Advanced Section #1: SVMs, logistic regression and neural networks

CS 209B: Data Science

Javier Zazo Pavlos Protopapas

javier.zazo.ruiz@gmail.com

Lecture Outline

Classifying Linear Separable Data

Classifying Linear Non-Separable Data

Introduction to convex optimization

Dual problem of the SVC

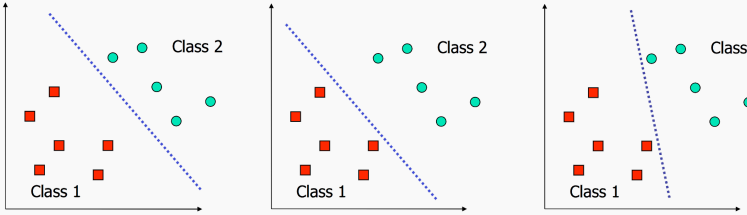
Extension to Non-linear Boundaries

Relationship to Logistic Regression

Classifying Linear Separable Data

Decision Boundaries Revisited

- **Goal:** find ***decision boundaries*** to separate classes.
- Multiple decision boundaries:
  - Linear separable data $\rightarrow$ Hard margin classifier.
  - Linear non-separable data $\rightarrow$ SVCs.
  - Non-linear boundaries $\rightarrow$ SVMs.
- Logistic regression $\rightarrow$ Kernel Logistic Regression.
- Motivation for NN.



Hyperplanes

- Hyperplane equation:

$$f(x) = \beta_0 + \beta^T x = 0.$$

- Normal vector: β.
- Halfspaces: $f(x) \lesseqgtr 0$.
- Given N pair data points:

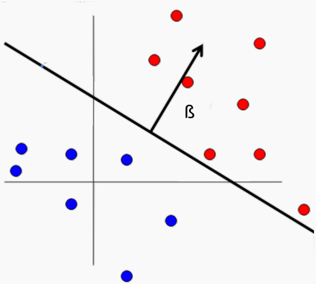
$$(x_1, y_1), \dots, (x_N, y_N)$$

- Binary classification:

$$y_i \in \{-1, 1\}$$

- Separable classes:

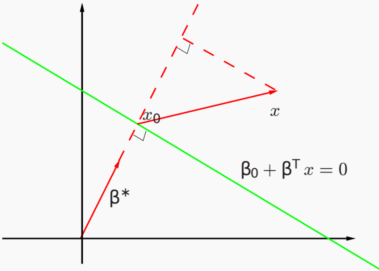
$$y_i(\beta_0 + \beta^T x_i) \geq 0$$



Maximizing Margins

- We establish a geometric principle to classify data.
- Distance between x and hyperplane:

$$D(x) = \frac{\beta_0 + \beta^T x}{\|\beta\|}.$$



- Unsigned distances: $\boxed{|D(x_i)| = y_i D(x_i) \geq M}$.
- Maximum margin classifier:

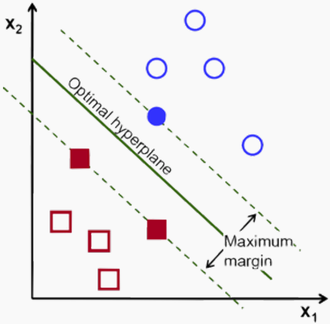
$$\begin{aligned} \max_{M,\beta_0,\beta} \quad & M \\ \text{s.t.} \quad & \frac{1}{\|\beta\|} y_i(\beta_0 + \beta^T x_i) \geq M, \quad \forall i. \end{aligned} \tag{1}$$

Maximum Margin Classifier

- We transform the previous problem into

$$\begin{aligned} \min_{\beta_0,\beta} \quad & \|\beta\| \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1, \quad \forall i. \end{aligned} \tag{2}$$

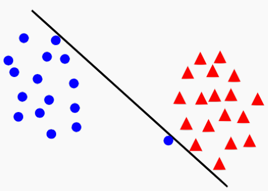
- Both problems can be solved using specific solvers.
- They become unfeasible if the data is non-separable.



Classifying Linear Non-Separable Data

The Margin/Error Trade-Off

- With every decision boundary, bias-variance trade-off.
- The maximum margin classifier:
  - Low bias but **very high variance**.
- Generalization to linear non-separable boundaries.



Support Vector Classifier (SVC)

- Classes overlap in the input space.
- Maximize the margin but relax the constraints.
- Use of slack variables $\xi = (\xi_1, \dots, \xi_N)$:

$$\begin{aligned} \min_{\beta_0, \beta, \xi_i \geq 0} \quad & \|\beta\| \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i \\ & \sum_{i=1}^{N} \xi_i \leq C, \end{aligned}$$

- C limits the amount of violation of the constraints.
  - Margin violation: points inside the margin.
  - Misclassification: points on the wrong side of the boundary.

Moving the constraint to the objective

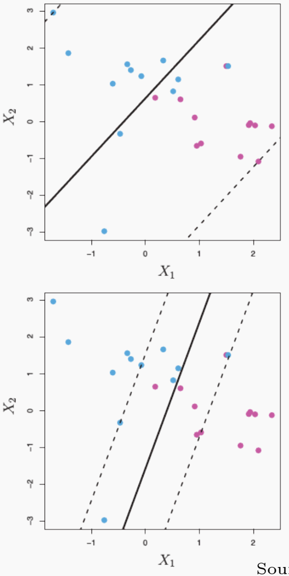
- Remove $\sum_i \xi_i \leq C$ and put it in the objective:

$$\begin{aligned} \min_{\beta_0, \beta, \xi_i \geq 0} \quad & \frac{1}{2}\|\beta\|^2 + \lambda \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i. \end{aligned}$$

- Tuning SVCs:
  - small $\lambda \to$ large margin
  - large $\lambda \to$ narrow margins
  - $\lambda = \infty$ produces the hard margin solution

Bias-variance trade-off examples

Lower to higher tuning parameter λ.



Source: Introduction to Statistical Learning

Introduction to convex optimization

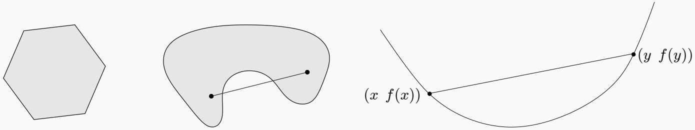
- Unconstrained optimization $\rightarrow$ minimize in Euclidean space:

$$\min_{x \in \mathbb{R}^n} \quad f(x). \tag{3}$$

- Constrained optimization $\rightarrow$ minimization respect to $X \subset \mathbb{R}^n$:

$$\min_{x \in X} \quad f(x). \tag{4}$$

- Convexity can be defined on sets and functions:



Unconstrained optimization

- Necessary condition for point x^* be optimal:

$$\nabla_x f(x^*) = 0.$$

- **Convex**: condition is both necessary and sufficient.

- Example 1: $f_1(x) = ax^2 + bx + c$ with $a > 0$.
  - $\frac{d}{dx} f(x) = 2ax + b = 0 \rightarrow \boxed{x^* = \frac{-b}{2a}}.$
- Example 2: $f_2(x) = x^T A x + 2b^T x + c$ and $A \succ 0$:
  - $\nabla_x f_2(x) = 2Ax + 2b = 0 \rightarrow \boxed{x^* = -A^{-1}b.}$

Constrained optimization

- **Primal problem**: constrained problem in standard form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad i \in \{1, \dots, m\} \\ & h_j(x) = 0 \quad j \in \{1, \dots, p\}. \end{aligned}$$

- **Convex** if $f(x)$, $g_i(x)$ are convex, and $h_j(x)$ are affine.
- **Slater's condition** $\rightarrow$ establishes strong duality.
- **Duality theory** provides:
  - Optimality analysis.
  - Algorithmic tools.
  - Theoretical insights.

Dual problem formulation

1. Construct the Lagrangian:

$$L(x, \lambda, \nu) = f(x) + \sum_i \lambda_i g_i(x) + \sum_j \nu_j h_j(x).$$

2. **Dual function**: the minimum of the Lagrangian over x:

$$q(\lambda, \nu) = \min_x L(x, \lambda, \nu).$$

3. **Dual problem**: maximization of the dual function over $\lambda_i \geq 0$:

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^m, \nu \mathbb{R}^p} \quad & q(\lambda, \nu) \\ \text{s.t.} \quad & \lambda_i \geq 0 \quad \forall i. \end{aligned} \tag{5}$$

Necessary conditions for optimality

Karush-Kuhn-Tucker (KKT) conditions:

- The minimization w.r.t. x of the Lagrangian is an unconstrained problem (step 2). Therefore,

$$\nabla_x L(x^*, \lambda, \nu) = 0 \tag{6}$$

  is necessary for any candidate solution x^*.

- Feasibility is also required:

$$\begin{aligned} g_i(x^*) &\leq 0 \quad \forall i && \text{(7a)} \\ h_j(x^*) &= 0 \quad \forall j && \text{(7b)} \\ \lambda_i^* &\geq 0 \quad \forall i && \text{(7c)} \\ \nu_j^* &\in \mathbb{R} \quad \forall j, && \text{(7d)} \end{aligned}$$

- These equations define a system to recover primal variables.

Dual problem of the SVC

Motivation for the dual problem

- **Primal problem:** linear classifier that should
  - maximize the distance between the points and the decision boundary (maximize margin).
  - misclassify as few points as possible.

$$\begin{aligned} \min_{\beta_0, \beta, \xi_i \geq 0} \quad & \frac{1}{2}\|\beta\|^2 + \lambda \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i. \end{aligned}$$

- **Dual problem:**
  - Allows for efficient computations (used in solvers).
  - Allows a natural derivation of kernel methods.

- **Consideration**: SVC is a convex problem.
  - Strong duality holds.

Derivation of the dual problem

- Standard SVC:

$$
\begin{aligned}
\min_{\beta_0, \beta, \xi_i \geq 0} \quad & \frac{1}{2}\|\beta\|^2 + \lambda \sum_{i=1}^{N} \xi_i \\
\text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i \in \{\, 1, \ldots, N \,\}.
\end{aligned}
$$

- Lagrangian (it's a long expression):

$$
L(\beta, \xi, \alpha, \mu) = f(\beta, \xi) + \sum_i \alpha_i g_i(\beta, \xi) + \mu_i \xi_i
$$

- First order condition: gradient with respect to β, β_0 and ξ

$$
\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, \qquad 0 = \sum_{i=1}^{N} \alpha_i y_i, \qquad \alpha_i = \lambda - \mu_i, \quad \forall i.
$$

- With feasibility $\rightarrow$ the system of equations can be solved exactly.

Dual problem of SVCs

- Substituting on the primal problem we obtain the dual:

$$\max_{0 \leq \alpha_i \leq \lambda} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j$$
$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

- Sequential Minimal Optimization (SMO) solves the problem very efficiently.
- Only support vectors have $\alpha_i \neq 0$.
- At test time, we use

$$y_{\text{test}} \leftarrow \text{sign}[\beta_0 + \beta^T x_{\text{test}}].$$

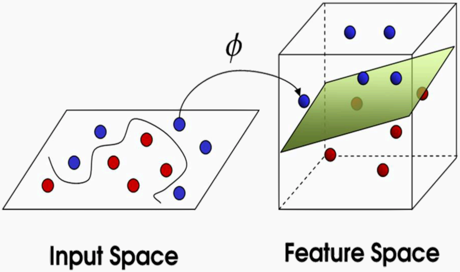
Extension to Non-linear Boundaries

Transforming the Data

- Training a polynomial model is just training a linear model on data with transformed predictors:

$$\phi : \mathbb{R} \to \mathbb{R}^4$$
$$\phi(x) = (x^0, x^1, x^2, x^3)$$

- $\mathbb{R}$ is the ***input space***; $\mathbb{R}^4$ is the ***feature space***.



Generalization:

1. Apply transform $\phi : \mathbb{R}^J \to \mathbb{R}^{J'}$ on training data

$$x_n \mapsto \phi(x_n)$$

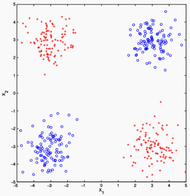
   where typically J' is much larger than J.

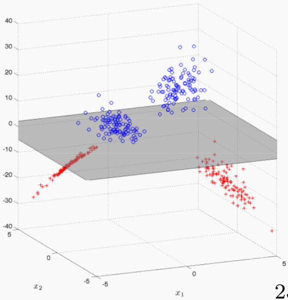
2. Train an SVC on the transformed data

$$\{(\phi(x_1), y_1), \ldots, (\phi(x_N), y_N)\}$$

- XOR example:

$$\phi(x) = (x_1, x_2, x_1 x_2)$$





Training in the feature space

- We can train the SVC in the feature space:

$$\max_{0 \le \alpha_i \le \lambda} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \phi(x_i)^T \phi(x_j)$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

- Designing $\phi(x)$ can be hard $\rightarrow$ (maybe not with NN).
- Computing ϕ explicitly can be costly.
- We are only interested in computing $\phi(x_i)^T \phi(x_j)$.

The ***inner product*** between two vectors is a measure of the similarity of the two vectors.

Definition

Given a transformation $\phi : \mathbb{R}^J \to \mathbb{R}^{J'}$, from input space $\mathbb{R}^J$ to feature space $\mathbb{R}^{J'}$, the function $K : \mathbb{R}^J \times \mathbb{R}^J \to \mathbb{R}$ defined by

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j), \quad x_i, x_j \in \mathbb{R}^J$$

is called the ***kernel function*** of ϕ.

Generally, ***kernel function*** may refer to any function $K : \mathbb{R}^J \times \mathbb{R}^J \to \mathbb{R}$ that measure the similarity of vectors in $\mathbb{R}^J$, without explicitly defining a transform ϕ.

Kernel Trick

- For a choice of kernel $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$,

$$\max_{0 \le \alpha_i \le \lambda} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j {\color{red} K(x_i, x_j)}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

  without computing the mappings $\phi(x_i), \phi(x_j)$.

- This way of training a SVC in feature space without explicitly using the mapping ϕ is called ***the kernel trick***.

Kernel Functions

Common kernel functions include:

- **Polynomial Kernel**

  $$K(x_1, x_2) = (x_1^\top x_2 + 1)^d$$

  where d is a hyperparameter

- **Radial Basis Function Kernel**

  $$K(x_1, x_2) = \exp\left\{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right\}$$

  where σ is a hyperparameter

- **Sigmoid Kernel**

  $$K(x_1, x_2) = \tanh(\kappa x_1^\top x_2 + \theta)$$

  where κ and θ are hyperparameters.

Relationship to Logistic Regression

SVCs via Loss + Penalty

- General structure for ***classification*** and ***regression***:

$$\min_{\beta} \quad J(X, y, \beta) + \lambda P(\beta),$$

  - $J(X, y, \beta)$ is a general loss function.
  - $P(\beta)$ is a general regularizer.

- SVCs take a similar form as well:

$$\min_{\beta, \beta_0} \quad \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \beta^T x_i)] + \lambda \|\beta\|^2,$$

$$y_i(\beta_0 + \beta^T x) \geq 1 - \xi_i \ \rightarrow \ \xi_i = 1 - y_i(\beta_0 + \beta^T x) \geq 0$$
$$\rightarrow \ \xi_i = \max[0, 1 - y_i(\beta_0 + \beta^T x)].$$

SVMs via Loss + Penalty

We can extend SVCs to incorporate feature maps ϕ:

$$\min_{\beta,\beta_0} \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \beta^T \phi(x_i))] + \lambda\|\beta\|^2.$$

Notice that from KKT condition, $\beta = \sum_{j=1}^{N} y_j \alpha_j \phi(x_j)$:

$$\min_{\beta,\beta_0} \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \sum_{j=1}^{N} y_j \alpha_j \phi(x_j)^T \phi(x_i))] + \lambda\|\beta\|^2.$$

and introducing kernel $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

$$\min_{\beta,\beta_0} \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \sum_{j=1}^{N} y_j \alpha_j K(x_i, x_j))] + \lambda\|\mathbf{K}^{1/2}\alpha\|^2.$$

SVMs via Loss + Penalty and Kernels

- General cost function for SVMs:

$$\max[0, 1 - yf(x)] + \lambda\|\mathbf{K}^{1/2}\alpha\|^2$$

with

$$\boxed{f(x) = \beta_0 + \sum_{j=1}^{N} \alpha_j K(x, x_j)}$$

- Our training variables are now β_0 and α_j.
- We perform classification as usual:

$$y_{\text{test}} \leftarrow \text{sign}[f(x)].$$

Logistic regression

- Given training data $x_i \in \mathbb{R}^p$, $y_i \in \{ 1, 0 \}$.
- Probabilities $\rightarrow$ based on the logistic function:

$$p = P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}},$$

- Likelihood $\rightarrow$ Bernoulli distribution:

$$J_l(X, y, \beta) = \prod_{i=1}^{N} P(y = y_i|x) = \prod_{i=1}^{N} p^{y_i}(1 - p)^{1-y_i}$$

- Objective $\rightarrow$ maximum log-likelihood:

$$\max_{\beta_0, \beta} \quad \log(J_l(X, y, \beta)) \iff$$

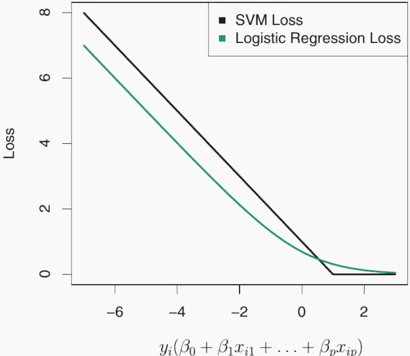
$$\min_{\beta_0, \beta} \quad -\sum_{i=1}^{N} \left[y_i \log(1 + e^{-(\beta_0 + \beta^T x_i)}) + (1 - y_i) \log(1 + e^{(\beta_0 + \beta^T x_i)} \right]$$

Kernel Logistic Regression (KLR)

- Logistic regression $\rightarrow$ loss function + penalization.
- We incorporate $f(x)$ based on kernels:

$$\min_{\beta_0, \alpha_i} \quad -\sum_{i=1}^{N} \left[y_i \log(1 + e^{-f(x_i)}) + (1 - y_i) \log(1 + e^{f(x_i)}) \right] + P(\alpha),$$

$$f(x) = \beta_0 + \sum_{j=1}^{N} \alpha_j K(x, x_j).$$



Comparison between SVMs and KLR

- Classification performance is very similar.
- KLR provides estimates of class probabilities.
- KLR generalizes naturally to M-class classification.
- KLR converges to the maximum margin classifier.
- KLR is computationally more expensive, $O(N^3)$ vs. $O(N^2m)$.
- In SVMs many α_i are zero $\rightarrow$ data compression.
- In KLR all α_i are typically non-zero.

Logistic regression as Neural Network

- Logistic regression constitutes a NN of a single neuron.
- Sigmoid function: $\sigma(z) = 1/(1 + \exp(-z))$.