Lecture 10: Soft-Margin SVM, Lagrangian Duality

Soft-Margin SVM

- One of the three approach to fit non-linearly separable data
    1. Transform the data (kernel)
    2. Relex the constraints (Soft-Margin)
    3. Combination of (1) and (2)
- Relax constraints to allow points to be:
    - **Inside the margin**
    - Or on the **wrong side of the boundary**
- Penalise boundaries by the **extent of "violation"** (distance from margin to wrong points)

Hinge loss: soft-margin SVM loss

- **Hard-margin SVM loss:**
    - $l_\infty = 0$ if prediction correct
    - $l_\infty = \infty$ if prediction wrong
- **Soft-margin SVM loss: (hinge loss)**
    - $l_h = 0$ if prediction correct
    - $l_h = 1 - y(w'x + b) = 1 - y\hat{y}$ if prediction wrong (penalty)
    - Can be written as: $l_h = max(0, 1 - y_i(w'x_i + b))$
- Compare with perceptron loss
    - $L(s, y) = max(0, -sy)$

Soft-Margin SVM Objective

- $\underset{\mathbf{w},b}{\arg\min}(\sum_{i=1}^n l_h(x_i, y, w, b) + \lambda||w||^2)$
    - Like ridge regression
- Reformulate objective:
    - Define **slack variables** as upper bound on loss
        - Allow you to relax the constraint
        - $\xi_i \geq l_h = \max(0, 1 - y_i(w'x_i + b))$
        - Non-zero means there is some violation
        - Don't like function like this in optimisation (no derivative)
    - **Then, new objective:**
        - $\underset{w,b,\xi}{\arg\min}(\frac{1}{2}||w||^2 + C\sum_{i=1}^n \xi_i)$
        - Constraints:
            - $\xi_i \geq 1 - y_i(w'x_i + b)$ for $i = 1, ..., n$
            - $\xi_i \geq 0$ for $i = 1, ..., n$
            - (Penalise based on the size of ξ_i, like having loss function in objective)
            - **ξ gets pushed down to be equal to l_h**
        - C: hyperparameter (have to tune by gridSearch)

Two variations of SVM

- **Hard-margin SVM objective:**
  - $\arg\min_{w,b} \frac{1}{2}||w||^2$
  - s.t. $y_i(w'x_i + b) \geq 1$ for $i = 1, ..., n$
- **Soft-margin SVM objective:**
  - $\arg\min_{w,b} \frac{1}{2}||w||^2$
  - s.t. $y_i(w'x_i + b) \geq 1 - \xi_i$ for $i = 1, ..., n$ and $\xi_i \geq 0$ for $i = 1, ..., n$
  - The constraints are **relaxed** by allowing violation by ξ_i

Constraint optimisation

- **Canonical form:**
  - minimise $f(x)$
  - s.t. $g_i(x) \leq 0, i = 1, ..., n$
  - and $h_j(x) = 0, j = 1, ..., m$
- Training SVM is also a constrained optimisation problem
- Method of **Lagrange multipliers**
  - Transform to unconstrained optimisation
  - (Or) Transform **primal** program to a related **dual** program
    - Analyze necessary & sufficient conditions for solutions of both program

Lagrangian and duality

- **Dual** objective function:
  - $L(x, \lambda, v) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{j=1}^{m} v_j h_j(x)$
  - Primal constraints became penalties
  - Called **Lagrangian** function
  - New λ and v are called the **Lagrange multipliers** or **dual variables**
- Primal program: $\min_x \max_{\lambda \geq 0} L(x, \lambda, v)$
- Dual program: $\max_{\lambda \geq 0, v} \min_x L(x, \lambda, v)$
  - May be easier to solve, advantageous
- Duality
  - Weak duality: dual optimum $\leq$ primal optimum
  - For convex problem, we have strong duality: optima coincide (same optima for primal and dual)
    - Including SVM

Dual program for hard-margin SVM

- Minimise Lagrangian w.r.t to primal variables <=> maximise w.r.t dual variables yields the **dual program:**
  - $\arg\max_{\lambda} \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i' x_j$
  - s.t. $\lambda_i \geq 0$ and $\sum_{i}^{n} \lambda_i y_i = 0$
- According to strong duality, solve dual <=> solve primal
- Complexity of solution:
  - $O(n^3)$ instead of $O(d^3)$
- Program depends on dot products of data only -> kenel

Making predictions with dual solution

- **Recovering primal variables**
  - From stationarity: get w_j^*
    - $w_j^* = \sum_{i=1}^n \lambda_i y_i (x_i)_j = 0$
  - From dual solution (complementary slackness): (get b^*)
    - $y_j(b^* + \sum_{i=1}^n \lambda_i^* y_i x_i' x_j) = 1$
    - For any example j with $\lambda_i^* > 0$ **(support vectors)**
- Make predictions (testing)
  - Classify new instance x based on sign of
    - $s = b^* + \sum_{i=1}^n \lambda_i^* y_i x_i' x$
    - ($s = w'x + b$)

Optimisation for Soft-margin SVM

- Training: find λ that solves (dual)
  - $\underset{\lambda}{\arg\max} \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j x_i' x_j$
  - s.t. $C \geq \lambda_i \geq 0$ (box constraints) and $\sum_i^n \lambda_i y_i = 0$
  - Where C is a box constraints **(only difference between soft and hard SVM)**
    - Vector λ is inside a box of side length C
    - Big C: penalise more training data, let training data has more influence
    - Small C: don't care about training data, want big margins
- Make predictions: (same as hard margin)
  - Classify new instance x based on sign of
    - $s = b^* + \sum_{i=1}^n \lambda_i^* y_i x_i' x$

Complementary slackness

- One of the KKT conditions:
  - $\lambda_i^*(y_i((w^*)' x_i + b^*) - 1) = 0$
- Remember:
  - $y_i(w' x_i + b) - 1 > 0$ means that x_i is outside the margin (classified correctly)
- Points outside the margin must have $\lambda_i^* = 0$
- Points with non-zero λ^* are **support vectors**
  - $w^* = \sum_{i=1}^n \lambda_i y_i x_i$
  - Other points has no influence on w^* (orientation of hyperplane)

Training SVM

- Inefficient
- Many λs will be zero (sparsity)

Lecture 11: Kernel Methods

Kernelising the SVM

- Two ways to handle non-linear data with SVM
  1. Soft-margin SVM

2. Feature space transformation
   - Map data to a new feature space
   - Run hard-margin / soft-margin SVM in new feature space
   - Decision boundary is non-linear in original space

- Naive workflow
  1. Choose / design a linear model
  2. Choose / design a high-dimensional transformation $\phi(x)$
     - **Hoping** that after adding a lot of various features, some of them will make the daa linearly separable
  3. For each training example and each new instance, compute $\psi(x)$
- **Problem:** impractical / impossible to compute $\psi(x)$ for high / infinite-dimensional $\psi(x)$
- **Solution:** Use kernel function
  - Since both training and prediction process in SVM only depend **dot products** between data points
  - Before data transformation:
    - Training: (parameter estimation)
      - $\arg\max_{\lambda} \Sigma_{i=1}^{n} - \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\lambda_i\lambda_j y_i y_j \mathbf{x_i'x_j}$
    - Making prediction: (computing predictions)
      - $s = b^* + \Sigma_{i=1}^{n}\lambda_i^* y_i \mathbf{x_i'x}$
  - After data transformation:
    - Training:
      - $\arg\max_{\lambda} \Sigma_{i=1}^{n} - \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\lambda_i\lambda_j y_i y_j \phi(\mathbf{x_i})'\phi(\mathbf{x_j})$
    - Making predictions:
      - $s = b^* + \Sigma_{i=1}^{n}\lambda_i^* y_i \phi(\mathbf{x_i})'\phi(\mathbf{x})$

Kernel representation

- Kernel:
  - A function that can be expressed as a dot product in some feature space: $K(u, v) = \psi(u)'\psi(v)$
- For some $\psi(x)$'s, kernel is faster to compute directly than first mapping to feature space then taking dot product
  - A "shortcut" function that gives exactly the same answer $K(x_i, x_j) = k_{ij}$
- Then, SVM becomes:
  - Training:
    - $\arg\max_{\lambda} \Sigma_{i=1}^{n} - \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\lambda_i\lambda_j y_i y_j \mathbf{K}(\mathbf{x_i}, \mathbf{x_j})$
  - Making predictions:
    - $s = b^* + \Sigma_{i=1}^{n}\lambda_i^* y_i \mathbf{K}(\mathbf{x_i}, \mathbf{x})$

Approaches to non-linearity

- ANNs:
  - Elements of $u = \phi(x)$ (second layer neurons) are transformed input x
  - This ϕ has weights learned from data
- SVMs:
  - Choice of kernel K determines feature ϕ

- Don't learn ϕ weights
- But, don't even need to compute ϕ so can support very high dimensional ϕ
- Also support arbitrary data types

Modular learning

- All information about feature mapping is **concentrated within the kernel**
- In order to use a different feature mapping -> change the kernel function
- Algorithm design decouples into:
  1. Choosing a "learning method" (e.g. SVM v.s. logistic regression)
  2. Choosing feature space mapping (i.e. kernel)
- Representer theorem
  - For any training set $x_i, y_i{}_{i=1}^n$, any empirical risk function E, monotonic increasing function g, then any solution:
    - $f^* \arg\min_f E(x_1, y_1, f(x_1), ..., x_n, y_n, f(x_n)) + g(||f||)$
    - has representation for some coefficients: $f^*(x) = \Sigma_{i=1}^n \alpha_i k(x, x_i)$
  - Tells us when a learner is kernelizable
  - The dual tells us the form this linear kernel representation takes
  - (SVM is an example)

Constructing kernels

- Polynomial kernel
  - $K(u, v) = (u'v + c)^d$
  - Can add $\sqrt{c}$ as a dummy feature to u and v (dim + 1)
  - $(u'v)^d = (u_1v_1 + ... + u_mv_m)^d = \Sigma_{i=1}^l (u_1v_1)^{a_{i1}} ... (u_mv_m)^{a_{im}} = \Sigma_{i=1}^l (u_1^{a_{i1}} ... u_m^{a_{im}}) = \Sigma_{i=1}^l \phi(u)_i \phi(v)_i$
  - Feature map ϕ: $\mathbb{R}^m \to \mathbb{R}^l$, where $\phi_i(x) = (x_1^{a_{i1}}, ..., x_m^{a_{im}})$
- Identifying new kernels
  1. Method 1: Let $K_1(u, v), K_2(u, v)$ be kernels, $c > 0$ be a constant, and $f(x)$ be a real-valued function, then each of the following is also a kernel:
     - $K(u, v) = K_1(u, v) + K_2(u, v)$
     - $K(u, v) = cK_1(u, v)$
     - $K(u, v) = f(u)K_1(u, v)f(v)$
  2. Method 2: Use Mercer's theorem
     - Consider a finite sequences of objects $x_1, ..., x_n$
     - Construct $n \times n$ matrix of pairwise values $K(x_i, x_j)$
     - $K(x_i, x_j)$ is a **valid kernel** if this matrix is **positive semi-definite (PSD)**, and this holds for all possible sequence $x_1, ..., x_n$
- Remember we need $K(u, v)$ to imply a dot product in some feature space