Logistic Regression (part 2)
Training Logistic with Selection Bias

- Biased data happens when

\[
\text{Log Likelihood} = \sum_{i=1}^{N} \frac{\alpha_i}{\pi_i} \log P(C_i = k|\phi_i)
\]

where \(\pi_i\) is probability of seeing sample \(i\) and \(\alpha_i\) is prevalence of \(i\) in the population

- Generally, a good idea to test training data for different weights
  - The weights can be used to protect against sampling designs which could cause selection bias.
  - The weights can be used to protect against misspecification of the model.
Multiclass Logistic Regression (MLR)

- Consider $K$ classes and $N$ observations
- Let $C_i$ be the class of the $i$-th example with feature vector $\phi_i$

$$P(C_i = k | \phi_i) = \frac{\exp(w_k^T \phi_i)}{\sum_{h=1}^{K} \exp(w_h^T \phi_i)} \quad \text{a.k.a. softmax}$$

- Easier to assume 1-of-K coding of target variable $t_i$
- Log-likelihood function (learned via iterative reweighted least squares, has unique maximum)

$$\sum_{i=1}^{N} \sum_{k=1}^{K} t_{i,k} \log \frac{\exp(w_k^T \phi_i)}{\sum_{h=1}^{K} \exp(w_h^T \phi_i)} = \sum_{i=1}^{N} \sum_{k=1}^{K} t_{i,k} w_k^T \phi_i - \sum_{i=1}^{N} \log \sum_{h=1}^{K} \exp(w_h^T \phi_i)$$

- Multiclass Logistic Regression (MLR)
What if Feature Vector is Latent in MLR?

- Example: word2vec (Mikolov et al., 2013)
- Words in a 5-word sentence: \( x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} \)
- \( x_t \) is word of interest, other words are nearby words

\[
P[\pi_{t-2}, \pi_{t-1}, \pi_{t+1}, \pi_{t+2} | \pi_t, \mathbf{W}, \Phi] = \prod_{j=-2, -1, 1, 2} P[\pi_{t+j} | \pi_t, \mathbf{W}, \Phi]
\]

where

\[
P[\pi_{t+j} | \pi_t, \mathbf{W}, \Phi] = \frac{\exp(\mathbf{w}_{\pi_t}^T \phi_{\pi_{t+j}})}{\sum_{k=1}^{K} \exp(\mathbf{w}_{\pi_t}^T \phi_k)}
\]

Learning hard via gradient descent. Noise contrastive estimation used instead (or MCMC or Hierarchical Softmax).
- We go back to learning feature vectors in neural network representations
- For now we keep engineering our own features
Classification without a Model
Issues with Least Squares Classification

Least Squares Solution

\[
\sum_j (y(x_j) - \text{hyperplane})^2
\]

LS cares too much about reducing classification error of well separable items
Input – Output of Our Problem

- **Input Data**
  - Item features \( D_x = \{x_1, \ldots, x_N\}, \quad x_n \in \mathbb{R}^k, k \geq 1 \)
  - Item labels \( D_t = \{t_1, \ldots, t_N\}, \quad t_n \in \{-1, 1\} \)

- **Output:** given new data point \( x' \)
  - Find \( y(x') \) that best approximates possible value of \( t' \)
In what follows we will ”ignore” all points that are far away from the border (separating hyperplane).

Two algorithms:
- Perceptron
- SVMs
Perceptron

Model:

\[ f(x) = \sum_{i=1}^{m} w_i x_i + b \]

\[ y = \text{sign}[f(x)] \]

Dot product is product of:
(i) projection of \( x \) onto \( w \), and (ii) the length of \( w \)

Dot product is 0 if \( x \) is perpendicular to \( w \)
Add \( b \), if >0 then positive class, if <0 then negative class
Perceptron

Model:

\[ f(x) = \sum_{i=1}^{m} w_i x_i + b \]

\[ y = \text{sign}[f(x)] \]

Learning:

if \( y(j)(\sum_{i=1}^{m} w_i x_i(j) + b) \leq 0 \)

then \( w \leftarrow w + \eta y(j)x(j) \) and \( b \leftarrow b + \eta y(j) \)

Iterate over training examples until error is below a pre-specified threshold \( \Theta \)
Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space \((\phi_1, \phi_2)\). The top left plot shows the initial parameter vector \(w\) shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.
procedure LEARNPERCEPTRON(data, numIters, learnRate)
    \( w \leftarrow 0 \) (for \( p = 1 \ldots numAttrs \))
    \( b \leftarrow 0 \)
    \( \eta \leftarrow learnRate \)
    for iter \( \leq \) numIters do
        for \( i = (x_i, y_i) \in data \) do
            \( \hat{y}_i = \text{sign}(w \cdot x_i + b) \)
            if \( y_i \hat{y}_i \leq 0 \) then
                \( e = \eta y_i \)
                \( w \leftarrow w^{old} + e x_i \)
                \( b \leftarrow b + e \)
            end if
        end for
    end for
    return \( w, b \)
end procedure
Perceptron components

- **Model space**
  - Set of weights $w$ and $b$ (hyperplane boundary)

- **Search algorithm**
  - Iterative refinement of weights and $b$

- **Score function**
  - Minimize misclassification rate
Perceptron Algorithm Convergence

- *Perceptron convergence theorem:*
  - If exact solution exists (i.e., if data is linearly separable), then perceptron algorithm will find an exact solution in a finite number of steps
  - The “finite” number of steps can be very large. The smaller the gap, the longer the time to find it

- If data not linearly separable?
  - Algorithm will not converge, and cycles develop. The cycles can be long and therefore hard to detect
How Can We Extend the Perceptron Idea?

- Is there a better formulation for two classes?

- What about find hyperplane that is equidistant to circled points?

- We arrived at Maximum Margin Classifiers (e.g. Support Vector Machines)
Linear Methods \rightarrow \text{Linear Boundaries?}

Data from three classes, with linear decision boundaries

Linear boundaries in the five-dimensional space
\[ \phi(x) = (X_1, X_2, X_1X_2, X_1^2, X_2^2) \]

\( \phi \equiv \phi(x) \)

\( \phi \) is known as a "kernel"
How to Find The "Support" Points? (assuming linearly separable data)

- Back to our linear model (with non-linear features \( \phi \))

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

- For two classes, if class \( t_n \in \{-1, 1\} \) of item \( x_n \)

- Then \( t_n y(\tilde{x}_n) > 0 \) means correctly classified

- Thus, distance of \( x_n \) from decision hyperplane is the maximum minimum distance

  \[
  \frac{t_n y(x_n)}{\|w\|} = \frac{t_n (w^T \phi(x_n) + b)}{\|w\|} \rightarrow \arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_n \left[ t_n (w^T \phi(x_n) + b) \right] \right\}
  \]

Maximum margin solution
Modified Solution

\[
\arg\max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[ t_n (\mathbf{w}^T \phi(x_n) + b) \right] \right\}
\]

- Too complex to compute
- Brings hyperplane too close to point

- We can recast problem into another optimization problem

\[
t_n (\mathbf{w}^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]

- Data points for which the equality holds are said to be *active*
  - All other data points are *inactive*
  - There is always at least one active constraint
  - When margin is maximized there will be two active constraints

Constraint takes care of this part of the optimization
The original problem becomes (as \( \arg \max ||w||^{-1} = \arg \min ||w||^2 \)) a quadratic programming problem

\[
\arg \max_{w,b} \left\{ \frac{1}{||w||} \min_n [t_n (w^T \phi(x_n) + b)] \right\}
\]

- The original problem becomes (as \( \arg \max ||w||^{-1} = \arg \min ||w||^2 \)) a quadratic programming problem

\[
\arg \min_{w,b} \frac{1}{2} ||w||^2
\]

s.t.

\[
t_n (w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]

Quadratic programming
Constrained optimization

- Linear programming (LP) is a technique for the optimization of a linear objective function, subject to linear constraints on the variables.

- Quadratic programming (QP) is a technique for the optimization of a quadratic function of several variables, subject to linear constraints on these variables.
The original problem becomes (as \( \arg\max \|w\|^{-1} = \arg\min \|w\|^2 \)) a quadratic programming problem

\[
\arg\max_{w,b} \left\{ \frac{1}{\|w\|} \min_n [t_n (w^T \phi(x_n) + b)] \right\}
\]

- The original problem becomes (as \( \arg\max \|w\|^{-1} = \arg\min \|w\|^2 \)) a quadratic programming problem

\[
\arg\min_{w,b} \frac{1}{2} \|w\|^2
\]

s.t.

\[
t_n (w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]
Dual problem

- For a convex problem (no local minima) there is a dual problem that is equivalent to the primal problem (i.e. we can switch between them)

- Primal objective: minimize wrt to M feature variables
  - Quadratic programming problem with parameters w, b

- Dual objective: maximize wrt to N instances
  - Dual depends on inner product between feature vectors
  - → simpler quadratic programming problem
The original problem becomes (as \( \arg \max ||w||^{-1} = \arg \min ||w||^2 \)) a quadratic programming problem

\[
\arg \min_{w,b} \frac{1}{2} ||w||^2
\]

s.t.

\[
t_n (w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]

- We can solve constrained optimization problem via Lagrange multipliers \( a_n \geq 0 \)

\[
L(w, b, a) = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} a_n \left\{ t_n (w^T \phi(x_n) + b) - 1 \right\}
\]
Solution to Modified Problem

\[ L(w, b, a) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n (w^T \phi(x_n) + b) - 1 \right\} \quad (1) \]

- Setting derivative of \( L(w,b,a) \) w.r.t. \( w \) and \( b \) to zero we get:

\[ w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \]

\[ 0 = \sum_{n=1}^{N} a_n t_n. \]

- Eliminating \( w \) and \( b \) from equation (1) using these conditions:

\[ \tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]

s.t.

\[ \begin{align*}
    a_n &\geq 0, \quad n = 1, \ldots, N, \\
    \sum_{n=1}^{N} a_n t_n &= 0.
\end{align*} \]

\[ k(x, x') = \phi(x)^T \phi(x') \]

\( k \) is the "kernel"
Important Properties

- Satisfies Karush-Kuhn-Tucker (KKT) conditions so a solution to the nonlinear programming is optimal, as the following properties hold:

\[
\begin{align*}
    a_n & \geq 0 \\
    t_n y(x_n) - 1 & \geq 0 \\
    a_n \{t_n y(x_n) - 1\} & = 0.
\end{align*}
\]

- Note that either \( a_n = 0 \) or \( t_n y(x_n) = 1 \)
  - What does it mean?
  - Point is either at the margin or we don't care about it in the sum

Modified equation:

\[
\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)
\]
Example with Gaussian Kernel

Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant \( y(x) \) obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

Form (6.23). Although the data set is not linearly separable in the two-dimensional data space \( x \), it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus the training data points are perfectly separated in the original data space.

This example also provides a geometrical insight into the origin of sparsity in the SVM. The maximum margin hyperplane is defined by the location of the support vectors. Other data points can be moved around freely (so long as they remain outside the margin region) without changing the decision boundary, and so the solution will be independent of such data points.

7.1.1 Overlapping class distributions

So far, we have assumed that the training data points are linearly separable in the feature space \( \phi(x) \). The resulting support vector machine will give exact separation of the training data in the original input space \( x \), although the corresponding decision boundary will be nonlinear. In practice, however, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization.

We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified. From (7.19) we see that in the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was classified correctly, and then optimized the model parameters to maximize the margin. We now modify this approach so that data points are allowed to be on the ‘wrong side’ of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance. To do this, we introduce slack variables \( \xi_n \geq 0 \) where \( n = 1, \ldots, N \), with one slack variable for each training data point (Bennett, 1992; Cortes and Vapnik, 1995). These are defined by

\[
\xi_n = 0 \quad \text{for data points that are on or inside the correct margin boundary and} \\
\xi_n = |t_n - y(x_n)| \quad \text{for other points.}
\]

Thus a data point that is on the decision boundary \( y(x_n) = 0 \) will have \( \xi_n = 1 \), and points hyperplane active constraints (support points)
Limitations of linear SVMs

- Linear classifiers cannot deal with:
  - Non-linear concepts
  - Noisy data

- Solutions:
  - Soft margin (e.g., allow mistakes in training data)
  - Network of simple linear classifiers (e.g., neural networks)
  - Map data into richer feature space (e.g., non-linear features) and then use linear classifier
What if data is not linearly separable?

- Add slack variables $\xi_n \geq 0$ to optimization problem defined as 0 if correctly classified not within margin and $\xi_n = |t_n - y(x_n)|$ otherwise

- The new constraints are
  $$t_n y(x_n) \geq 1 - \xi_n, \quad n = 1, \ldots, N$$

- And it is easy to show that the optimization is now to minimize
  $$C \sum_{n=1}^{N} \xi_n + \frac{1}{2} \|w\|^2$$
  for $C > 0$
SVMs for Non-linearly Separable Data

- Minimize

\[ \tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]

s.t.

\[ 0 \leq a_n \leq C \]

\[ \sum_{n=1}^{N} a_n t_n = 0 \]

where \( C \) can be seen as a penalty for misclassification

- For \( C \to \infty \) we recover the linearly separable case
Kernel SVMs

- State-of-the-art classifier (with good kernel)
- Solves computational problem of working in high dimensional space
- Non-parametric classifier (keeps all data around in kernel)
- Learning: $O(N^2)$, but approximations available for $O(N)$
  - **Hint:** Focus computation on good support point candidates
- What is the model space?
Logistic Regression
Relationship Between Logistic Regression and SVMs

- SVM can be described as optimizing

\[
\sum_{n=1}^{N} E_{SV}(y_n, t_n) + \frac{1}{2C} \|w\|^2
\]

where \( E_{SV}(y_n t_n) = [1 - y_n t_n]_+ \)

is known as hinge loss function,

\([.]_+\) is positive part

Logistic Regression generally faster than SVM
Classification \, Contextual Bandits \& Reinforcement Learning

- **Classification task (our current goal):**
  - Given \((x_i, y_i), i=1,2,\ldots\)
  - Goal 1: Learn \(p(Y_i|X_i=x_i)\)
  - Goal 2: Just predict \(y_i\)

- Generative model task
  (interesting if you want to ask questions about \(x\))
  - Given \((x_i, y_i), i=1,2,\ldots\)
  - Goal: Learn \(p(Y_i, X_i)\)

- Contextual Bandits
  - Given \(x_t\) and after performing action \(a\), get reward \(y_t\)
  - Problem is stateless: \(p(y_t | a, x_t, x_{t-1},\ldots) = p(y_t | a, x_t)\)
  - Goal: Learn action \(a\) that maximizes \(p(Y | a, X)\)

- Reinforcement Learning
  - Given \(x_t\) and after performing action \(a\), get reward \(y_t\)
  - Problem is stateful: \(p(y_t | a, x_t, x_{t-1},\ldots) \neq p(y_t | a, x_t)\)
We will see decision trees when we see ensembles