: stumps, shallow decision trees. Regularization of H: early stopping, or controlling the complexity of h_m.

5.2 Stepwise Regression, Gradient Boosting, Random Forests

Summary

- Gradient Boosting (Stepwise Regression)
- Bagging: Random Forest

5.2.1 Stepwise Regression

Stepwise Regression Intuition The linear regression model can be viewed as a combination of simple regressors $F_d(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x}; \mathbf{w})$ where $f_j(\mathbf{x}; \mathbf{w}) = w_j \phi_j(\mathbf{x})$. Parameterize the set of functions: $f(\mathbf{x}; \theta)$ where $\theta = [w, j]$, we can build this combination greedily: pick the best feature $\phi_m(\mathbf{x})$ to construct a regressor $w_m \phi_m(\mathbf{x})$, and build on the regressor by minimizing the residuals through other features.

Stepwise Regression Algorithm The algorithm is given by

1. Fit the first simple model
$$\theta_1 = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i; \theta))^2$$
2. Fit other simple models to minimize the residuals of the previous step:
$$\theta_j = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} ((y_i - F_j(\mathbf{x}_i)) - f(\mathbf{x}_i; \theta))^2$$
3. Stop when no significant improvement in training error. The final estimate after M steps is:
$$\widehat{y}(\mathbf{x}) = F_M(\mathbf{x}; \mathbf{w}) = f_1(\mathbf{x}; \theta_1) + \cdots + f_M(\mathbf{x}; \theta_M)$$

Equivalence to $\mathbf{L_0}$ Regularization The ensemble regressor $y = \sum_{j=1}^{M} f(\mathbf{x}; \theta_j)$ where $\theta_m = \operatorname{argmin}_\theta L(\{\theta_1, \cdots, \theta_{m-1}, \theta\}, \mathbf{X})$ is a greedy approximation to $\operatorname{argmin}_\theta L(\boldsymbol{\theta}, \mathbf{X})$ such that $\|\boldsymbol{\theta}\|_0 \le M$, equivalently
$$\operatorname{argmin}_\theta L(\boldsymbol{\theta}, \mathbf{X}) + \lambda \|\boldsymbol{\theta}\|_0$$

5.2.2 Gradient Boosting

Generic Algorithm

1. Start with initial best constant fit $F_1(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} y_i$
2. For $m = 1, \cdots$, until convergence, calculate the negative gradients for each $i = 1, \cdots, N$
$$-g(\mathbf{x}_i) = \frac{\partial L(y_i, F_m(\mathbf{x}_i))}{\partial F_M(\mathbf{x}_i)} = y_i - F_m(\mathbf{x}_i)$$
fit a regression function f_{m+1} to negative gradients $f_{m+1}(\mathbf{x}_i) \approx -g(\mathbf{x}_i)$, update model $F_{m+1} = F_m + \eta f_{m+1}$.

We can modify L to derive new gradient boosting algorithm.

Classification Gradient Boosting With C-way classification, F predicts a matrix (let $F_c(\mathbf{x}_i)$ denotes the prediction score of class c for the data point $\mathbf{x}_i$). Negative gradients are aslos a matrix, with entry at (c, i)
$$-g_c(\mathbf{x}_i) = -\frac{\partial L}{\partial F_c(\mathbf{x}_i)}$$

Gradient boosting: start with $F_1^{(1)}$, at each iteration, m fit $f_c^{(m+1)}$ to negative gradients $-g_c(\mathbf{x}_1), \cdots,$ $-g_c(\mathbf{x}_N)$.

Step Size Model update in GB: $F_{m+1} = F_m + \eta f_{m+1}$.

- An aggressive strategy: steepest descent in the direction of f_{m+1}. Once we fit f_{m+1}, solve $\eta_m = \text{argmin}_\eta \sum_i L(y_i, F_m(\mathbf{x}_i) + \eta f_{m+1}(\mathbf{x}_i))$
- Alternative strategy (may regularize better): fixed $\eta < 1$, or decaying η_m (ex. $\eta_m = n_0/\sqrt{m}$).

Example: XGBoost Let m denote the leaf, t be the tree, $w_{m,c}^t$ be the value (score) of the leaf, and $Q_t(\mathbf{x}_i)$ be the index of the leaf where $\mathbf{x}_i$ falls in tree. Given T trees, the score of class c is given by the sum of scores in T trees:

$$F_c^{(T)}(\mathbf{x}_i) = \sum_{t=1}^{T} w_{Q_t(\mathbf{x}_i),c}^t$$

XGBoost: in iteration t, minimize regularized objective

$$\min_{f_t} \sum_{i=1}^{N} L\left(y_i, F^{(t)}(\mathbf{x}_i) + f_t(\mathbf{x}_i)\right) + \gamma|f_t| + \lambda\|\mathbf{w}^t\|^2$$

where $|f_t|$ is the size of the tree added in this iteration and $\mathbf{w}^t$ is the parameter (score) vector for the tree (all classes and all leaves).

5.2.3 Random Forests

Random forest is a ***bagging*** approach (bootstrap aggregation).

Intuition Deep decision trees have low bias but high variance, and CART pruning often leads to poor bias/variance tradeoff, so we want to let trees be deep (low bias), averages many trees (low variance).

Random Forests Suppose we have N data points and d features. Injecting randomness into tree construction ensures diversity thus low variance. For each tree, we introduce two sources of randomness:

- Bootstrap sampling: sample N with replacement (some points appear more than once, and approx. 37% will not appear, out of bag) or $N' < N$ without replacement.
- Sampling feature in each node, when considering splits, only look at a random $m < d$ features. Recommended heuristic value of m is $\sqrt{d}$ for classification, and $d/3$ for classification.

Each tree is less likely to overfits since over-fittings are likely to cancel out in averaging. To make prediction: average (regression) or vote.

Tuning parameters: number of trees T, feature set size m, tree depth / min number in leaves.

Bagging in General Bagging reduce the variance through averaging (best with unstable nonlinear predictors, e.g., trees). The “out of bag” (OOB) data can be used as validation set.

6 Support Vector Machines

6.1 Recitation: Constrained Optimization

Equality Constraint Suppose the constraint problem is

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && G(\mathbf{x}) = 0. \end{aligned}$$

$\mathbf{x}^*$ is optimal if there exists $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = \lambda \nabla G(\mathbf{x}^*)$ (λ is the Lagrange multiplier). Define the ***Lagrangian***:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda G(x).$$

The optimality condition is equivalent to

$$\nabla L(\mathbf{x}^*, \lambda^*) = \begin{bmatrix} \nabla f(\mathbf{x}^*) + \lambda^* \nabla G(\mathbf{x}^*) \\ G(x^*) \end{bmatrix} = 0$$

Inequality Constraint Suppose the constraint problem is

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && G_1(\mathbf{x}), \cdots, G_n(\mathbf{x}) \le 0. \end{aligned}$$

Note that all constraints are equivalent to this form: $G(\mathbf{x}) \ge 0 \Leftrightarrow -G(\mathbf{x}) \le 0$, and $G(\mathbf{x}) = 0 \Leftrightarrow G(\mathbf{x}) \le 0$ and $-G(\mathbf{x}) \le 0$.

We can introduce $\max_{\lambda_i \ge 0} \lambda_i G_i(\mathbf{x})$ as the zero-infinity penalty functions on $G_i(\mathbf{x})$, which evaluates to ∞ if the condition is violated and zero otherwise. The optimality condition is equivalent to $\min_{\mathbf{x}}[f(x) + \sum_i \max_{\lambda_i \ge 0} \lambda_i G_i(\mathbf{x})]$. Define the Lagrangian

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_i \lambda_i G_i(\mathbf{x})$$

Then the primal problem become the dual problem

$$\min_{\mathbf{x}} \max_{\lambda_i \ge 0} L(\mathbf{x}, \boldsymbol{\lambda})$$

KKT Condition The KKT condition states that the following conditions

1. Primary feasibility: $G_i(\mathbf{x}^*) \le 0$
2. Dual feasibility: $\lambda_i^* \ge 0$
3. Complementary slackness: $\lambda_i^* G_i(\mathbf{x}^*) = 0 \; \forall \, i$
4. Stationarity: $\nabla L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0$

are sufficient to the optimality of the dual problem

$$\max_{\boldsymbol{\lambda} \ge 0} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}).$$

Duality The dual function is $d(\lambda) = \min_{\mathbf{x}} L(\mathbf{x}, \lambda)$, and the dual problem is

$$d^* = \max_{\lambda \ge 0} d(\lambda) = \max_{\lambda \ge 0} \min_{\mathbf{x}} L(\mathbf{x}, \lambda)$$

Weak duality, $d^* \leq p^*$ (where $p^* = \min_{\mathbf{x}} \max_{\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda})$ is the optimal $f(\mathbf{x})$ under the constraints), always holds. Strong duality holds if

$$\max_{\lambda \geq 0} \min_{\mathbf{x}} L(\mathbf{x}, \lambda) = \min_{\mathbf{x}} \max_{\lambda \geq 0} L(\mathbf{x}, \lambda),$$

and under strong duality, KKT is the necessary and sufficient to optimality. ***Slaker's Condition***: Strong duality holds if

- f, G_i are convex, and
- $\exists\, \mathbf{x},\ \forall\, i,\ G_i(\mathbf{x}) < 0.$

Given a constrained optimization problem, if strong duality holds and the KKT conditions will yield the solution or a simpler optimization problem if both conditions hold.

6.2 Support Vector Machines

Summary

- Support Vector Machine (SVM)
- Dual Problem and KKT
- SVM with Slack (Hinge Loss)

6.2.1 Max-Margin Classification and SVM

Classification Margin Discriminant function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0$. The distance from a correctly classified $(\mathbf{x}, y)$ to the boundary is $d = y(\mathbf{w} \cdot \mathbf{x} + w_0)/\|\mathbf{w}\|$, which is constant under scaling.

Margin of the classifier on $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, assuming X is separable, is the distance to the closed point $\min_i d_i = \min_i y_i(\mathbf{w} \cdot \mathbf{x} + w_0)/\|\mathbf{w}\|$. The motivation of support vector machines is to find the classifier that attains the largest margin, namely

$$\text{argmax}_{\mathbf{w},w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i y_i(\mathbf{w} \cdot \mathbf{x} + w_0) \right\}.$$

Optimal Separating Hyperplane We can set $\min_i y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 1$ (so the positive and negative hyperplane are $\mathbf{w} \cdot \mathbf{x} + w_0 = \pm 1$) by rescale $\|\mathbf{w}\|$, w_0. Then the optimization becomes $\text{argmax}_{\mathbf{w},w_0} 1/\|\mathbf{w}\|$ such that $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \; \forall \, i$. Equivalently,

$$\underset{\mathbf{w},w_0}{\text{argmin}} \, \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad \forall \, i, \; y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) - 1 \geq 0.$$

Solving by Lagrange Multiplier Using Lagrange Multiplier method with inequality constraint, KKT condition, and Strong duality (Refer to Section 6.1), we can reformulate the problem as

$$\min_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^N \max_{\lambda_i \geq 0} \lambda_i \left[1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)\right] \right\} \quad \Leftrightarrow$$

$$\max_{\boldsymbol{\lambda} \geq 0} \min_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^N \lambda_i \left[1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)\right] \right\} \tag{6.1}$$

where the first expression comes from the Lagrange multiplier and the second expression is followed by the strong duality guaranteed by Slaker's Theorem. Define $J(\mathbf{w}, w_0; \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^N \lambda_i \left[1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)\right]$ to be the Lagrangian.

Strategy for optimization:

1. Minimize $J(\mathbf{w}, w_0; \boldsymbol{\lambda})$ as a function of $\mathbf{w}, w_0$ (fixed $\boldsymbol{\lambda}$): find $\mathbf{w}(\boldsymbol{\lambda}), w_0(\boldsymbol{\lambda})$ as a function of $\boldsymbol{\lambda}$
2. Maximize $J(\mathbf{w}(\boldsymbol{\lambda}), w_0(\boldsymbol{\lambda}); \boldsymbol{\lambda})$ as a function of $\boldsymbol{\lambda}$: find $\boldsymbol{\lambda}^*$
3. Find $\mathbf{w}^*, w_0^*$ by substitution: $\mathbf{w}^* = \mathbf{w}(\boldsymbol{\lambda}^*)$ and $w_0^* = w_0(\boldsymbol{\lambda})$

Minimizing J with respect to $\mathbf{w}, w_0$ For fixed $\boldsymbol{\lambda}$, we can minimize J by setting derivatives w.r.t. $w_0, \mathbf{w}$ to 0:

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}, w_0; \boldsymbol{\lambda}) = \mathbf{w} - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial}{\partial w_0} J(\mathbf{w}, w_0; \boldsymbol{\lambda}) = \sum_{i=1}^N \lambda_i y_i = 0.$$

Therefore,

$$\mathbf{w}(\boldsymbol{\lambda}) = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i, \qquad \sum_{i=1}^{N} \lambda_i y_i = 0$$

where the first equality can be characterized by the Representer's Theorem

Solving for $\boldsymbol{\lambda}$ Substitute the result from previous step into Lagrangian yields

$$\begin{aligned}
J^*(\mathbf{x}) &= \max_{\boldsymbol{\lambda}\geq 0, \sum \lambda_i y_i = 0} J(\mathbf{w}(\boldsymbol{\lambda}), w_0(\boldsymbol{\lambda}); \boldsymbol{\lambda}) \\
&= \max_{\boldsymbol{\lambda}\geq 0, \sum \lambda_i y_i = 0} \left\{ \frac{1}{2} \left(\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i^T \right) \left(\sum_{j=1}^{N} \lambda_j y_j \mathbf{x}_j \right) + \sum_{i=1}^{N} \left[1 - \lambda_i y_i \left(\left(\sum_{j=1}^{N} \lambda_j y_j \mathbf{x}_j \right) \mathbf{x}_i^T + w_0 \right) \right] \right\} \\
&= \max_{\boldsymbol{\lambda}\geq 0, \sum \lambda_i y_i = 0} \left\{ \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_i \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_i \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^{N} \lambda_i y_i w_0 \right\} \\
&= \max_{\boldsymbol{\lambda}\geq 0, \sum \lambda_i y_i = 0} \left\{ \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \right\}
\end{aligned}$$

Max-Margin and Quadratic Programming The max-margin problem is the dual program in $\boldsymbol{\lambda}$:

$$\max_{\boldsymbol{\lambda}\geq 0, \sum \lambda_i y_i = 0} \left\{ \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \right\} \tag{6.2}$$

Solving the quadratic program yields $\boldsymbol{\lambda}^*$, we can substitute $\boldsymbol{\lambda}^*$ back to get $\mathbf{w}$:

$$\widehat{\mathbf{w}} = \mathbf{w}(\boldsymbol{\lambda}^*) = \sum_{i=1}^{N} \lambda_i^* y_i \mathbf{x}_i.$$

Support Vectors Suppose under the optimal solution, the margin of a particular $\mathbf{x}_i$ is $y_i(\widehat{\mathbf{w}} \cdot \mathbf{x}_i + w_0) > 1$, then necessarily $\lambda_i^* = 0$, so $\mathbf{x}_i$ is not a support vector. That is, $\mathbf{x}_i$ is a support vector if and only if $\widehat{\mathbf{w}} \cdot \mathbf{x} + w_0 = 1$, and the affine plane which $\mathbf{x}_i$ lies on is the marginal hyperplane.

6.2.2 SVM with Slack

Non-separable Case In non-separable case, the constraint violation is unavoidable $(1 - y_i(\mathbf{w} \cdot \mathbf{x} + w_0 < 0$ for some $\mathbf{x})$, so the constraint $\boldsymbol{\lambda} \geq 0$ will cause $J \to \infty$. We will set maximum penalty on constraint violation $0 \leq \boldsymbol{\lambda} \leq C$.

Slack Variables We introduce slack variables to satisfy margin constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) - 1 + \xi_i \geq 0, \quad \xi_i \geq 0$$

ξ_i (as a function of $\mathbf{w}$) captures the minimum amount to fix, $\xi_i = \max\{0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)\}$. The objective function becomes

$$\min_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \right\} \tag{6.3}$$

subject to $y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) - 1 + \xi_i \geq 0$ and $\xi_i \geq 0$. Introduce additional multipliers $\boldsymbol{\mu}$ for the $\boldsymbol{\xi} \geq 0$, we have the Lagrangian

$$L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\lambda_i\left(1 - \xi_i - y_i(\mathbf{w}\cdot\mathbf{x} + w_0)\right) - \sum_{i=1}^{N}\mu_i\xi_i$$

By the stationarity, namely $\nabla L = 0$, we have

- $\partial L/\partial\mathbf{w} = \mathbf{w} - \sum_i \lambda_i y_i \mathbf{x}_i = 0$, thus $\mathbf{w} = \sum_i \lambda_i y_i \mathbf{x}_i$
- $\partial L/\partial w_0 = \sum_i \lambda_i y_i = 0$
- $\partial L/\partial \xi_i = C - \lambda_i - \mu_i = 0$, thus $C = \lambda_i + \mu_i$

Solving the optimization by using Lagrange multipliers results in the dual problem

$$\max_{\boldsymbol{\lambda}}\left\{\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j\right\}$$

$$\text{subject to } \sum_{i=1}^{N}\lambda_i y_i = 0,\ 0 \leq \boldsymbol{\lambda} \leq C.$$

Note that

1. When $0 < \lambda_i^* < C$, then $\xi_i = 0$ thus $y_i(\mathbf{w}\cdot\mathbf{x} + w_0)$, namely $\mathbf{x}_i$ is on the marginal hyperplane.
2. When $\lambda_i^* = C$, $y_i(\mathbf{w}\cdot\mathbf{x} + w_0) = 1 - \xi_i$, namely $\mathbf{x}_i$ lies inside the margin or miss-classified.

6.2.3 SVM in the Primal

Surrogate Loss: Hinge Loss $\xi_i = \max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0)\}$ is the ***hinge loss***.

Solving SVM in the Primal Setting $\lambda = 2/C$ we get the objective function

$$\text{primal:} \quad \min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N}\max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0)\}$$

we will solve by subgradient descent, since hinge loss is not differentiable.

Subgraident Descent The subgradient of L at $\mathbf{w}$ is any g such that $\forall\ \mathbf{w}' :\ L(\mathbf{w}') \geq L(\mathbf{w}) + g(\mathbf{w}' - \mathbf{w})$ (i.e., g defines a tight linear lower bound). Subdifferential of L at $\mathbf{w}$ is $\partial L(\mathbf{w}) = \{g \mid g \text{ is a subgradient of } L \text{ at } \mathbf{w}\}$; if L is differentiable, then $\partial L(\mathbf{w}) = \nabla L(\mathbf{w})$.

Let $L_i(\mathbf{w}, w_0) = \max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x} + w_0)\}$, the the subgradient of the hinge loss on $(\mathbf{x}_i, y_i)$ is

$$\nabla_{\mathbf{w}} L_i(\mathbf{w}, w_0) = \begin{cases} -\,y_i\mathbf{x}_i, & \text{if } y_i(\mathbf{w}\cdot\mathbf{x} + w_0) < 1 \\ 0, & \text{if } y_i(\mathbf{w}\cdot\mathbf{x} + w_0) < 1 \end{cases}.$$

Similarly compute for $\partial L_i/\partial w_0$ and perform the gradient descent. Remember to add gradient of the regularizer.

6.3 Kernel Methods

Summary

- Kernel Trick
- Kernels: Polynomial, RBF
- Regression SVM and Multiclass SVM

6.3.1 Motivations

SVM and Dot Product Similarity Suppose $|\mathbf{u}| = |\mathbf{v}| = 1$, the dot product measures angle between then, $\mathbf{u} \cdot \mathbf{v} = \cos(\mathbf{u}, \mathbf{v})$; this is a measure of similarity. SVM prediction

$$\widehat{y} = \text{sign}\left(\widehat{w}_0 + \sum_{\lambda_i > 0} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x}\right)$$

can be interpreted as each SV $\mathbf{x}_i$ votes for $\mathbf{x}$ to be assigned to class y_i, and it is modulated by similarity to $\mathbf{x}$ measured by the dot product. We often normalize every example to unit length before training (especially for sparse, high dimensional data).

Feature Space Data can be mapped into nonlinear feature space so the data can be easily separable by a hyperplane, and we can then compute the inner product and the SVM. Indeed, we are able to calculate dot product in the feature space implicitly using ***kernel*** K (without perform feature expansion).

Kernel Trick Given feature map $\phi : \mathcal{X} \to \mathbb{R}^D$, there exists $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Replacing dot products in SVM formulation with kernel values, the optimization problem and the classifier becomes

$$\max\left\{\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\right\} \qquad \text{and} \qquad \widehat{y} = \text{sign}\left(\widehat{w}_0 + \sum_{\lambda_i > 0} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x})\right),$$

and note that we need to compute the kernel matrix for the training data and $K(\mathbf{x}_i, \mathbf{x})$ for all i. Also, we have

$$\|\mathbf{w}\|^2 = B^T K B$$

where $K \in \mathbb{R}^{N \times N}$ and $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.

6.3.2 Kernels

Representer Theorem Suppose $\mathbf{w}^*$ is the minimizer $\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \|\mathbf{w}\|^2$ such that $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1$ for all i, then the solution can be represented as

$$\mathbf{w}^* = \sum_{i=1}^{N} \beta_i \mathbf{x}_i.$$

Proof: Suppose $\mathbf{w}^* = \mathbf{w}_X + \mathbf{w}_\perp$ where $\mathbf{w}_X \in \text{span}(\mathbf{x}_1, \cdots, \mathbf{x}_N)$, and $\mathbf{w}_\perp \notin \text{span}(\mathbf{x}_1, \cdots, \mathbf{x}_N)$ thus $\mathbf{w}_\perp \cdot \mathbf{x}_i = 0$ for all i. Then $y_i(\mathbf{w}^* \cdot \mathbf{x}_i + w_0) = y_i(\mathbf{w}_X \cdot \mathbf{x}_i + \mathbf{w}_\perp \cdot \mathbf{x}_i + w_0) = y_i(\mathbf{w}_X \cdot \mathbf{x}_i + w_0) \geq 1$ for all i, so $\mathbf{w}_X$ is a feasible solution. Now $\|\mathbf{w}^*\|^2 = \|\mathbf{w}_X\|^2 + \|\mathbf{w}_\perp\|^2 + 2\mathbf{w}_X \cdot \mathbf{w}_\perp = \|\mathbf{w}_X\|^2 + \|\mathbf{w}_\perp\|^2 \geq \|\mathbf{w}_X\|^2$. Hence $\mathbf{w}^* = \mathbf{w}_X$ is the optimal solution.

Remark: The Representer theorem implies that

$$\widehat{\mathbf{w}} = \sum_{i=1}^{N} \lambda_i y_i \phi(\mathbf{x}_i)$$

Kernel SVM in the Primal Since $\|\mathbf{w}^2\| = \sum_{i,j=1}^{N} \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, by SVM in the primal (Section 6.2.3), the learning objective is (define $[a]_+ = \max\{0, a\}$),

$$\min_{\boldsymbol{\lambda}} \left\{ \frac{\lambda}{2} \sum_{i,j=1}^{N} \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_i \left[1 - y_i \sum_j \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \right]_+ \right\}$$

Mercer's Kernels Theorem due to Mercer (1909): K must be

1. continuous;
2. symmetric $K(\mathbf{x}, \mathbf{z}) = K(\mathbf{z}, \mathbf{x})$, and
3. positive definite: for any $\mathbf{x}_1, \cdots, \mathbf{x}_N$, the kernel matrix K, where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ must be positive definite (i.e., $\mathbf{x}^T K \mathbf{x} > 0$ for all nonzero $\mathbf{x}$).

Popular Kernels

1. ***Linear Kernel***: $K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$, leading to the original linear SVM.
2. ***Polynomial Kernel***: $K(\mathbf{x}, \mathbf{z}; b, p) = (b + \mathbf{x} \cdot \mathbf{z})^p$.
3. ***Radial Basis Function (RBF) Kernel***:

$$K(\mathbf{x}, \mathbf{z}; \sigma) = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{z}\|^2 \right)$$

The RBF kernel is a measure of similarity between two examples. (RBF approaches 0 if two examples are very different and is 1 if they are identical.) The feature space is infinite-dimensional (Taylor expansion) thus any data will be separable.

The parameter σ regularize overfitting: small σ likely to overfits the data, and small σ likely to underfits the data.

6.3.3 SVM Regression and Multiclass Classification

SVM Regression The key idea is use an ϵ-tube (data points inside the $\widehat{y} \pm \epsilon$ tube are not support vectors), and introduce two sets of slack variables:

$$\begin{cases} y_i \le f(\mathbf{x}_i) + \epsilon + \xi_i, & \xi_i \ge 0 \\ y_i \ge f(\mathbf{x}_i) - \epsilon - \widehat{\xi}_i, & \widehat{\xi}_i \ge 0 \end{cases}$$

The optimization objective is

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i (\xi_i + \widehat{\xi}_i)$$

Multiclass Classification With any native binary classifier (e.g., AdaBoost, logistic regression, SVM), options for $C > 2$ include:

- one vs all: build C classifiers, then reconcile all classifiers.

- one vs one: build $\binom{C}{2}$ classifiers, need to reconcile and then we can build "tournament" (e.g. trees). More problematic because of inconsistencies.
- Extend to multiclass by modifying the machinery (e.g. softmax from logistic regression, and multiclass SVM)

Multiclass SVM For K classes, learn $\mathbf{w}_k$ for $k = 1, \cdots, K$,

$$\widehat{y}(\mathbf{x}; \mathbf{w}_1, \cdots, \mathbf{w}_K) = \mathrm{argmax}_k \mathbf{w}_k \cdot \mathbf{x}$$

we can the stack $\mathbf{w}_k$'s into rows of $\mathbf{W}$. Surrogate loss on $(\mathbf{x}, y)$:

$$\max_r \left\{ \mathbf{w}_r^T \mathbf{x} + 1 - \delta_{r,y} \right\} - \mathbf{w}_y^T \mathbf{x}$$

where $\delta_{y,k} = 1$ iff $y = k$, otherwise 0; and the surrogate loss is an upper bound on $0/1$ loss. The optimization objective is

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{W}\|_F^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & \mathbf{w}_y \cdot \mathbf{x}_i - \mathbf{w}_r \cdot \mathbf{x}_i \geq 1 - \delta_{r,y} - \xi_i \end{aligned}$$

where $\|\mathbf{W}\|_F$ is the Frobenius norm. For $k = y$, $0 \geq 0 - \xi_i$ thus $\xi_i \geq 0$. For $k \neq y$, $(\mathbf{w}_y - \mathbf{w}_k)^T \mathbf{x}_i \geq 1 - \xi_i$, so $\xi_i \geq 1 - (\mathbf{w}_y - \mathbf{w}_k)^T \mathbf{x}_i$, which is equivalent to the hinge loss.

Introducing Lagrange multipliers λ_i, r, the optimization objective is

$$\min_{\mathbf{W}, \boldsymbol{\xi}} \max_{\boldsymbol{\lambda}} \frac{\lambda}{2} \sum_{k=1}^{K} \|\mathbf{w}_c\|_2^2 + \sum_{i=1}^{N} \xi_i + \sum_{i=1}^{N} \sum_{k=1}^{K} \lambda_{i,k} \left[(\mathbf{w}_k - \mathbf{w}_{y_i})^T \mathbf{x}_i + 1 - \delta_{k,y_i} - \xi_i \right]$$

subject to $\lambda_{i,k} \geq 0, \xi_i \geq 0$.

7 Multi-armed Bandit (Optional)

7.1 Multi-armed Bandit Introduction

Multi-armed Bandit (MAB) Multi-armed Bandit is a basic model to study sequential decision making under uncertainty. (***Sequential decision making***: interact with the environment and learn from the consequence of actions.) MABs are simplified models for reinforcement learning (RL).

Problem Setup A fixed set of k actions ("arms"), and for each arm $i \in \{1, \cdots, k\}$ is associated with a reward distribution with mean μ_i. In each round, $t = 1, \cdots, n$, the learner chooses an arm $a_t \in \{1, \cdots, k\}$ and observe reward r_t for the chosen arm. Bandit feedback setting: the rewards for unchosen arms are not observed. The goal is to maximize the total rewards.

Exploration-exploitation Trade-off The problem of whether to acquire new information ("exploration") or make the best decision based ib available information ("exploitation"). Exploration-exploitation trade-off is the fundamental trade-off in decision making with limited information.

Bandit Variations The variations includes:

1. Reward:
   (a) i.i.d. rewards: the reward for each arm is drawn independently from a fixed distribution.
   (b) Adversarial rewards: rewards are chosen by an adversary
   (c) Constrained adversary: rewards are chosen by an adversary with known constraints
   (d) Stochastic rewards (beyond iid): reward of each arm evolves over time as a stochastic process
2. Contexts: in each round, there might be a context observable before the decision is made.
3. Combinatorial bandits: multi-armed bandit setting but in each round, we play a combinatorial set S of arms and receive the reward of the set.
4. Top-m arm identification: the goal is to dins the top-m arms out of k arms using as few samples as possible. (applications: medical trails, crowd-sourcing)
5. Other variations: structural rewards, global constraints, complex action space, complex outcomes/feedback, delayed feedback.

7.2 Multi-armed Bandit Algorithms

Formal Setup

- Arm set: $\mathcal{A} = \{1, \cdots, k\}$ and $|\mathcal{A}| = k$.
- At every round $t = 1, \cdots, n$:
  - learner chooses an arm $a_t \in \mathcal{A}$
  - data point $z_t = \{z_{t,1}, \cdots, z_{t,k}\} \in [0,1]^k$ is sampled independently from an unknown distribution with unknown means $(\mu_1, \cdots, \mu_k) \in [0,1]^k$
  - learner observes reward z_{t,a_t} (but not other rewards, bandit feedback)
- The goal is to minimize the ***pseudo-regret*** R_n defined as
$$R_n = \sum_{t=1}^{n} \mathbb{E}[z_{t,a^*} - z_{t,a_t}] = \sum_{t=1}^{n} (\mu_{a^*} - \mu_{a_t}) = n\mu_{a^*} - \sum_{t=1}^{n} \mu_{a_t}$$
where $a^* = \text{argmax}_{a \in \mathcal{A}} \mu_a$.
- a_tm the decision at round t, is a function of $a_1, \cdots, a_{t-1}$ and $z_{1,a_1}, \cdots, z_{t-1,a_{t-1}}$.
- Note: learning occurs when algorithm achieves sub-linear growth in n, namely $\mathbb{E}[R_n]/n \to 0$.

Multi-armed Bandit Problem Let $N_{t,a} := \sum_{i=1}^{t} 1_{\{a_i = a\}}$ denotes the times arm a is pulled up to time t. The suboptimality of arm a is $\Delta_a := \mu_{a^*} - \mu_a$.

Lemma 1: $R_n = \sum_{a=1}^{k} \Delta_a N_{n,a}$.

Proof: $R_n = n\mu_{a^*} - \sum_{t=1}^{n} \mu_{a_t} = \sum_{a=1}^{k} \mu_{a^*} N_{n,a} - \sum_{a=1}^{k} \mu_a N_{n,a} = \sum_{a=1}^{k} \Delta_a N_{n,a}$. ■

Two attempts: (1) Explore-then-commit: explore all arms for m times and then commit to the arm with the highest sample mean. Exploration-exploitation trade off controlled by m. (2) ϵ-Greedy: keep exploration on with probability ϵ. Exploration-exploitation trade off controlled by ϵ. Both approaches suffer from linear pseudo regret.

Upper Confidence Bound (UCB) Algorithm The idea is to let the exploration depends on the confidence of means estimates. The exploration-exploitation trade-offs are controlled by the confidence parameters $\{\beta_t\}$ to be tuned later. After the initial exploration, follow the following algorithm

Algorithm Upper Confidence Bound (UCB)

```
1: for t = k+1, ···, n do
2:     Set $a_t = \text{argmax}_a \widehat{\mu}_{t,a} + \sqrt{\beta_t / N_{t,a}}$
3: end for
```

Lemma 2: In UCB, set $\beta_t = \frac{1}{2}\log[4(n-k)/\delta]$ for any $\delta > 0$. Then the expected pseudo-regret is
$$\mathbb{E}[R_n] \le 2\log\frac{4(n-k)}{\delta} \sum_{a \ne a^*} \frac{1}{\Delta_a} + n\delta \sum_{a=1}^{k} \Delta_a.$$

Proof: *Step 1*: Construct a confidence region around the sample mean. $M_t := \sqrt{\frac{\log[4(n-k)/\delta]}{2N_{t,a}}}$ for later convenience. By Hoeffding's inequality and union bound,
$$P\left(|\widehat{\mu}_{t,a} - \mu_a| \le M_t,\ \forall\, t \in [k+1, n]\right) \ge 1 - \delta/2.$$

Define lower confidence bound (LCB) and upper confidence bound (UCB) $L_{t,a} = \widehat{\mu}_{t,a} - M_t$ and $U_{t,a} = \widehat{\mu}_{t,a} + M_t$. For any arm a, we have $P(\mu_a \in [L_{t,a}, U_{t,a}]) \geq 1 - \delta/2$.

Step 2: show that any non-optimal action a cannot be pulled too frequently. Fix any non-optimal arm $a \nVdash$. Consider the event $E_a := \{\mu_a \in [L_{t,a}, U_{t,a}]\} \cap \{\mu_{a^*} \in [L_{t,a^*}, U_{t,a^*}]\}$, and let n_a be the largest round in which arm a is played, then we must have $U_{n_a,a} \geq U_{n_a,a^*}$. Under event E_a,

$$\mu_a \geq L_{n_a,a} = U_{n_a,a} - 2M_{n_a} \geq U_{n_a,a^*} - 2M_{n_a} \geq \mu_{a^*} - 2M_{n_a}$$

It implies that $N_{n,a} = N_{n_a,a} \leq 2\log[4(n-k)/\delta]/\Delta_a^2$. Thus, we have

$$\Delta_a E[N_{n,a}] = \Delta_a E\left[N_{n,a} 1_{\{E_a\}}\right] + \Delta_a \mathbb{E}\left[N_{n,a} 1_{\{E_a^c\}}\right] \leq \tfrac{2\log[4(n-k)/\delta]}{\Delta_a} + n\Delta_a\delta.$$

Combing via $\mathbb{E}[R_n] = \sum_{a=1}^{k} \Delta_a \mathbb{E}[N_{n,a}]$, we obtain the desired result. ■

Lemma 3: In UCB, set $\beta_t = \frac{1}{2}\log[4(n-k)/\delta]$ for any $\delta > 0$, then

$$\mathbb{E}[R_n] \leq x\sqrt{2nk\log[4n(n-k)]} + k = \mathcal{O}(\sqrt{nk\log n})$$

The UCB algorithms we considered so far yields the regret bound $\mathcal{O}(\sqrt{nk\log n})$. We can construct minimax lower bounds to know whether this bound is improvable.

8 Reinforcement Learning (Optional)

8.1 Reinforcement Learning Introduction

Reinforcement Learning Problems involve learning from interacting with an environment, which provides reward signals for each action. Key features include:

1. The learner is not told what actions to take, instead it needs to find out by *trail-and-error* search.
2. The environment is stochastic.
3. The reward may be delayed.
4. Learner need to balance the need to explore its environment and exploit its current knowledge.

Agent Environment Interaction Protocol Agent and environment interact at discrete time steps $t = 0, 1, \cdots$, agent observed state $s_t \in \mathcal{S}$, chooses action $a_t \in \mathcal{A}$, and gets rewards $r_t \in \mathcal{R}$. The objective is to get as much total reward as possible.

Type of tasks: (1) Episodic Task: interaction breaks naturally into episodes. We usually use the total rewards $G_t = r_{t+1} + \cdots + r_T$ where T is a final step at which a terminal state is reached. (2) Continuing Tasks: interaction does not have natural episodes. In this tasks class, we use the discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ where $\gamma \in [0, 1]$ is the discount rate.

Policy Execute actions in environment, observe rewards, and learn policy (strategy, way of behaving) $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$,

$$\pi(a|s) = \Pr(a_t = a \,|\, s_t = s).$$

The policy can be deterministic, $\pi : \mathcal{S} \mapsto \mathcal{A}$, which $\pi(s) = a$ giving action chosen in the state s. Suppose the sequence of rewards after step t is $r_{t+1}, r_{t+2}, \cdots$. We want to compute or maximize the expected return $\mathbb{E}[G_t]$ on each step t.

Markov Decision Process A mathematical formulation of RL problems. ***Markov Property***: current state completely characterizes the state of the world and next state depends only on the current state and the action. Defined by tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$ where P is the transition probability (i.e., probability of next state given current state).

At time $t = 0$, environment sample initial state $s_0 \sim p(s_0)$. From $t = 0$ until done,

- agent selects action a_t,
- environment samples reward $r_t \sim R(s_t, a_t)$,
- environment samples next state $s_{t+1} \sim P(\cdot \,|\, s_t, a_t)$,
- agent receives reward r_t and observed next state s_{t+1}.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps a state into a distribution over action space. Discounted return: $G_t := \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. Objective: find an optimal policy π^* that maximizes G_t on each step t thus maximizes the expected cumulative reward.

Value Function and Bellman Equation ***State-value function***

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\, s_t = s \right].$$

is the expected cumulative reward when following policy π starting from state s. ***Q-value function***

$$Q^\pi(s,a) := \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,|\, (s_t,a_t)=(s,a)\right].$$

is the expected cumulative reward when following policy π, starting from state-action pair (s,a). ***Bellman Theorem*** and Optimality Bellman equation:

$$\begin{aligned} Q^\pi(s,a) &= \sum_{r,s'}\sum_{a'}\pi(a' \,|\, s')p(r,s' \,|\, s,a)\,(r+\gamma Q^\pi(s',a')) \\ Q^*(s,a) &= \mathbb{E}_{r\sim R(s,a),\, s'\sim P(\cdot \,|\, s,a)}\left[r+\gamma\max_{a'\in\mathcal{A}}Q^*(s',a')\right] \end{aligned}$$

The value of a state under an optimal policy must equal the expected return for the best action from that state $V^*(s) = \max_a Q^*(s,a) = \max_a \sum_{r,s'}^{a} p(r,s' \,|\, s,a)(r+\gamma V^*(s'))$. Given Q^*, the optimal action-value function is $\pi^*(s) \in \mathrm{argmax}_a Q^*(s,a)$.

Dynamic Programming Approach

- Policy Evaluation: Key idea: use Bellman equation. Procedure: initialize any function Q_1, iteratively compute
$$Q_{t+1}(s,a) \leftarrow \sum_{r,s'}\sum_{a'}\pi(a' \,|\, s')p(r,s' \,|\, s,a)\,(r+\gamma Q_t(s',a'))$$
Result: Q_t converges to Q^π when $t\to\infty$.
- Policy Improvement: Given a policy π, we can greedify with respect to a given the value function Q^π, denoted π'. Namely, $\pi'(\mathrm{argmax}_a Q^\pi(s,a) \,|\, s) = 1$, then we have $V^{\pi'}(s) \geq V^\pi(s)$.
- Value Iteration: Use Bellman equation as an iterative update: $Q_{i+1}(s,a) \leftarrow \mathbb{E}\left[r+\gamma\max_{a'}Q_i(s',a')\right]$. Q_i converges to Q^* when $i\to\infty$.

Drawback: DP require the full knowledge about MDP, but we don't know MDP and must learn from experience in learning setting.

8.2 Reinforcement Learning Algorithm

8.2.1 Action-Value Methods

The core idea is to learn the value of each action and pick the maximum.

Temporal Difference (TD) Simple Monte Carlo: $V(s_t) \mapsto V(s_t) + \alpha(G_t - V(s_t))$.

TD (Temporal-difference) learning for policy evaluation: A bootstrapping method that does not wait until the end of an episode to make update. Given an experience (s_t, a_t, r_t, s_{t+1}):

$$V(s_t) \mapsto V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)).$$

Learning for control tasks: learn an optimal policy from experience (need exploration). Type of explorations:

- Randomization: add noise to the greedy policy (e.g. ϵ-greedy)
- Optimism in face of uncertainty principle: prefer actions with high estimated value and high uncertainty (e.g., UCB)
- Probability Matching: select actions based according to their probability of being optimal (e.g., Thompson)

SARSA: On-policy TD Control Given current action-value function estimate $Q(s, a)$. Choose a from s using current Q. Take a and observe r, s'. Choose a' from s' using current Q. TD update:

$$Q(s, a) \mapsto Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

SARSA always learn the action-value function of the current policy, and it always act near-greedily w.r.t. the current action-value function estimates.

Q-Learning: Off-policy TD Control Given current action-value function estimate $Q(s, a)$. Choose a from s using current Q. Take a and observe r, s'. TD Update: ?

$$Q(s, a) \mapsto Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

In tabular case, converges to Q^*.

Deep Q-Network (DQN) and Double Q-Learning DQN intuition: changing the value of one action will change the value of other actions and similar states $\rightarrow$ network can end up chasing its own tail because of bootstrapping.

Q-learning suffers from larger overestimation caused by maximization bias (since max is taken), so we introduce Double Q-Learning. Train two action-value functions Q_1, Q_2 independently. At each time step, randomly pick Q_1, Q_2 and do Q-learning in it. If updating Q_1, use Q_2 for the value of the next state and vice versa,

$$Q_1(s, a) \mapsto Q_1(s, a) + \alpha[r + \gamma Q_2(s', a) - Q_1(s, a)]$$

Action selection is ϵ-greedy w.r.t the sum of Q_1 and Q_2.

8.2.2 Policy Gradient Methods

This approach learn the parameters of a stochastic policy and update by gradient ascent in performance.

The policy is usually simpler to approximate than value functions, and policy is usually stochastic, enabling smoother change in policy and avoid a search on every step.

General Policy-Gradient Directly parameterize policy $\pi_\theta(a \mid s)$. Objective function:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi_\theta\right]$$

Gradient ascent $\theta \to \theta + \eta \nabla_\theta J(\theta)$.

Note that $\nabla_\theta \pi_\theta(a \mid s) = \pi_\theta(a \mid s) \nabla \theta \log \pi_\theta(a \mid s)$, we have

$$\nabla_\theta = \mathbb{E}\left[\left(\sum_{t=0}^{T} \gamma^t r_t\right)\left(\nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a \mid s)\right)\right]$$

Stochastic gradient descent: $\theta \mapsto \theta - \alpha \nabla_\theta \widehat{J}(\theta)$.

Actor-Critic Monte-Carlo policy gradient has high variance, we can use a critic to estimate the action-value function: $Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$. Critic updates action-value function parameter w; actor updates policy parameters θ, in a direction suggested by the critic. Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_\theta(s,a) Q_w(s,a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$$

We can subtract a ***baseline function*** $B(s)$ to reduce variance, without changing the expectation $\mathbb{E}_{\theta_\pi}[\nabla_\theta \pi_\theta(s,a) B(s)] = 0$. A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$. We can rewrite policy gradient using the ***advantage function***

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]$$

9 Generative Models, Mixture Models, EM

9.1 Generative Models

Summary

- Generative Model

Review: Bayes Risk The Bayes classifier is given by

$$\begin{aligned} h^*(\mathbf{x}) &= \text{argmax}_c \, p(y = c \,|\, \mathbf{x}) = \text{argmax}_c \, p(\mathbf{x} \,|\, y = c) p(y = c)/p(\mathbf{x}) \\ &= \text{argmax}_c \, [\log p(\mathbf{x} \,|\, y = c) + \log p(y = c)]. \end{aligned}$$

The risk (probability of error) of Bayes classifier h^* is called the ***Bayes risk*** R^*, $R^* = 1 - \int_{\mathbf{x}} \max_c [p(\mathbf{x} \,|\, y = c)p(y = c)] \, dx$. It is the minimal achievable risk for the given $p(\mathbf{x}, y)$ with any classifier, and it measures the inherent difficulty of the classification problem.

Review: Multivariate Gaussians The multivariate Gaussians is given by

$$\mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] \tag{9.1}$$

where the covariance matrix $\boldsymbol{\Sigma}$ determines the shape of the density, and its determinant measures the spread (analogous to σ^2). $\mathcal{N}$ is the joint density of coordinates $x_1, \cdots, x_d$. The log-likelihood is given by

$$\log \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{d}{2} \log 2\pi - \frac{1}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}).$$

Therefore, the maximum likelihoods for the mean and the covariance are

$$\widehat{\boldsymbol{\mu}}_{ML} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \qquad \text{and} \qquad \widehat{\boldsymbol{\Sigma}}_{ML} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

Discriminant Functions For each class c, a discriminant function is constructed as

$$\delta_c(\mathbf{x}) := \log p(\mathbf{x} \,|\, y = c) + \log p(y = c) \tag{9.2}$$

such that $h^*(\mathbf{x}) = \text{argmax}_c \, \delta_c(\mathbf{x})$.

Binary Classification Case: In case of two classes $y \in \{\pm 1\}$, the Bayes classifier is

$$h^*(\mathbf{x}) = \text{argmax}_c \, \delta_c(\mathbf{x}) = \text{sign}(\delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x}))$$

and the decision boundary is given by $f(\mathbf{x}) = \delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x}) = 0$, and $f(\mathbf{x})$ is sometimes referred to as a discriminant function. With equal prior, this is equivalent to the (log)-likelihood ratio test: $h^*(\mathbf{x}) = \text{sign}[\log[p(\mathbf{x} \,|\, y = +1)/p(\mathbf{x} \,|\, y = -1)]]$.

Equal Covariance Gaussian Case: Consider the case of $p_x(\mathbf{x}) = \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$, and equal prior for all classes, i.e., $p(y = c) = 1/C$ for all c.

$$\begin{aligned}\delta_k(\mathbf{x}) = \log p(\mathbf{x} \mid y = k) &= -\log(2\pi)^{d/2} - \frac{1}{2}\log|\mathbf{\Sigma}| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \\ &\propto \text{const} - \mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k^T \\ &\propto 2\boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k^T\end{aligned}$$

where the last proportionality holds because $\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x}$ is independent of k (so it is equal for all k). Now consider two classes r, q, two class discriminant is given by

$$\begin{aligned}\delta_r - \delta_q &= (\boldsymbol{\mu}_r^T\mathbf{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_r^T) - (\boldsymbol{\mu}_q^T\mathbf{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_q^T) \\ &= 2(\boldsymbol{\mu}_r^T - \boldsymbol{\mu}_q^T)\mathbf{\Sigma}^{-1}\mathbf{x} + (-\boldsymbol{\mu}_r^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_r^T + \boldsymbol{\mu}_q^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_q^T)\end{aligned}$$

so the optimal decision boundary is *linear*, in the form of $\mathbf{w}^T\mathbf{x} + w_0$. By knowing $\boldsymbol{\mu}_{1,\cdots,C}$ and $\mathbf{\Sigma}$, we can compute the optimal $\mathbf{w}, w_0$. However, we don't always know the Gaussian.

Gaussian With Unequal Covariances: Suppose we have the covariance $\mathbf{\Sigma}_c$ for each c, we need to compute ML estimate for $\boldsymbol{\mu}_c, \mathbf{\Sigma}_c$ for each c. We get discriminants quadratic in $\mathbf{x}$:

$$\delta_c(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T\mathbf{\Sigma}_c^{-1}\mathbf{x} + \boldsymbol{\mu}_c^T\mathbf{\Sigma}_c^{-1}\mathbf{x} - \text{const}_c(\mathbf{x})$$

The decision boundary with two classes is $\delta_1 - \delta_0 = 0$, which is quadratic in $\mathbf{x}$.

Quadratic decision boundary: Second-degree curves can be any conic section, e.g., linear functions (parallel or intersecting), parabola, hyperbola, ellipse (circle).

Generative Models In generative model, we need one explicitly models $p(\mathbf{x}, y)$ (equivalently, $p(\mathbf{x} \mid y = c)$ and $p(y = c)$) to derive discriminant. Typically, the model imposes certain parametric form on the assumed distributions, and requires estimation of the parameters from data. The most popular model is Gaussian for continuous and multinomial for discrete.

9.2 Mixture Models, EM Algorithm

9.2.1 Mixture Models

Mixture Models The mixture model is motivated to solve the case where each class contains number of distinct types. We will focus on one class, suppose the class has:

- k underlying types (components),
- h_i is the identity of the component responsible for $\mathbf{x}_i$, and
- h_i is a hidden (latent) variable, which is never observed.

A mixture model is

$$p(\mathbf{x}; \boldsymbol{\pi}) = \sum_{j=1}^{k} p(h = j)\, p(\mathbf{x} \,|\, h = j)$$

$\pi_j := p(h = j)$ are the mixing probabilities. We need to parametrize the component densities $p(\mathbf{x} \,|\, h = j)$. Suppose the parameters of the j-th component are θ_j, then we can denote $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_k\}$ and write

$$p(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{j=1}^{k} \pi_j \cdot p(\mathbf{x}; \boldsymbol{\theta}_j). \tag{9.3}$$

Any valid setting of $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, subject to $\sum_{j=1}^{k} \pi_j = 1$, produces a valid pdf.

Generative Model for a Mixture The generative process with k-component mixture:

- The parameters $\boldsymbol{\theta}_j$ for each component j are fixed.
- Draw $h_i \sim [\pi_1, \cdots, \pi_k]$.
- Given h_i, draw $\mathbf{x}_i \sim p(\mathbf{x} \,|\, h_i; \boldsymbol{\theta}_{h_i})$

The entire generative model for $\mathbf{x}$ and h

$$p(\mathbf{x}, h; \boldsymbol{\theta}, \boldsymbol{\pi}) = p(h; \boldsymbol{\pi}) \cdot p(\mathbf{x} \,|\, h; \boldsymbol{\theta}_h)$$

Mixture Density Estimate Mixture of Gaussian: If the h-th component is a Gaussian, $p(\mathbf{x} \,|\, j = h) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, a Gaussian mixture model is

$$p(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{j=1}^{k} \pi_j \cdot \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j),$$

where $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \cdots, \boldsymbol{\Sigma}_k\}$, and π_j are the mixing probabilities.

Likelihood of a mixture model: estimate set of parameters that maximize likelihood given the observed data. The log-likelihood is given by

$$\log p(\mathbf{X}; \boldsymbol{\pi}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \sum_{j=1}^{k} \pi_j \, \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

There is no closed-form solution because of the sum inside log (we need to take into account all possible components that could generate $\mathbf{x}_i$).

Mixture Density Estimate - Known Types: Assume we know the type of each data point. Let $\boldsymbol{z}_i = [z_{i1}, \cdots, z_{ik}]$ where $z_{ij} = 1_{\{h_i = j\}}$ be the binary indicator variables, and let the count of example from j-th component to be denoted by $N_j := \sum_{i=1}^{N} z_{ij}$. If we know $\mathbf{z}_i$, the ML estimates of the Gaussian components are

$$\widehat{\pi}_j = \frac{N_j}{N} \tag{9.4a}$$

$$\widehat{\boldsymbol{\mu}}_j = \frac{1}{N_j}\sum_{i=1}^{N} z_{ij}\mathbf{x}_i \tag{9.4b}$$

$$\widehat{\boldsymbol{\Sigma}}_j = \frac{1}{N_j}\sum_{i=1}^{N} = z_{ij}(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_j)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_j)^T \tag{9.4c}$$

Expected Likelihood Credit Assignment: Suppose we don't know the types but know the component parameters, $\boldsymbol{\theta}$ and mixing probabilities $\boldsymbol{\pi}$. The posterior of each label using Bayes rule:

$$\gamma_{ij} = \widehat{p}(j = h \mid \mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi}) = \frac{\pi_j \cdot p(\mathbf{x}; \boldsymbol{\theta}_j, \boldsymbol{\pi}_j)}{\sum_{l=1}^{k} \pi_l \cdot p(\mathbf{x}; \boldsymbol{\theta}_l, \boldsymbol{\pi}_l)}$$

We call γ_{ij} the responsibility of the j-th component for $\mathbf{x}$.

The "complete data" likelihood (when $\boldsymbol{z}$ are known) and the log-likelihood

$$p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\pi}, \boldsymbol{\theta}) \propto \prod_{i=1}^{N}\prod_{j=1}^{k}\left[\pi_j \cdot \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right]^{z_{ij}}$$

$$\log p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\pi}, \boldsymbol{\theta}) = \text{const} + \sum_{i=1}^{N}\sum_{j=1}^{k} z_{ij}\left[\log \pi_j + \log \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right]$$

We are unable to compute the log-likelihood, but we can be take the expectation w.r.t. the posterior of $\mathbf{z}$, which is γ_{ij}, thus the ***expected likelihood*** of the data is given by

$$J(\mathbf{X}, \boldsymbol{\pi}, \boldsymbol{\theta}) = \mathbb{E}_{z_{ij}\sim\gamma_{ij}}\left[\log p(\mathbf{x}_i, z_{ij}; \boldsymbol{\pi}, \boldsymbol{\theta})\right] = \sum_{i=1}^{N}\sum_{j=1}^{k} \gamma_{ij}\left[\log \pi_j + \log \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right]$$

since $\mathbb{E}_{z_{ij}\sim\gamma_{ij}}[z_{ij}] = \gamma_{ij}$. Note that the constraint $\sum_j \pi_j = 1$ must hold.

Expectation Maximization Derivation: The Lagrangian is $\mathcal{L}(\mathbf{X}, \boldsymbol{\pi}, \boldsymbol{\theta}) = J(\mathbf{X}, \boldsymbol{\pi}, \boldsymbol{\theta}) + \lambda\left(\sum_{j=1}^{k}\pi_j - 1\right)$. Setting the partial derivative to zero is gives

$$0 = \frac{\partial \mathcal{L}}{\partial \pi_j} = \sum_{i=1}^{N}\frac{\gamma_{ij}}{\pi_j} + \lambda \quad \Rightarrow \quad \pi_j = -\frac{1}{\lambda}\sum_{i=1}^{N}\gamma_{ij}$$

Note that $\sum_j \pi_j = -(1/\lambda)\sum_{i=1}^{N}\sum_{j=1}^{k}\gamma_{ij} = -N/\lambda = 1$, so $\lambda = -N$. Therefore,

$$\widehat{\pi}_j = \frac{1}{N}\sum_{i=1}^{N}\gamma_{ij}.$$

Compute partial derivative of $\log \mathcal{N}(\mathbf{x}; \boldsymbol{\theta})$ is $(\partial \log p)/(\partial \mu_j) = \boldsymbol{\Sigma}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)$, and setting the partial derivative of J to zero yields

$$0 = \frac{\partial J}{\partial \mu_j} = \sum_{i=1}^{N}\gamma_{ij}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_j) = \boldsymbol{\Sigma}_j^{-1}\left(\sum_{i=1}^{N}\gamma_{ij}\mathbf{x}_i - \sum_{i=1}^{N}\gamma_{ij}\mu_j\right), \text{ so}$$

$$\widehat{\mu}_j = \frac{1}{\sum_{l=1}^{N}\gamma_{lj}}\sum_{i=1}^{N}\gamma_{ij}\mathbf{x}_i.$$

The derivation of $\widehat{\boldsymbol{\Sigma}}$ is omitted.

Expectation Maximization: Maximizing the expected likelihood (by setting derivatives to zero and using Lagrange multipliers), we obtain

$$\widehat{\pi}_j = \frac{1}{N}\sum_{i=1}^{N}\gamma_{ij} \tag{9.5a}$$

$$\widehat{\boldsymbol{\mu}}_j = \frac{1}{\sum_{l=1}^{N}\gamma_{lj}}\sum_{i=1}^{N}\gamma_{ij}\mathbf{x}_i \tag{9.5b}$$

$$\widehat{\boldsymbol{\Sigma}}_j = \frac{1}{\sum_{l=1}^{N}\gamma_{lj}}\sum_{i=1}^{N}\gamma_{ij}(\mathbf{x}_i-\widehat{\boldsymbol{\mu}}_j)(\mathbf{x}_i-\widehat{\boldsymbol{\mu}}_j)^T \tag{9.5c}$$

Recap We want to find the parameters and indicators (assignments).

- If we know the indicators but not the parameters, we can do ML estimation of the parameters.
- Conversely, if we know the parameters but not the indicators, we can compute the posterior of indicators (γ), and the we can estimate parameters that maximize the expected likelihood.

However, we know neither parameters not the indicators, so we introduce EM algorithm.

9.2.2 The EM Algorithm

EM Algorithm

1. Starting with a guess of $\boldsymbol{\pi}, \boldsymbol{\theta}$. (Typically, we use a random Gaussians and $\pi_j = 1/k$).
2. Iterate until convergence:
    - E-step: compute values of expected assignments, i.e., calculate γ_{ij}, using current estimates of $\boldsymbol{\pi}, \boldsymbol{\theta}$,
    $$\gamma_{ij} = \frac{\pi_j\, p(\mathbf{x}_i; \boldsymbol{\theta}_j)}{\sum_{l=1}^{k}\pi_l\, p(\mathbf{x}_i; \boldsymbol{\theta}_l)}$$
    - M-step: maximize the expected likelihood, under current γ_{ij},
    $$\boldsymbol{\pi}, \boldsymbol{\theta} \mapsto \operatorname*{argmax}_{\boldsymbol{\pi},\boldsymbol{\theta}} \sum_{i=1}^{N}\sum_{j=1}^{k}\gamma_{ij}\left[\log \pi_j + \log \mathcal{N}(\mathbf{x}_i; \boldsymbol{\theta})\right]$$

EM Algorithm in General

1. Observe data $\mathbf{X}$, hidden variables $\mathbf{Z}$.
2. Initialize $\boldsymbol{\theta}$, and iterate until convergence:
    - E-step: compute the expected complete data log-likelihood as a function of $\boldsymbol{\theta}$,
    $$Q(\boldsymbol{\theta}'; \boldsymbol{\theta}) = \mathbb{E}_{p(Z|X,\boldsymbol{\theta})}\left[\log p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\theta}') \,|\, \mathbf{X}, \boldsymbol{\theta}\right]$$
    - M-step: compute $\boldsymbol{\theta}' \mapsto \operatorname{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}'; \boldsymbol{\theta})$.

Regularized EM The problem of EM: we can be unlucky with the initial guess. To regularize EM we can impose a prior on $\boldsymbol{\theta}$. Instead of maximizing the likelihood in the M-step, maximize the posterior:

$$\boldsymbol{\theta} \mapsto \underset{\boldsymbol{\theta}'}{\operatorname{argmax}} \left\{ \mathbb{E}_{p(Z|X;\boldsymbol{\theta}')} \left[\log p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\theta}') \mid \mathbf{X}; \boldsymbol{\theta} \right] + \log p(\boldsymbol{\theta}') \right\}$$

A common prior on a covariance matrix: the Wishart distribution

$$p(\boldsymbol{\Sigma}; \mathbf{S}, \boldsymbol{n}) \propto \frac{1}{|\boldsymbol{\Sigma}|^{n/2}} \exp\left(-\frac{1}{2} \mathrm{Tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}) \right)$$

(intuition: $\mathbf{S}$ is the covariance of n “hallucinated” observation)

Overfitting Selecting a large k will likely to overfits the data. In order to reduce overfitting, we can (1) validate on a held-out data set, or (2) penalize model complexity directly (e.g., Bayesian Information Criterion (BIC)). For a model class $\mathcal{M}$ with parameters $\boldsymbol{\theta}_{\mathcal{M}}$, we find ML (or MAP) estimates of the parameters on $\mathbf{X}$:

$$L^*(\mathcal{M}) := \max_{\boldsymbol{\theta}_{\mathcal{M}}} \log p(\mathbf{X} \mid \mathcal{M}; \boldsymbol{\theta}_{\mathcal{M}}).$$

The BIC score for the model $\mathcal{M}$ is

$$\mathrm{BIC}(\mathcal{M}) = L^*(\mathcal{M}) - \frac{1}{2} |\boldsymbol{\theta}_{\mathcal{M}}| \log N$$

9.2.3 Mixture Model for Regression

Mixture of Experts Model The distribution of y is a *conditional mixture* model:

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = \sum_{j=1}^{k} p(j \mid \mathbf{x})\, p(y \mid \mathbf{x}; \theta_j)$$

A ***gating network*** specifies the conditional distribution $p(j \mid \mathbf{x}; \eta)$ (i.e., determining which expert should be responsible); a common choice is softmax $\boldsymbol{\eta} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$. Responsibilities (under softmax):

$$\gamma_{ij} = p(j \mid \mathbf{x}_i, y_i; \boldsymbol{\theta}, \boldsymbol{\eta}) = \frac{p(j \mid \mathbf{x}_i; \boldsymbol{\eta})\, p(y_i \mid \mathbf{x}_i; \theta_j)}{\sum_{l=1}^{k} p(l \mid \mathbf{x}_i; \boldsymbol{\eta})\, p(y_i \mid \mathbf{x}_i; \theta_j)}$$

EM for Mixtures of Experts

- Initialize: random $\theta_j, \sigma_j^2, \boldsymbol{\eta}$.
- E-step: compute responsibilities $\gamma_{ij} = p(j \mid \mathbf{x}_i, y_i; \boldsymbol{\theta}, \boldsymbol{\eta})$
- M-step: for each expert j, estimate

  $$\theta_j \mapsto \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \gamma_{ij} \log p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta}),$$

  and then estimate the gating network separately

  $$\boldsymbol{\eta} \mapsto \operatorname{argmax}_{\boldsymbol{\eta}} \sum_{i=1}^{N} \sum_{j=1}^{k} \gamma_{ij} \log p(j \mid \mathbf{x}_i; \boldsymbol{\eta}).$$

10 Neural Networks, Deep Learning

10.1 Perceptron, Neural Networks

Summary

- Neural Network
- Back-propagation

10.1.1 Neural Networks

Perceptron Mistake-driven algorithm: update weights only when making a mistake on the example, i.e.,

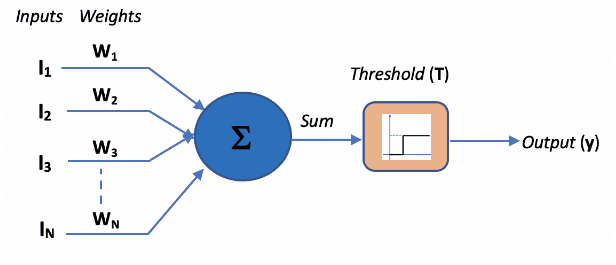
$$\mathbf{w} \mapsto \mathbf{w} + y_i \mathbf{x}_i \quad \text{and} \quad w_0 \mapsto w_0 + y_i$$

The loss is ***perceptron loss***:

$$l(\mathbf{x}_i, y_i) = \begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) > 0, \\ y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

Assume data are linearly separable, let $\mathbf{w}, w_0$ be a linear separator where $\|\mathbf{w}\| = 1$, and the margin be γ. Theorem (Novikoff, 1962): perceptron will converge after $O[(\max \|\mathbf{x}_i\|)^2/\gamma^2]$.

Neural Networks McCullough-Pitts Model: inputs ($\phi_j(\mathbf{x})$, output from the previous layer) $\rightarrow$ weights (w_{jt}) $\rightarrow$ adder (a_t) $\rightarrow$ activation function (h) $\rightarrow$ output (z_t).



Feed-forward network: Consider two-layer network, we fit the model by those features $\widehat{y} = f(\mathbf{w}^{(2)} \cdot \phi(x))$ (f is logistic regression or linear regression), and the key idea is to learn parametric features $\phi_j(\mathbf{x}) = h(\mathbf{w}_j^{(1)} \cdot \mathbf{x} + \mathbf{w}_{0j}^{(1)})$ for some functions h. Feed-forward networks: feed-forward operation from input $\mathbf{x}$ to output $\widehat{y}$ is given by

$$\widehat{y}(\mathbf{x}; \mathbf{w}) = f\left[\sum_{j=1}^{m} w_j^{(2)} h\left(\sum_{i=1}^{d} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right) + w_0^{(2)}\right].$$

Then common choice for the activation functions are linear, logistic, tanh, and threshold.

10.1.2 Network Architectures

Training the Network The error of the network on a training set is $L(\mathbf{X}; \mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2}(y_i - \widehat{y}(\mathbf{x}_i; \mathbf{w}))^2$, and there is no closed-form solution in general, so we need to resort to gradient descent. The derivative on a single example is

$$\frac{\partial L(\mathbf{x}_i)}{\partial w_j} = (\widehat{y}_i - y_i) x_{ij}.$$

Back-propagation Idea of back-propagation: apply chain rule of derivation to $\partial L(\widehat{y}, y)/\partial \mathbf{w}_j^{(k)}$.

General unit activation: $z_t = h(\sum_j w_{jt} z_j)$. Forward-propagation - calculate for each unit $a_t = \sum w_{jt} z_j$ where z is the output from the previous layer. The loss L depends on w_{jt} only through a_t, let $\delta_t := \partial L/\partial a_t$ (note that $\delta_t = \widehat{y} - y$ by $L = (\widehat{y} - y)^2/2$), then

$$\frac{\partial L}{\partial w_{jt}} = \frac{\partial L}{\partial a_t} \frac{\partial a_t}{\partial w_{jt}} = \delta_t z_j$$

Suppose hidden unit $z_t = h(a_t)$ sends input to units S. Since $a_s = \sum_{j:j \to s} w_{js} h(a_j)$, δ_t is given by

$$\delta_t = \sum_{s \in S} \frac{\partial L}{\partial a_s} \frac{\partial a_s}{\partial a_t} = h'(a_t) \sum_{s \in S} w_{ts} \delta_s \tag{10.1}$$

That is, the loss of a node t depends on the loss of all nodes that t feeds into. Recursively from the output layer back-propagating, we obtain δ_t for all nodes, and we can update the weights by using gradient descent.

Classification Networks For multi-class output, the final layer consists of K units (number of classes)

$$z_{T,c} = \mathbf{w}_{t,c}^T \mathbf{z}_{T-1} \quad \text{i.e.,} \quad \mathbf{z}_T = \mathbf{W}_T \mathbf{z}_{T-1}$$

The features (previous layer) are shared, but used with different weights. Loss computed on $\mathbf{z}_T$ and used in back-propagation.

10.1.3 Learning DNN

Model Complexity of MLP Traditionally, two form of regularization in neural networks: (1) weight decay

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{i=1}^{N} L(y_i, \widehat{y}(\mathbf{x}_i; \mathbf{W})) + \frac{\lambda}{2} \|\mathbf{W}\|^2 \right\}$$

and (2) early stopping (when the validation loss starts increasing) to prevent overfitting.

Learning Rate and Momentum *Learning Rate*: Objective: $J(\mathbf{w}) = L(y_y, \widehat{y}(\mathbf{x}_t; \mathbf{w})) + R(\mathbf{w})$, SGD update: $\mathbf{w}_T = \mathbf{w}_{t-1} + \Delta \mathbf{w}_t$ where $\Delta \mathbf{w}_t = -\eta_t \nabla_{\mathbf{w}} J$. Theoretical suggested schedule for $\eta_t = \eta_0/(1 + \gamma t)$, but in in practice most common schedule: constant η_t, occasionally dropping when training loss plateaus.

Momentum: Vanilla SGD update $\Delta \mathbf{w}_t = -\eta_n \nabla_{\mathbf{w}} J$, with momentum, $\Delta \mathbf{w}_t = \mu \Delta \mathbf{w}_{t-1} - \eta_n \nabla_{\mathbf{w}} J$ (typically $\mu = 0.9$).

RELU Traditional activation function has the problem of vanishing gradient due to saturating non-linearity. Rectified linear unit action (RELU) $\max(0, a)$. Sometimes an order of magnitude speed-up.

Dropout Problem: fully connected units near the top of the network suffer from co-adaptation. Idea of dropout: during each iteration of training, randomly remove a fraction of units with a probability p. During the test time, we multiply activation of unit by p. Drop out is equivalent to L_2 regularization, assuming the data is normalized. It is often important in deep networks.

11 Course Summary