Lecture 8: Machine Learning II

Linear Classifier

Yen-Yi Ho

Department of Statistics

Outline

- The Model
- The Loss Function
- Regularization
- Optimization
- Performance Metrics
- Data Leakage
- Nested Cross-validation

Gene Expression Profiles Predict Clinical Outcomes

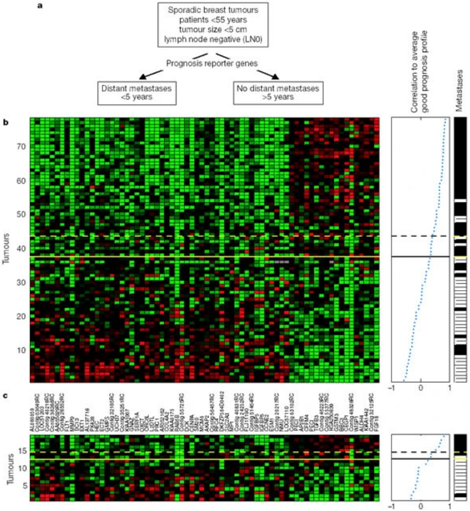


Figure adapted from van 't Veer, L., Dai, H., van de Vijver, M. et al. Gene expression profiling predicts clinical outcome of breast cancer. Nature 415, 530–536 (2002).

Parametric Approach

Input data

$\longrightarrow$ $f(\color{blue}{\mathbf{X}}, \color{red}{\mathbf{W}})$ $\longrightarrow$ **Class Scores**

X: input features

W: parameters / weights

$\rightarrow$ **tumor**

$\rightarrow$ **normal**

Parametric Approach: Linear Classifier

$$f(\mathbf{X},\ \mathbf{W}) = \mathbf{X}\mathbf{W} + \mathbf{b}$$

Input data

$\longrightarrow f(\mathbf{X},\ \mathbf{W}) \longrightarrow$ **Class Scores**

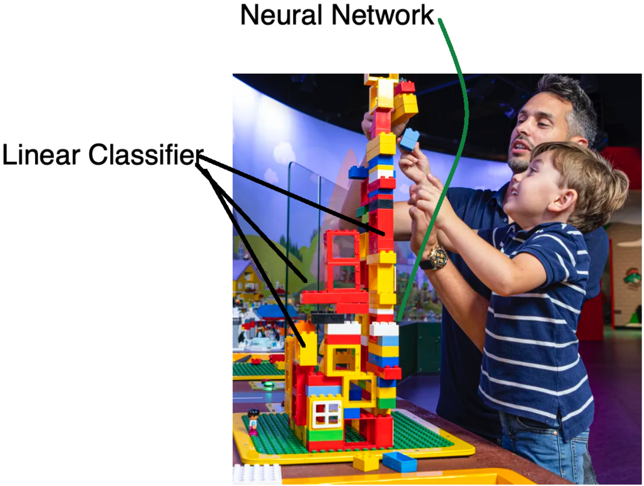
X: input features

W: parameters / weights

→ **tumor**

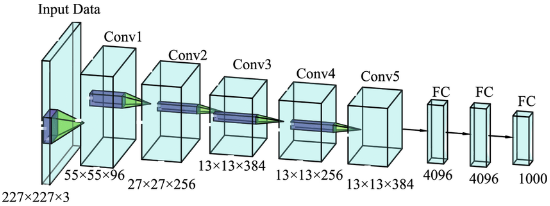
→ **normal**

Building Blocks of A Neural Network



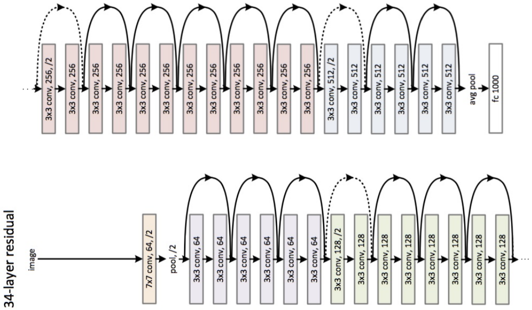
Neural Network

Linear Classifier



Conv: Convolution; FC: Full Connection

[Krizhevsky et al. 2012, AlexNet]



[He et al, 2015, ResNet]

Linear Classifier for Disease Prediction

Input

$$\mathbf{x} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_p \end{bmatrix}$$

Gene expression vector for one patient

Linear model

$$s = \mathbf{w}^\top \mathbf{x} + b$$

$\mathbf{x}$: gene expression

$\mathbf{w}$: learned weights

$$p = \sigma(s)$$

(sigmoid)

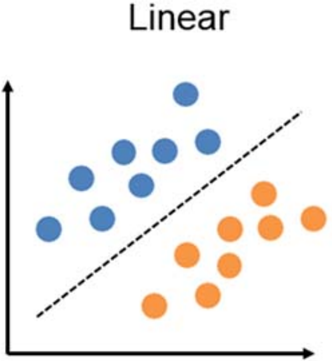
Output

$$p(\text{disease} \mid \mathbf{x})$$

$$\begin{cases} > 0.5 & \Rightarrow \textbf{tumor} \\ \leq 0.5 & \Rightarrow \textbf{normal} \end{cases}$$

Interpreting a Linear Classifier: Geometric Viewpoint

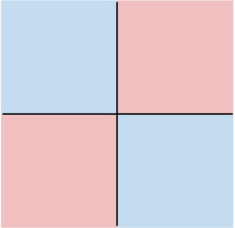
$$f(\mathbf{X},\ \mathbf{W}) = \mathbf{XW} + \mathbf{b}$$



Hard Cases for a Linear Classifier

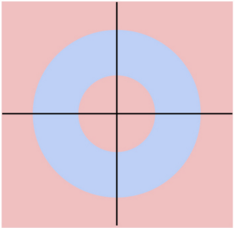
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



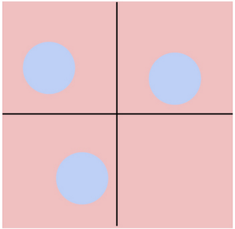
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



Linear Classifier - Choose a good W

- Define a loss function
- Minimize the loss function

Loss functions

A loss function tells how good our current classifier is. Given a dataset of N data points:

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Loss over the dataset is a average of loss over all data points:

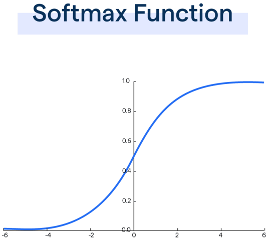
$$L = \frac{1}{N} \sum_i L_i[f(x_i, W), y_i]$$

Softmax function

$$s_i = \mathbf{w}^\top \mathbf{x_i} + b$$

$$p(\text{disease} \mid \mathbf{x}) = p(Y = 1 | X = x_i) = \frac{e^{s_i}}{1 + e^{s_i}}$$

Also named the logistic function.


Softmax Function

The softmax function turns a score (s) into a value (probability) between $[0, 1]$

Minimize the Loss Function

- Maximum likelihood estimation
- Minimize the Kullback-Leibler divergence or cross entropy

$$\mathrm{KL}(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

$$\mathcal{H}(P, Q) = -\sum_x P(x) \log Q(x)$$

Training Objective: Cross-Entropy Loss

True distribution

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

One-hot encoded label (e.g. disease present)

Model prediction

$$\hat{\mathbf{y}} = \begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \text{softmax}(\mathbf{s})$$

Predicted class probabilities

Loss

$$\mathcal{L} = \mathrm{KL}(\mathbf{y} \,\|\, \hat{\mathbf{y}})$$

$$= -\sum_i y_i \log \hat{y}_i$$

Cross-entropy loss

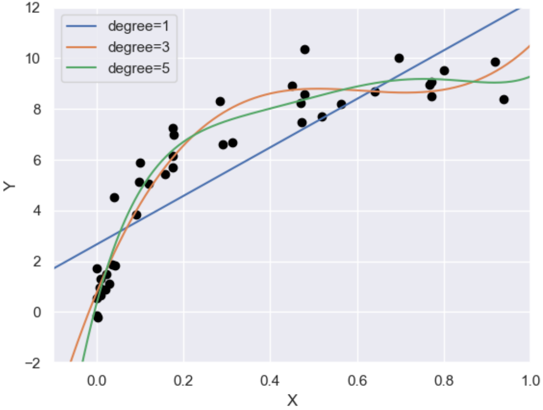
$$\mathcal{L}(w) = -\left[\sum_{i=1}^{N} y_i \log p_i + (1 - y_i) \log(1 - p_i)\right]$$

Outline

- The Model
- The Loss Function
- **Regularization**
- Optimization
- Performance Metrics
- Data Leakage
- Nested Cross-validation

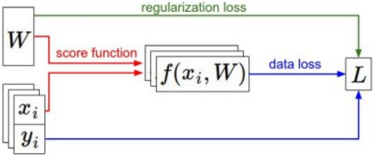
Polynomial Regression

$$
\begin{aligned}
y &= ax + b \\
y &= ax^3 + bx^2 + cx + d
\end{aligned}
$$



Regularziation

$$\mathcal{L}_\lambda(\mathbf{w}) \;=\; \frac{1}{N}\sum_{i=1}^{N} L_i\big(y_i, f(\mathbf{x}_i;\mathbf{w})\big) \;+\; \lambda R(\mathbf{W})$$



- Regularization: prefer simpler models to improve generalization.
- λ is the regularization strength (hyper-parameter).
- L1 regularization: $R(W) = \sum |W|$
- L2 regularization: $R(W) = \sum W^2$
- Elastic net (L1+L2): $R(W) = \sum W^2 + |W|$
- Dropout in neural network

Regularziation

$$\begin{aligned}
\mathbf{x} &= [1,\ 1,\ 1,\ 1] \\
\mathbf{w}_1 &= [1,\ 0,\ 0,\ 0] \\
\mathbf{w}_2 &= [0.25,\ 0.25,\ 0.25,\ 0.25] \\
\mathbf{w}_1^T\mathbf{x} &= \mathbf{w}_2^T\mathbf{x} = 1
\end{aligned}$$

- L2 prefer spread out weights
- L1 tends to have sparse weights

Regularziation

$$
\begin{aligned}
\mathbf{x} &= [1,\ 1,\ 1,\ 1] \\
\mathbf{w}_1 &= [1,\ 0,\ 0,\ 0] \\
\mathbf{w}_2 &= [0.25,\ 0.25,\ 0.25,\ 0.25] \\
\mathbf{w}_1^T\mathbf{x} &= \mathbf{w}_2^T\mathbf{x} = 1
\end{aligned}
$$

- L2 prefer spread out weights
- L1 tends to have sparse weights

Regularziation: Logistic Regression Example

The (regularized) logistic regression objective minimized is:

$$\mathcal{L}(w) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log p_i + (1-y_i)\log(1-p_i)\right] + \lambda R(w)$$

where $p_i = \sigma(w^T x)$ and $R(W)$ is the penalty (L2, L1, elastic-net).

Implement Logistic Regression in scikit-learn

```
clf = LogisticRegression(
        penalty=’l2’,
        C=C,
        solver=’saga’,
        max_iter=5000,
        random_state=42
)
scaler = StandardScaler()
Xs = scaler.fit_transform(X)
clf.fit(Xs, y)
```

In `scikit-learn`, the regularization strength is controlled by $C = \frac{1}{\lambda}$

C value	Effect	Genomics interpretation
$C = 0.01$	Very strong regularization	Only strongest signals survive
$C = 0.1$	Strong regularization	Aggressive shrinkage
$C = 1.0$	Moderate regularization	Reasonable default
$C = 10$	Weak regularization	Increased risk of overfitting
$C \to \infty$	No regularization	Ill-posed when $p \gg n$

Outline

- The Model
- The Loss Function
- Regularization
- **Optimization**
- Performance Metrics
- Data Leakage
- Nested Cross-validation

The Loss Landscape

Idea 1: Random Search

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in range(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

Idea 2: Follow the Slope

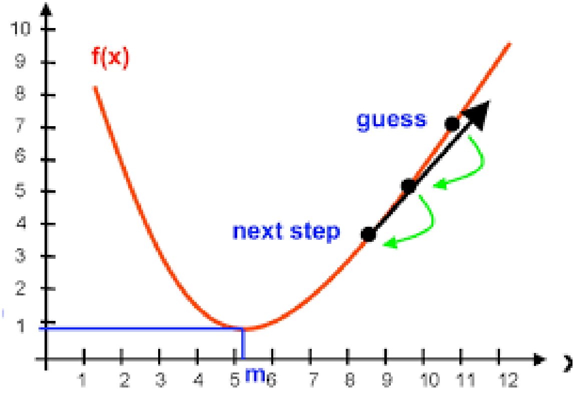
In 1-dimension, the derivative of a function:

$$\frac{\partial f(x)}{\partial x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the gradient is the vector of (partial derivatives) in each dimension:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

Gradient



Finding good Ws

$$
\begin{aligned}
s_i &= f(x_i; W) = W^T x_i \\
p_i &= \frac{1}{1 + e^{-s_i}} \\
\mathcal{L}(w) &= -\left[\sum_{i=1}^{N} y_i \log p_i + (1 - y_i) \log(1 - p_i) \right]
\end{aligned}
$$

want $\nabla_W \mathcal{L}(W)$

Gradients

- Numerical gradient: approximate, slow, easy to write
- Analytical gradient: exact, fast, could be error-prone during derivation

Optimizer: Gradient Descent (batch)

Gradient Descent (batch)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \, \nabla \mathcal{L}(\mathbf{w}_t),$$

where η is the step size (learning rate).

- Stable, smooth descent
- Uses full dataset each step (expensive)
- Good for small datasets or final fine-tuning

```
  # Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Stochastic Gradients Descent (SGD)

Stochastic / Mini-batch GD

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\,\widehat{\nabla}_{\mathcal{B}_t}\mathcal{L}(\mathbf{w}_t)$$

- Cheap updates (one sample or mini-batch)
- Noisy gradients — can help exploration
- Requires learning rate schedules / tuning

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Optimizer: Momentum

Momentum

$$\mathbf{v}_t = \rho\,\mathbf{v}_{t-1} + \nabla\mathcal{L}(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\,\mathbf{v}_t$$

- Accumulates a velocity vector from past gradients
- Dampens oscillations in narrow valleys
- Accelerates convergence along consistent directions

Typical settings:

- Momentum coefficient: Default $\rho = 0.9$
- Learning rate: same scale as SGD

Optimizer: RMSProp

RMSProp (per-parameter adaptive)

$$v_t = \rho\, v_{t-1} + (1-\rho)\, g_t^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \frac{g_t}{\sqrt{v_t} + \varepsilon}$$

- Adaptive per-parameter learning rates
- Typical: $\rho = 0.9, \ \varepsilon = 10^{-8}$

Optimizer: Adam

Adam (momentum + adaptive scaling)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}$$

- Momentum + RMS scaling (fast convergence)
- Defaults: $\beta_1 = 0.9, \ \beta_2 = 0.999, \ \varepsilon = 10^{-8}$
- Often a great default for deep learning

Outline

- The Model
- The Loss Function
- Regularization
- Optimization
- **Performance Metrics**
- Data Leakage
- Nested Cross-validation

Accuracy: Definition and Intuition

Accuracy measures the fraction of correctly classified samples:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- TP: true positives TN: true negatives
- FP: false positives FN: false negatives

Interpretation:

- Probability that a randomly chosen sample is classified correctly
- Simple and intuitive

Why Accuracy Can Be Misleading

Problem: Class imbalance (common in genomics)

Example:

- 95 healthy samples, 5 disease samples
- Classifier predicts "healthy" for everyone

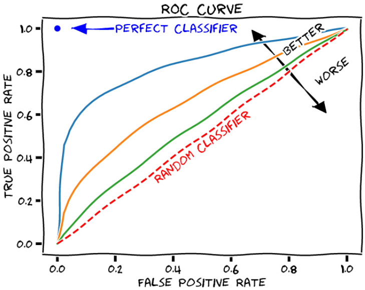
$$\text{Accuracy} = \frac{95}{100} = 95\%$$

But:

- Sensitivity (recall for disease) = 0
- Model is biologically useless

Key takeaway: Accuracy ignores the *decision threshold* and the *cost of errors*.

ROC Curve and ROC-AUC



Most classifiers output a **score or probability**, not just a label. The **ROC curve** plots:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{vs} \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

as the classification threshold varies.

Interpreting ROC-AUC

ROC-AUC = Area Under the ROC Curve

- Ranges from 0.5 (random guessing) to 1.0 (perfect ranking)
- Threshold-independent

Probabilistic interpretation:

$$\text{ROC-AUC} = P\left(\text{score}_{\text{positive}} > \text{score}_{\text{negative}}\right)$$

That is:

- Probability that a randomly chosen disease sample is ranked higher than a randomly chosen control sample

Precision–Recall Curve and PR-AUC (Rare Disease Genomics)

When the positive class is rare (e.g. disease cases), ROC-AUC can be misleading.

The **Precision–Recall (PR) curve** plots:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{vs} \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

as the classification threshold varies.

PR-AUC = Area Under the Precision–Recall Curve

- Focuses on performance for the *positive (rare) class*
- Sensitive to false positives
- Baseline equals prevalence of the positive class

Why PR-AUC Matters for Rare Disease Genomics

Typical setting:

- Disease prevalence: $< 5\%$
- Goal: identify a small set of high-confidence candidates

Key differences vs ROC-AUC:

- ROC-AUC may appear high even with many false positives
- PR-AUC penalizes false positives directly
- Better reflects experimental validation cost

Interpretation:

- High PR-AUC $\Rightarrow$ predicted positives are trustworthy
- Low PR-AUC $\Rightarrow$ many false leads for follow-up

Outline

- The Model
- The Loss Function
- Regularization
- Optimization
- Performance Metrics
- **Data Leakage**
- Nested Cross-validation

Data Leakage: Definition

Data leakage occurs when information from outside the training data is used to build the model.

Formally:

Training procedure has access to information from the test set

Consequences:

- Overly optimistic performance estimates
- Poor generalization to new data
- Irreproducible scientific results

Common Sources of Data Leakage (Genomics)

- Scaling or normalization using the full dataset
- Feature selection performed before cross-validation
- Batch correction using test samples
- Reusing test data for model tuning

Key insight:

> *If the test data influences any step of model construction, performance estimates are invalid.*

Feature Selection Can Cause Leakage

Incorrect workflow:

1. Use all samples to select top k genes
2. Cross-validate classifier on selected genes

Why this is wrong:

- Feature selection uses label information
- Test folds influence which features are chosen

Result: Inflated accuracy, ROC-AUC, PR-AUC

Correct Feature Selection Workflow

Correct principle:

Feature selection must be performed inside the training data only.

Implementation:

- Encapsulate feature selection in a pipeline
- Apply selection separately in each CV fold

This ensures that test data remains truly unseen.

Nested Cross-Validation: Concept

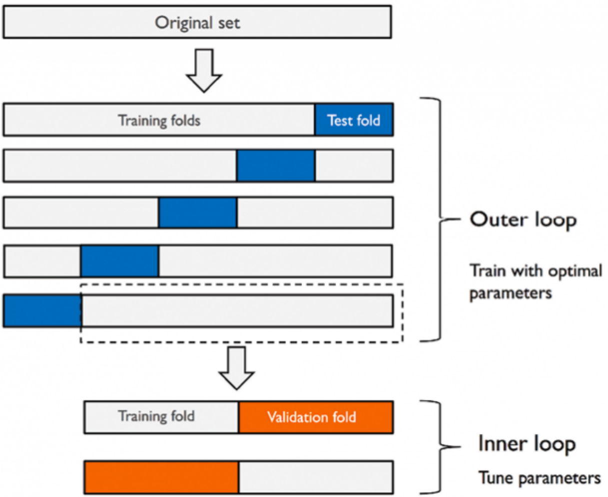
Nested cross-validation separates:

- **Model selection** (hyperparameters, features)
- **Model evaluation** (generalization performance)

Two loops:

- Inner CV: tuning and selection
- Outer CV: unbiased evaluation

Nested Cross-Validation


Original set
Training folds
Test fold
Outer loop
Train with optimal parameters
Training fold
Validation fold
Inner loop
Tune parameters