Hinge loss and support vector machines (SVMs)
SVMs: Kernels, margins, & sparsity
Decision Trees & Forests
Logistic Regression & GP Classification

\[ x = \{x_1, x_2, \ldots, x_N\} \quad y = [y(x_1), \ldots, y(x_N)]^T \]

\[ p(t_n \mid y_n) = \text{Ber}(t_n \mid \sigma(y_n)) \]

**Parametric** logistic regression based on \( M \) features:

\[ y_n = w^T \phi(x_n) \quad \phi(x_n) \in \mathbb{R}^{M \times 1} \]

\[ p(w) = \mathcal{N}(w \mid 0, \alpha^{-1} I_M) \quad w \in \mathbb{R}^{M \times 1} \]

**Learning:** Estimation of \( M \)-dim. weight vector

**Nonparametric** Gaussian Process classification based on kernel:

\[ p(y) = \mathcal{N}(y \mid 0, K) \quad K \in \mathbb{R}^{N \times N} \quad K_{ij} = k(x_i, x_j) \]

**Learning:** Estimation of \( N \) function values \( y \) at each of the \( N \) training examples

**Mercer:** Corresponding features exist, but no need to compute
**Gaussian Process Classification**

\[
p(y) = \mathcal{N}(y \mid 0, K)
\]

\[
K_{ij} = k(x_i, x_j)
\]

\[
p(t_n \mid y_n) = \text{Ber}(t_n \mid \sigma(y_n))
\]

**Interpretation of function values** \(y\)

- **Positive:** \(p(t_n = 1 \mid f_n) > 0.5\)
- **Zero:** \(p(t_n = 1 \mid f_n) = 0.5\)
- **Negative:** \(p(t_n = 1 \mid f_n) < 0.5\)

**Interpretation of kernel values** \(K_{ij}\)

- **Positive:** Likely have same label
- **Zero:** Inputs are totally independent
- **Negative:** Likely have different labels

*Kernel encodes these relationships for all observation pairs.*
Logistic Likelihood Function

\[ p(t_n \mid y_n) = \text{Ber}(t_n \mid \sigma(y_n)) \quad t_n \in \{0, 1\} \]

\[ p(t_n = 1 \mid y_n) = \sigma(y_n) \]

\[ p(t_n = 0 \mid y_n) = 1 - \sigma(y_n) = \sigma(-y_n) \]

- Write compactly using spin representation of labels:
  \[ \tilde{t}_n = 2t_n - 1 \in \{-1, +1\} \]
  \[ p(t_n \mid y_n) = \sigma(\tilde{t}_n y_n) \]

- Negative log-probability is called the logarithmic loss:
  \[ -\log p(t_n \mid y_n) = \log(1 + e^{-\tilde{t}_n y_n}) \]

  Prefers functions such that \( \tilde{t}_n y_n \gg 0 \)
Learning versus Decision Making

Perceptron Mark 1 Computer, Frank Rosenblatt, late 1950s

**Decision Rule:**  \[ y(x) = \mathbb{I}(w^T \phi(x) > 0) \]

If our goal is classification, why bother to learn probabilities?

- **Optimization:** Hard to find \( w \) minimizing training error rate (non-convex)
- **Uniqueness:** Many \( w \) may achieve optimal error rate
$$\hat{w} = \arg \min_w \frac{\lambda}{2} ||w||^2 + \sum_{n=1}^{N} L(\tilde{t}_n w^T \phi(x_n))$$

\(\tilde{t}_n \in \{+1, -1\}\)
Losses for Binary Classification

\[ \hat{w} = \arg \min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} L(\tilde{t}_n w^T \phi(x_n)) \quad \tilde{t}_n \in \{+1, -1\} \]

**Training Error Rate (0-1 loss)**
- For many applications, the objective we care about
- Hard to optimize (gradients zero everywhere, non-convex)
- Cannot distinguish top-performing training classifiers

**Logistic Regression** (logarithmic loss)
- Estimates label probabilities for calibrated decision-making
- Easy to optimize (convex, smooth bound on 0-1 loss)
- Scalability problems with large datasets and many features

**Support Vector Machine** (hinge loss)
- Does not estimate valid probability distribution on labels
- Possible to optimize (convex, non-smooth bound on 0-1 loss)
- Chooses boundary which maximizes margin of training data
- Gives sparse solutions, allowing greater scalability
Support Vector Machines

\[
\hat{w} = \arg \min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} L(\tilde{t}_n w^T \phi(x_n)) \\
\tilde{t}_n \in \{+1, -1\} \\
L(\eta) = \max(0, 1 - \eta) \triangleq (1 - \eta)_+
\]

**SVM Penalized Objective**

\[
\hat{w} = \arg \min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} (1 - \tilde{t}_n w^T \phi(x_n)) +
\]

**SVM Constrained Objective**

\[
\hat{w} = \arg \min_{w, \zeta} \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \zeta_n \quad \text{subject to} \quad \tilde{t}_n w^T \phi(x_n) \geq 1 - \zeta_n \\
\zeta_n \geq 0
\]

**Quadratic Program:** Quadratic function with linear constraints based on “slack variables” \( \zeta_n \)
Hinge loss and support vector machines (SVMs)
SVMs: Kernels, margins, & sparsity
Decision Trees & Forests
**Maximum Margin Hyperplanes**

- **Margin**: For a hyperplane which perfectly separates training data, orthogonal distance of closest training example to plane.
- **Intuition**: Expect similar features to have similar labels, so would like decision boundary as far as possible from data.
- **Statistical Learning Theory**: Formal bounds on generalization performance (test error) of large-margin classifiers.
Margins and SVMs

\[ \phi(x) = z = z_\perp + r \frac{w}{\|w\|} \]

\[ r = \frac{w^T z}{\|w\|} \text{ signed distance from boundary} \]

\[ z_\perp = \text{ orthogonal projection onto boundary} \]

- **Accuracy:** Classify all training data correctly by enforcing
  \[ \tilde{t}_n w^T \phi(x_n) > 0 \]

- **Margin:** Maximize distance of closest point to boundary
  \[ \max_w \min_{n=1,\ldots,N} \frac{\tilde{t}_n w^T \phi(x_n)}{\|w\|} \]

- **Invariance:** Scale \( w \) so closest point distance 1 from boundary
  \[ \hat{w} = \arg \min_w \frac{1}{2} \|w\|^2 \text{ subject to } \tilde{t}_n w^T \phi(x_n) \geq 1 \]
Margins and SVMs

**SVM:** Maximizes margin, but slack variables allow some data to be misclassified with appropriate penalties.

$$\hat{w} = \arg \min_w \lambda \frac{1}{2} ||w||^2 + \sum_{n=1}^{N} \zeta_n$$

subject to 

$$\tilde{t}_n w^T \phi(x_n) \geq 1 - \zeta_n \quad \zeta_n \geq 0$$

- **Accuracy:** Classify all training data correctly by enforcing 

$$\tilde{t}_n w^T \phi(x_n) > 0$$

For SVM, some violations

- **Margin:** Maximize distance of closest point to boundary

$$\max \min_{w, n=1,\ldots,N} \frac{\tilde{t}_n w^T \phi(x_n)}{||w||}$$

SVM maximizes margin excluding some outliers

- **Invariance:** Scale $w$ so closest point distance 1 from boundary

$$\hat{w} = \arg \min_w \frac{1}{2} ||w||^2$$

subject to 

$$\tilde{t}_n w^T \phi(x_n) \geq 1$$
\[ \hat{w} = \arg \min_w \frac{\lambda}{2} ||w||^2 + \sum_{n=1}^N (1 - \tilde{t}_n w^T \phi(x_n))_+ \]

- Optimal solution takes following form:
  \[ \hat{w} = \sum_{n=1}^N \alpha_n \tilde{t}_n \phi(x_n) \quad \alpha_n \geq 0 \]

- Here, the \( \alpha_n \) are Lagrange multipliers for constrained QP
- Because loss exactly zero for arguments greater than one, only a sparse subset of training examples have \( \alpha_n > 0 \)
- Training examples with non-zero weight are support vectors

**SVM Quadratic Program:**

\[ \hat{w} = \arg \min_{w, \zeta} \frac{\lambda}{2} ||w||^2 + \sum_{n=1}^N \zeta_n \quad \text{subject to} \quad \tilde{t}_n w^T \phi(x_n) \geq 1 - \zeta_n \]

\[ \zeta_n \geq 0 \]
Support Vectors and Slack Variables

\[ \hat{w} = \sum_{n=1}^{N} \alpha_n \tilde{t}_n \phi(x_n) \]
\[ \alpha_n \geq 0 \]

Data points with non-zero weight:
- Points with minimum margin (on optimized boundary)
- Points which violate margin constraint, but are still correctly classified
- Points which are misclassified

For all other training data, features have \textit{no impact} on learned weight vector

Linear decision boundary in feature space, where data violating margin have nonzero “slack variables”
SVMs and Kernels

Optimal weights always take this form, with non-zero weights only for support vectors.

Primal SVM:

\[ \hat{w} = \arg \min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} (1 - \tilde{t}_n y_n)^+ \]

Dual SVM:

\[ \hat{y} = \arg \min_y \frac{\lambda}{2} y^T K^{-1} y + \sum_{n=1}^{N} (1 - \tilde{t}_n y_n)^+ \]

Kernel Tricks:

\[ K \in \mathbb{R}^{N \times N}, \quad K_{ij} = k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \]

\[ y = K \beta, \quad y_i = w^T \phi(x_i) \quad \|w\|^2 = y^T K^{-1} y \]
SVMs versus Gaussian Processes

Logistic Regression:

\[ \hat{w} = \arg\min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \log(1 + e^{-\tilde{t}_n w^T \phi(x_n)}) \]

GP Classification:

\[ \hat{y} = \arg\min_y \frac{\lambda}{2} y^T K^{-1} y + \sum_{n=1}^{N} \log(1 + e^{-\tilde{t}_n y_n}) \]

Dual SVM:

\[ \hat{y} = \arg\min_y \frac{\lambda}{2} y^T K^{-1} y + \sum_{n=1}^{N} (1 - \tilde{t}_n y_n) + \]

Primal SVM:

\[ \hat{w} = \arg\min_w \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} (1 - \tilde{t}_n w^T \phi(x_n)) + \]
Hinge Loss versus Logistic Loss

- With identical features or kernels, SVMs and GPs usually have very similar accuracy.
- Logistic regression approximately maximizes margin.
- SVM algorithms are often more scalable by exploiting data sparsity.
- Post-hoc methods for extracting probabilities from SVMs exploit logistic regression analogy.

Support Vector Machines

Logistic Regression & Gaussian Process Classifiers

Training Error Rate

Error → Correct Decision
Support Vectors and Slack Variables

Support vectors (green) for data separable by radial basis function kernels, and non-linear margin boundaries

Linear decision boundary in feature space, where data violating margin have nonzero “slack variables”

\[ y(x) = w^T \phi(x) \]

\[ y = 1 \]

\[ y = -1 \]

\[ \xi > 1 \]

\[ \xi < 1 \]

\[ \xi = 0 \]
How Many Support Vectors?

Only need to evaluate kernel at support vectors, not all training data.

But there may still be a lot of support vectors.
Hinge loss and support vector machines (SVMs)
SVMs: Kernels, margins, & sparsity
Decision Trees & Forests
Example: Decision Trees for Classification

Features: color, shape, size

- Each node in tree splits examples according to a single feature
- Leaves have categorical distribution learned from labels of all training data whose path through tree ends there
Example: Decision Trees for Regression

- Each node in tree splits examples according to a single feature
- Leaves have Gaussian distribution (mean and variance) learned from values of all training data whose path through tree ends there
Decision Tree Learning

Leaves of tree: \( R_1, R_2, \ldots, R_J \).

**Regression:**
\[
p(t_n \mid x_n \in R_j) = \mathcal{N}(t_n \mid \mu_j, \sigma_j^2)
\]

**Binary Classification:**
\[
p(t_n \mid x_n \in R_j) = \text{Ber}(t_n \mid \mu_j)
\]

- Let \( w \) denote decision tree parameters: structure, splits, leaf distributions
- Conceptually, we would like to minimize the following objective:

\[
f(w) = \lambda L(w) - \sum_{n=1}^{N} \log p(t_n \mid x_n, w)
\]

- Here, \( L(w) \) is some measure of tree complexity: count of number of nodes, negative log of prior on trees, etc.
PROBLEM: There are an enormous number of tree structures & split thresholds. Optimizing leaf distributions given structure is easy, but searching structures is hard. Classic approach: Build tree greedily, using various heuristics to control complexity, and check with validation data.

CART: Classification & Regression Trees, C4.5, ID3, …
Greedy Decision Tree Learning

Leaves of tree: \( R_1, R_2, \ldots, R_J \).

**Regression:**

\[
p(t_n \mid x_n \in R_j) = \mathcal{N}(t_n \mid \mu_j, \sigma_j^2)
\]

**Binary Classification:**

\[
p(t_n \mid x_n \in R_j) = \text{Ber}(t_n \mid \mu_j)
\]

\( N_j = \text{number of training data for leaf } j \)

- Given a candidate tree, estimate leaf parameters via maximum likelihood:
  - Regression:  
    \[
    \mu_j = \frac{1}{N_j} \sum n \mid x_n \in R_j t_n \\
    \sigma_j^2 = \frac{1}{N_j} \sum n \mid x_n \in R_j (t_n - \mu_j)^2
    \]
  - Classification:  
    \[
    \mu_j = \frac{1}{N_j} \sum n \mid x_n \in R_j t_n \\
    \text{Var}(t_n \mid x_n \in R_j) = \mu_j(1 - \mu_j)
    \]

- Greedily, pick the leaf with largest variance, choose new split that minimizes sum of child variances (solvable via exhaustive search)

- Recurse to refine tree. Initialize with all data in single root node.
Controlling Decision Tree Complexity

Leaves of tree: $R_1, R_2, \ldots, R_J$.

**Regression:**

$$p(t_n \mid x_n \in R_j) = \mathcal{N}(t_n \mid \mu_j, \sigma_j^2)$$

**Binary Classification:**

$$p(t_n \mid x_n \in R_j) = \text{Ber}(t_n \mid \mu_j)$$

- **PROBLEM:** Once number of leaves in tree equals $N$, can set variance in each leaf to zero. This perfectly predicts each training data, but overfits!

- **Pruning:** Recursively prune leaves from tree, use validation data to check whether increases or decreases prediction accuracy

- **Random forests:** Randomly subsample data and features. Greedily fit a decision tree to each, then average their predictions. This is a form of the bootstrap that can lead to enormous gains in prediction accuracy.
Microsoft Kinect Human Pose Estimation

Shotton et al., PAMI 2012