CS 559 Machine Learning

Lecture 4: Model Selection and Logistic Regression

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Today’s Lecture

- Model Selection
- Evaluation Metrics for Classification
- Probabilistic Generative Models
- Logistic Regression

Model Selection

What is Model Selection?

Given a set of models $M = \{M_1, M_2, \dots, M_R\}$, choose the model that is expected to do the best on the test data. M may consist of:

- Same learning model with different complexities or hyperparameters.
  - Nonlinear regression: polynomials with different degrees
  - K-Nearest Neighbors: Different choices of K
  - Decision Trees: Different choices of the number of levels/leaves
  - SVM: Different choices of the misclassification penalty
  - Regularized models: Different choices of the regularization parameter
  - Kernel based methods: Different choices of kernels ...and almost any learning problem
- Different learning models (e.g. SVM, kNN, DT, etc)
- Note: usually considered in supervised learning but unsupervised learning faces this issue too.

Held-out Data

- Set aside a fraction (10-20%) of the training data.
- This part becomes our held-out data (validation/development)
- Remember: Held-out data is NOT the test data
- Train each model using the remaining training data
- Evaluate error on the held-out data
- Choose the model with the smallest held-out error
- Problems:
  - Wastes training data
  - If there was an unfortunate split, the data distributions may be changed (can be alleviated by repeated random subsampling).

Cross-Validation

- K-fold Cross-Validation on N training examples
- Create K equal sized partitions of the training data
- Each partition has N/K examples
- Train using $K-1$ partitions, validate on the remaining partition
- Repeat the same K times, each with a different validation partition
- Choose the model with the smallest average validation error
- Usually K is chosen as 5 or 10.

Leave-One-Out (LOO) Cross-Validation

- Special case of K-fold Cross-Validation when $K = N$
- Each partition is now an example
- Train using $N - 1$ examples, validate on the remaining example
- Repeat the same N times, each with a different validation example
- Choose the model with the smallest average validation error
- Can be expensive for large N. Typically used when N is small

Random Subsampling Cross-Validation

- Randomly subsample a fixed fraction $\alpha N (0 < \alpha < 1)$ of examples; call it the validation set
- Training using the rest of the examples, measure error on the validation set
- Repeat K times, each with a different randomly chosen validation set
- Choose the model with the smallest average validation error
- Usually α is chosen as 0.1, K as 10

Bootstrapping

- Given a set of N examples
- Idea: Sample N elements from this set with replacement (already sampled elements can be picked again)
- Use this new set as the training data
- The set of examples not selected as the validation data
- For large N, training data consists of about only 63% unique examples
- Expected model error:

$$e = 0.632 e_{test} + 0.368 e_{training}$$

- This can break down if we overfit and $e_{training} = 0$

Evaluation Metrics for Classification

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, interpretability, scalability, etc.
- The contingency table or confusion matrix:

	Actual Class		
Predicted Class		Positive	Negative
	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Metrics for Performance Evaluation

	Actual Class		
Predicted Class		Positive	Negative
	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

- Most widely-used metric: Accuracy= $\frac{TP+TN}{TP+TN+FP+FN}$
- Limitation:
  - For a 2-class problem, $N_0 = 9990, N_1 = 10$.
  - If the model predicts everything to be C_0, the accuracy is $\frac{9990}{10000} = 99.9\%$.
  - This is misleading since the model does not detect any example from C_1.

Cost of Classification

- Consider the cost of misclassification
- Example: cancer vs. non-cancer

Cost Matrix	Actual Class		
Predicted Class		Positive	Negative
	Positive	-1	1
	Negative	100	0

Model 1	Actual Class		
Predicted Class		Positive	Negative
	Positive	150	60
	Negative	40	250

Accuracy=80%
Cost=3910

Model 2	Actual Class		
Predicted Class		Positive	Negative
	Positive	250	5
	Negative	45	200

Accuracy=90%
Cost=4255

Cost-Sensitive Classification

- So far, we haven’t considered costs during training.
- Most learning algorithms do not perform cost-sensitive learning.
- Taking the costs into the training procedure will be an algorithm specific task.
  - Consider the cost when making the predictions and minimize the expected cost.
  - Only predict high-cost class when the model is very confident about the prediction.

Metrics for Performance Evaluation

	Actual Class		
Predicted Class		Positive	Negative
	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

- Recall=$\frac{TP}{TP+FN}$, the fraction of relevant instances that were retrieved.
- Precision=$\frac{TP}{TP+FP}$, the fraction of relevant instances among the retrieved instances.
- F-measure is the harmonic mean between recall and precision.

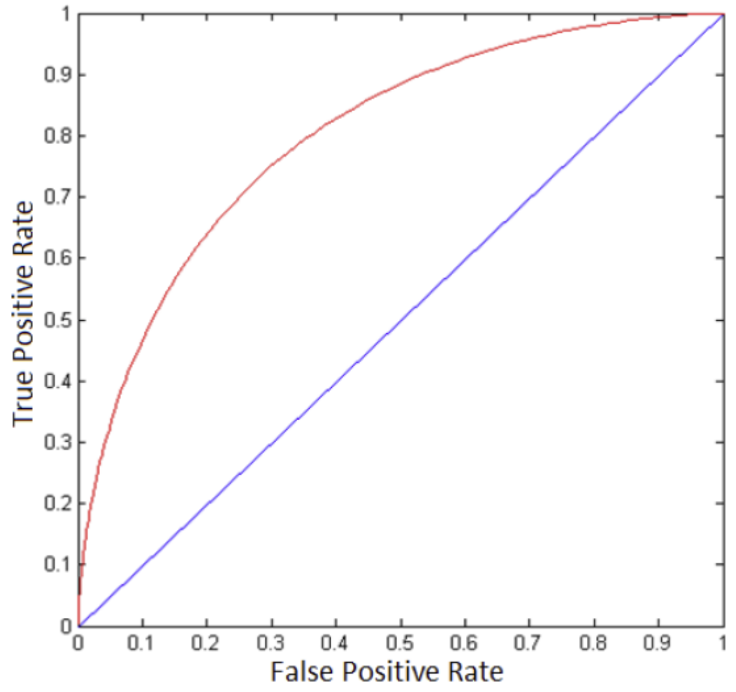
$$F - measure = \frac{2Recall \times Precision}{Recall + Precision} = \frac{2TP}{2TP + FP + FN}$$

ROC Curve

- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.
- This curve plots two parameters:
  - True positive rate on y-axis: $\mathrm{TPR} = \frac{TP}{TP+FN}$
  - False positive rate on x-axis: $\mathrm{FPR} = \frac{FP}{FP+TN}$
- Performance of each classifier represented as a point on the ROC curve.
  - Changing the threshold of algorithm changes the location of the point.
  - Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

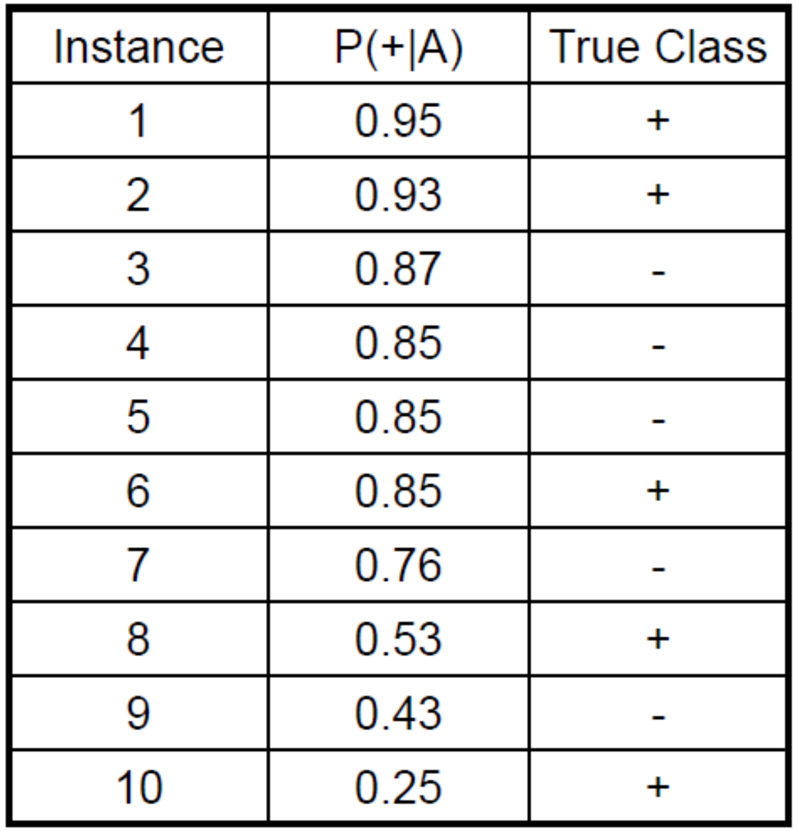
ROC Curve

- (TPR, FPR)
  - (0,0): declare everything to be negative class
  - (1,1): declare everything to be positive class
  - (1,0): Ideal/perfect classification
- Diagonal line represents random guessing. Points above the diagonal represent good classification results.
- AUC: Area Under the ROC Curve
  - Measures the entire two-dimensional area under the ROC curve
  - Provides an aggregate measure of performance across all possible classification thresholds.
  - Ideal: AUC=1; Random guess: AUC =0.5



How to Construct an ROC Curve

1. Use a classifier to produce posterior probability for each test instance P(+|A)
2. Sort the instances according to P(+|A) in decreasing order
3. Apply threshold at each unique value of P(+|A)
4. Count the number of TP, FP, TN, FN at each threshold
5. TPR = TP/(TP+FN); FPR = FP/(FP + TN)



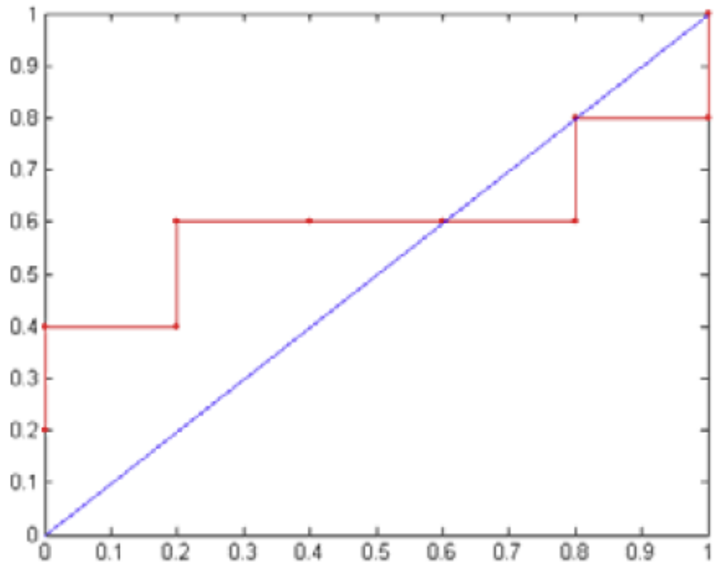
Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

How to Construct an ROC Curve

Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

ROC Curve



Feature Selection

Feature Selection

Selecting a useful subset from all the features. Why?

- Some algorithms scale (computationally) poorly with increased dimension
- Irrelevant features can confuse some algorithms
- Redundant features adversely affect regularization
- Removal of features can increase (relative) margin (and generalization)
- Reduces data set and resulting model size
- Note: Feature Selection is different from Feature Extraction
  - The latter transforms original features to get a small set of new features (we will discuss in dimensionality reduction).

Feature Selection Methods

- Methods agnostic to the learning algorithm
  - Preprocessing based methods
    - E.g., remove a binary feature if it's ON in very few or most examples
  - Filter Feature Selection methods
    - Use some ranking criteria to rank features
    - Select the top-ranking features
- Wrapper Methods (keep the learning algorithm in the loop)
  - Forward/ Backward Search
  - Requires repeated runs of the learning algorithm with different set of features
  - Can be computationally expensive

Filter Feature Selection

- Uses heuristics but is much faster than wrapper methods
- Correlation Criteria: Rank features based on their correlation with the labels

$$R(X_d, y) = \frac{cov(X_d, y)}{\sqrt{var(X_d)var(y)}}$$

- Mutual Information Criteria

$$MI(X_d, y) = \sum_{X_d \in \{0,1\}} \sum_{y \in \{-1,1\}} P(X_d, y) \log \frac{P(X_d, y)}{P(X_d)P(y)}$$

  - High mutual information means high relevance of that feature
  - These probabilities can be easily estimated form the data

Wrapper Methods

- Forward Search
  - Start with no features
  - Greedily include the most relevant feature
  - Stop when selected the desired number of features
- Backward Search
  - Start with all features
  - Greedily remove the least relevant feature
  - Stop when selected the desired number of features
- Inclusion/Removal criteria uses cross-validation

Probabilistic Generative Models

Probabilistic Generative Models

- We now turn to a probabilistic approach to classification.
- Show how models with linear decision boundaries arise from simple assumptions about the distribution of the data.
- Adopt the generative approach: Solve the inference problem of estimating the class-conditional densities $p(x|C_k)$ for each class C_k.
- Infer the prior class probabilities $p(C_k)$.
- Use Bayes' theorem to find the class posterior probabilities:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$

where $p(x) = \sum_k p(x|C_k)p(C_k)$

Probabilistic Generative Approach: Two Classes

- Two-class case:

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \frac{1}{1 + e^{-a}} = \sigma(a)$$

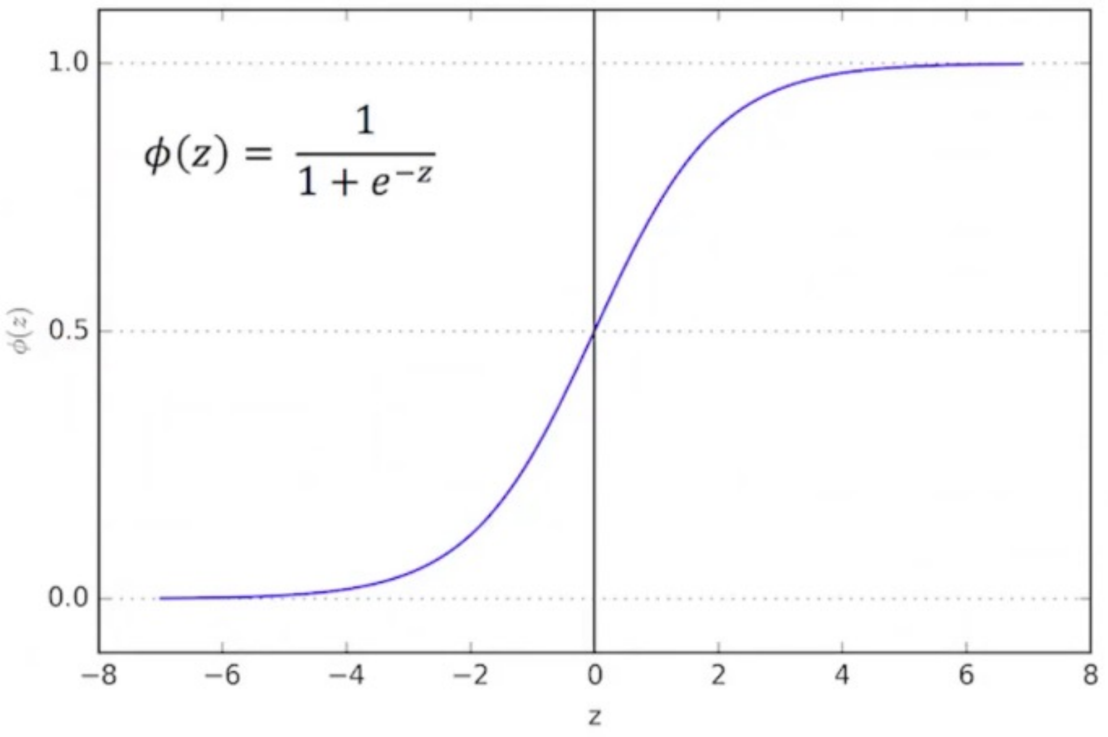
where

$$a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$$

$\sigma(a) = \frac{1}{1+e^{-a}}$ (Logistic Sigmoid Function)

Logistic Sigmoid Function

- It tends to 1 as $z \to \infty$
- It tends to 0 as $z \to -\infty$
- It is bounded between 0 and 1.



Probabilistic Generative Approach: Multiple Classes

- Multiple ($K > 2$) classes case:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_{j=1}^{K} p(x|C_j)p(C_j)} = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

where

$$a_k = \ln p(x|C_k)p(C_k)$$

$$\sigma(a) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$ (softmax function)

Probabilistic Generative Models

- Let's assume the class-conditional densities are Gaussian with the same covariance matrix:

$$p(x|C_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)}$$

- Two-class case: we can show the following result:

$$p(C_1|x) = \sigma(w^T x + \omega_0)$$
$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$
$$\omega_0 = -\frac{1}{2}\mu_1^T \Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T \Sigma^{-1}\mu_2 + ln\frac{p(C_1)}{p(C_2)}$$

- How to get the result?

Probabilistic Generative Models

$$p(C_1|x) = \sigma(a) = \frac{1}{1+e^{-a}} \qquad p(x|C_k) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}e^{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)}$$

$$a = \ln\frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$$

$$= \ln p(x|C_1) - \ln p(x|C_2) + \ln\frac{p(C_1)}{p(C_2)}$$ (Replace $p(x|C_k)$)

$$= -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) + \ln\frac{p(C_1)}{p(C_2)}$$

Probabilistic Generative Models

$$a = -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) + \ln\frac{p(C_1)}{p(C_2)}$$

$$= -\frac{1}{2}x^T\Sigma^{-1}x + \frac{1}{2}\mu_1{}^T\Sigma^{-1}x + \frac{1}{2}x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1$$

$$+\frac{1}{2}x^T\Sigma^{-1}x - \frac{1}{2}\mu_2{}^T\Sigma^{-1}x - \frac{1}{2}x^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_2{}^T\Sigma^{-1}\mu_2 + \ln\frac{p(C_1)}{p(C_2)}$$

$$= \boldsymbol{\mu_1{}^T\Sigma^{-1}x - \mu_2{}^T\Sigma^{-1}x - \frac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2{}^T\Sigma^{-1}\mu_2 + ln\frac{p(C_1)}{p(C_2)}}$$

$$= \underbrace{\boxed{\boldsymbol{(\mu_1-\mu_2)^T\Sigma^{-1}}}}_{w^T}\boldsymbol{x}\underbrace{\boxed{\boldsymbol{-\frac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2{}^T\Sigma^{-1}\mu_2 + ln\frac{p(C_1)}{p(C_2)}}}}_{\omega_0}$$

The quadratic terms in x have cancelled, leading to a linear function of x.

Probabilistic Generative Models

- We have shown:

$$p(C_1|x) = \sigma(w^T x + \omega_0)$$

$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$\omega_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + ln\frac{p(C_1)}{p(C_2)}$$

- Decision boundary is linear in input space:

$$p(C_1|x) = p(C_2|x) = 0.5$$

$$\Rightarrow \frac{1}{1+e^{-(w^T x+\omega_0)}} = 0.5 \;\Rightarrow w^T x + \omega_0 = 0$$

Probabilistic Generative Models

- Multiple ($K > 2$) classes case:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_{j=1}^{K} p(x|C_j)p(C_j)} = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

$$a_k = \ln p(x|C_k)p(C_k)$$

- Decision boundary will occur when two of the posterior probabilities are equal, which will be defined by linear functions of x:

$$p(C_k|x) = \frac{e^{w_k^T x + \omega_{k0}}}{\sum_j e^{w_j^T x + \omega_{j0}}}$$

$$w_k = \Sigma^{-1}\mu_k$$

$$\omega_{k0} = -\frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k + \ln p(C_k)$$

Maximum Likelihood Solution

- Now we have a parametric functional form for the class-conditional densities:

$$p(x|C_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k)}$$

- We can estimate the parameters and the prior class probabilities using maximum likelihood.
  - The case of two classes, each having a Gaussian class-conditional density with a shared covariance matrix.
  - Training data: $\{x_n, y_n\}, n = 1, \dots, N$

$y_n = 1$ denotes class C_1 (N_1 data samples);

$y_n = 0$ denotes class C_2 (N_2 data samples);

Prior class probability: $p(C_1) = \gamma,\ p(C_2) = 1 - \gamma$

Maximum Likelihood Solution

- For a data point x_n from class C_1, we have $y_n = 1$ and therefore:
$$p(x_n, C_1) = p(x|C_1)p(C_1) = \gamma\mathcal{N}(x_n|\mu_1, \Sigma)$$
- For a data point x_n from class C_2, we have $y_n = 0$ and therefore:
$$p(x_n, C_2) = p(x|C_2)p(C_2) = (1-\gamma)\mathcal{N}(x_n|\mu_2, \Sigma)$$
- Assuming observations are drawn independently, the likelihood function is given as follows, where $y = (y_1, \dots, y_N)^T$.
$$p(y|\gamma, \mu_1, \mu_2, \Sigma) = \prod_{\boldsymbol{n=1}}^{\boldsymbol{N}} [p(x_n, C_1)]^{\boldsymbol{y_n}} [p(x_n, C_2)]^{1-y_n}$$
$$= \prod_{\boldsymbol{n=1}}^{\boldsymbol{N}} [\gamma\mathcal{N}(x_n|\mu_1, \Sigma)]^{\boldsymbol{y_n}} [(1-\gamma)\mathcal{N}(x_n|\mu_2, \Sigma)]^{1-y_n}$$

Maximum Likelihood Solution

- We want to find the values of the parameters that maximize the likelihood function, i.e., fit a model that best describes the observed data.

$$p(y|\gamma,\mu_1,\mu_2,\Sigma) = \prod_{n=1}^{N} [\gamma\mathcal{N}(x_n|\mu_1,\Sigma)\,]^{y_n}\,[(1-\gamma)\mathcal{N}(x_n|\mu_2,\Sigma)\,]^{1-y_n}$$

- As usual, we consider the log of the likelihood:

$$\ln p(y|\gamma,\mu_1,\mu_2,\Sigma) = \sum_{n=1}^{N} [y_n \ln\gamma + y_n \ln\mathcal{N}(x_n|\mu_1,\Sigma)$$

$$+(1-y_n)\ln(1-\gamma) + (1-y_n)\ln\mathcal{N}(x_n|\mu_2,\Sigma)]$$

Maximum Likelihood Solution: Parameter γ

$$\ln p(y|\gamma, \mu_1, \mu_2, \Sigma) = \sum_{n=1}^{N} [y_n \ln \gamma + y_n \ln \mathcal{N}(x_n|\mu_1, \Sigma)$$
$$+(1 - y_n) \ln(1 - \gamma) + (1 - y_n) \ln \mathcal{N}(x_n|\mu_2, \Sigma)]$$

- We first maximize the log-likelihood with respect to γ (set derivate to 0).

$$\gamma = \frac{1}{N} \sum_{n=1}^{N} y_n = \frac{N_1}{N} = \frac{N_1}{N_1 + N_2}$$

- The maximum likelihood estimate of γ is the fraction of points in class C_1.
- For multi-class: ML estimate for $p(C_k)$ is given by the fraction of points in the training set in C_k.

Maximum Likelihood Solution: Parameter μ

$$\ln p(D|\gamma, \mu_1, \mu_2, \Sigma) = \sum_{n=1}^{N} [y_n \ln \gamma + y_n \ln \mathcal{N}(x_n|\mu_1, \Sigma)$$
$$+(1 - y_n) \ln(1 - \gamma) + (1 - y_n) \ln \mathcal{N}(x_n|\mu_2, \Sigma)]$$

- We then maximize the log-likelihood with respect to μ_1 (set derivate to 0).

$$\mu_1 = \frac{1}{N_1} \sum_{n=1}^{N} y_n x_n$$

- The maximum likelihood estimate of $\mu_1$is the sample mean of all inputs x_n in class C_1.
- Similarly, the maximum likelihood estimate of μ_2 is given by

$$\mu_2 = \frac{1}{N_2} \sum_{n=1}^{N} (1 - y_n) x_n$$

Maximum Likelihood Solution: Parameter Σ

- Maximize the log-likelihood with respect to the covariance matrix Σ (set derivate to 0), we obtain the estimate Σ_{ML}

$$\Sigma_{ML} = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2$$

where

$$S_1 = \frac{1}{N_1} \sum_{x_n \in C_1} (x_n - \mu_1)(x_n - \mu_1)^T \quad S_2 = \frac{1}{N_2} \sum_{x_n \in C_2} (x_n - \mu_2)(x_n - \mu_2)^T$$

- The maximum likelihood estimate of the covariance is given by the weighted average of the sample covariance matrices associated with each of the classes.
- The results extend to K classes.

Summary So Far

- We assumed $p(x|y = 1) \sim \mathcal{N}(\mu_1, \Sigma)$ and $p(x|y = 0) \sim \mathcal{N}(\mu_2, \Sigma)$, and two class-probabilities $p(y = 1)$ and $p(y = 0)$.
- This is called a generative model, as we have written down a full joint model over the data.
- Violations of the model assumption can lead to "bad" decision boundaries.

Probabilistic Generative Models: Parameters

- How many parameters did we estimate to fit Gaussian class-conditional densities (the generative approach)?
- Suppose d is the dimension of the input space.

$$p(C_1) \Rightarrow 1$$

$$\text{2 mean vectors} \Rightarrow 2d$$

$$\Sigma \Rightarrow d + \frac{d^2 - d}{2} = \frac{d^2 + d}{2}$$

$$Total = 1 + 2d + \frac{d^2 + d}{2} = O(d^2)$$

Logistic Regression

Logistic Regression

$$P(C_1|x) = \sigma(w^T x + \omega_0) = f(x)$$

- We use maximum likelihood to determine the parameters of the logistic regression model.

$$\{x_n, y_n\}, n = 1, \dots, N$$

$$y_n = 1 \text{ denotes class } C_1;\ y_n = 0 \text{ denotes class } C_2$$

- We want to find the values of w that maximize the posterior probabilities associated to the observed data.
- Likelihood function:

$$L(w) = \prod_{n=1}^{N} p(C_1|x_n)^{y_n}(1 - p(C_1|x_n))^{1-y_n}$$

$$= \prod_{n=1}^{N} f(x_n)^{y_n}(1 - f(x_n))^{1-y_n}$$

Logistic Regression

$$P(C_1|x) = \sigma(w^T x + \omega_0) = f(x)$$

- We consider the negative logarithm of the likelihood (Cross Entropy):

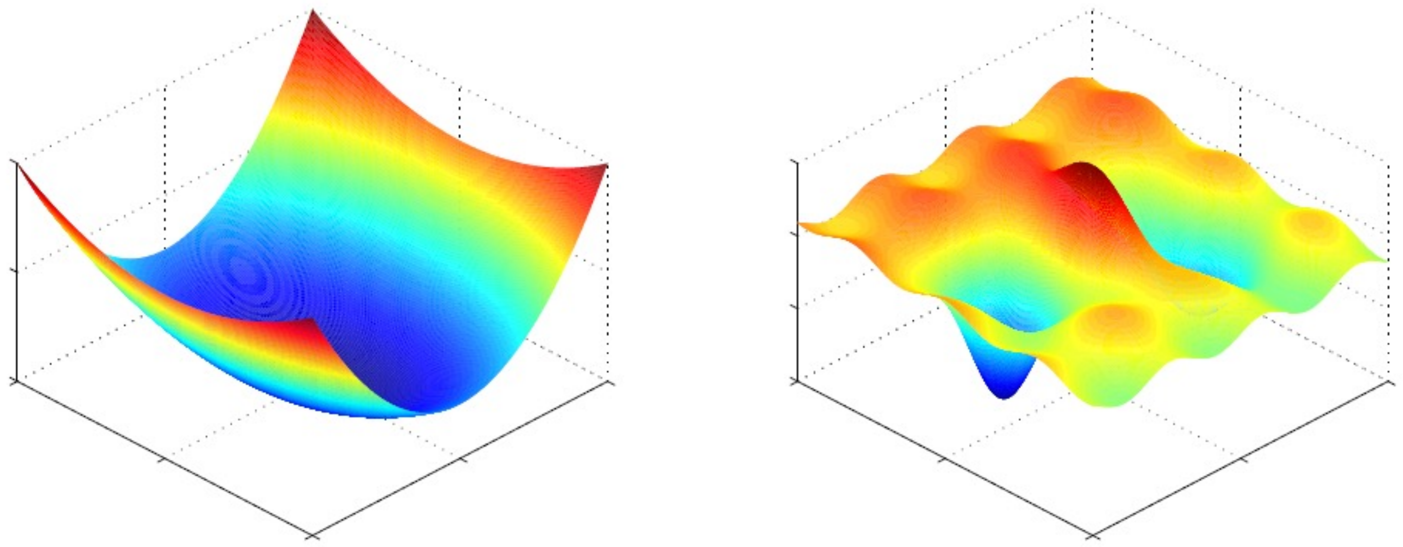
$$\varepsilon(w) = -\ln L(w) = -\ln \prod_{n=1}^{N} p(C_1|x_n)^{y_n}\left(1 - p(C_1|x_n)\right)^{1-y_n}$$

$$= -\sum_{n=1}^{N} (y_n \ln f(x_n) + (1 - y_n) \ln(1 - f(x_n)))$$

- Thus:

$$\max L(w) = \min \varepsilon(w)$$

The Cost-function for Logistic Regression is Convex.



- Fact: The negative log-likelihood is convex - this makes life much easier.
- There are no local minima to get stuck in, and there is good optimization techniques for convex problems.

Review of Gradient Descent Algorithms

- Three important elements: data(x; y), loss function, model parameters w
- One important line, gradient update:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla_w$$

Gradient Descent

```
1: procedure BATCH GRADIENT DESCENT
2:     for $i$ in range(epochs) do
3:         $g^{(i)}(w)$ = evaluate_gradient(TrainLoss, data, w)
4:         $w = w - \text{learning\_rate} * g^{(i)}(w)$
```

Some Limitations:

- Each iteration requires going over all training examples (Batch gradient descent) – expensive and slow when have lots of data.
- Intractable for datasets that don't fit in memory.
- Guaranteed to converge to the global minimum for convex functions, but may end up at a local minimum for non-convex functions.

Stochastic Gradient Descent (SGD)

```
procedure STOCHASTIC GRADIENT DESCENT
    for $i$ in range(epochs) do
        np.random.shuffle(data)
        for example $\in$ data do
            $g^{(i)}(w)$ = evaluate_gradient(loss, example, w)
            $w = w -$ learning_rate $* g^{(i)}(w)$
```

- Shuffling: to ensure that each data point creates an “independent” change on the model, without being biased by the same points before them.
- Easy to fit in memory as only one data point needs to be processed at a time, thus, computationally less expensive.
- With a high variance that cause the objective function to fluctuate heavily.
- May never reach local minima and oscillate around it due to the fluctuations in each step.

Mini-batch Gradient Descent

```
procedure MINI-BATCH GRADIENT DESCENT
    for $i$ in range(epochs) do
        np.random.shuffle(data)
        for batch $\in$ get_batches(data, batch_size=50) do
            $g^{(i)}(w)$ = evaluate_gradient(loss, batch, w)
            $w = w -$ learning_rate $* g^{(i)}(w)$
```

- Instead of using the whole data for calculating gradient, we use only a mini-batch of it (batch size is a hyperparameter).
- Reduces the variance of the parameter updates, which can lead to more stable convergence.
- Can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Logistic Regression: Gradient

$$P(C_1|x) = \sigma(w^T x + \omega_0) = f(x)$$

- We compute the derivative of the error function with respect to w.

$$\frac{\partial \varepsilon(w)}{\partial w} = \frac{\partial}{\partial w}\left[-\ln \prod_{n=1}^{N} p(C_1|x_n)^{y_n}\left(1 - p(C_1|x_n)\right)^{1-y_n}\right]$$

- The derivative of the logistic sigmoid function:

$$\frac{\partial}{\partial a}\sigma(a) = \frac{\partial}{\partial a}\frac{1}{1+e^{-a}} = \frac{e^{-a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\frac{e^{-a}}{(1+e^{-a})}$$
$$= \frac{1}{1+e^{-a}}\left(1 - \frac{1}{1+e^{-a}}\right) = \sigma(a)(1-\sigma(a))$$

Logistic Regression

- The derivative of the error function with respect to w is:

$$\nabla_w \, \varepsilon(w) = \sum_{n=1}^{N} (f(x_n) - y_n) x_n$$

Homework: show the detailed steps to calculate this derivation.

- Then, the parameter can be updated by:

$$w^{t+1} := w^t - \eta \nabla_w \, \varepsilon(w)$$

Probabilistic Discriminative Models: parameters

- Two-class case:

$$P(C_1|x) = \sigma(w^T x + \omega_0) = f(x)$$

$$P(C_2|x) = 1 - P(C_1|x)$$

- This model is known as Logistic Regression.
- Assuming $x \in \mathbb{R}^d$, how many parameters do we need to estimate?

$$d + 1$$

Summary So Far

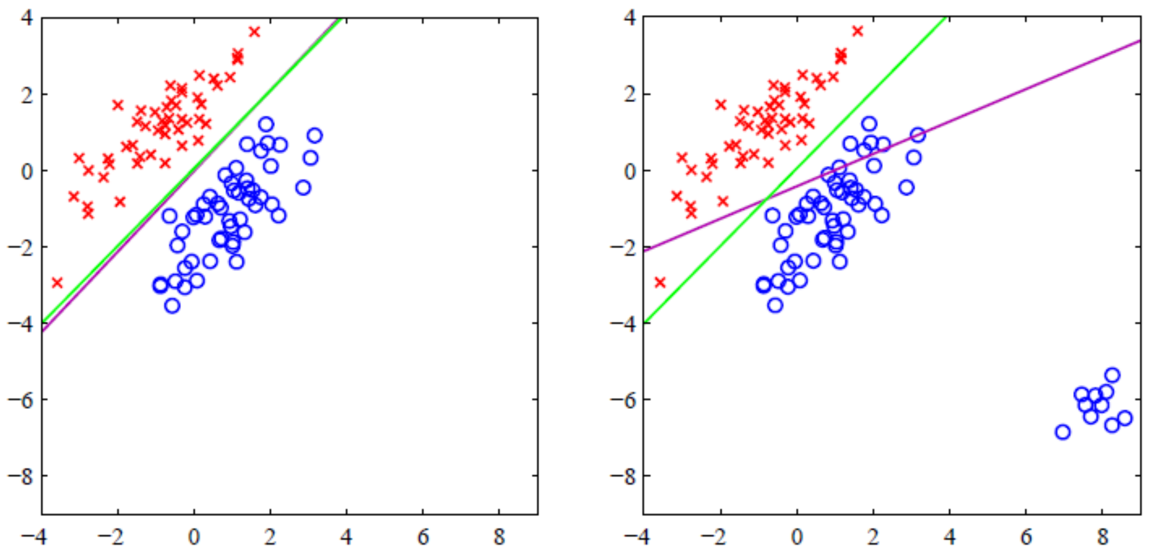
- From the previous introduction, we know that
$$P(y = 1|x) = \sigma(a(x))$$
where $\sigma(a) = 1/(1 + \exp(-a))$ and $a(x) = w^T x + \omega_0$.
- Notation is simpler if we use 0 and 1 as class labels, so we define $y_n = 1$ as the label for the positive class, and $y_n = 0$ a label for the negative class.
- The parameters of $a(x) = w^T x + \omega_0$ can be learnt by minimizing the negative log-likelihood:
$$L(w) = -\sum_n \{y_n \log p(y_n|x_n, w) + (1 - y_n) \log(1 - p(y_n|x_n, w))\}$$

Summary So Far

- This is a discriminative approach to classification, as we only model the labels, and not the inputs.
- Decision rule and function shape of $p(y|x)$ will be the same for the generative and the discriminative model, but the parameters were obtained differently.

Maximum Likelihood Estimation of Logistic Regression

- Logistic regression is a much better algorithm than the algorithms we discussed last week.
- Need to optimize log-likelihood numerically.
- People typically minimize the negative log-likelihood $\mathcal{L}$ rather than maximizing the log-likelihood.
- To numerically minimize the negative log-likelihood, we need its gradient.



Figures from Bishop PRML, 44a and b

Multiclass Logistic Regression

- Multiclass case:

$$p(C_k|x) = \frac{e^{w_k^T x + \omega_{k0}}}{\sum_j e^{w_j^T x + \omega_{j0}}} = f_k(x)$$

- We use maximum likelihood to determine the parameters:

$$\{x_n, y_n\}, n = 1, \dots, N$$

$$y_n = (0, \dots, 1, \dots, 0) \text{ denotes class } C_k$$

- We want to find the values of $w_1, \dots, w_k$ that maximize the posterior probabilities associated to the observed data likelihood function:

$$L(w_1, \dots, w_k) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k|x_n)^{y_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} f_k(x_n)^{y_{nk}}$$

Multiclass Logistic Regression

$$L(w_1, \dots, w_k) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k | x_n)^{y_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} f_k(x_n)^{y_{nk}}$$

- Consider the negative logarithm of the likelihood (cross-entropy error):

$$\varepsilon(w_1, \dots, w_k) = -\ln L(w_1, \dots, w_k) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln f_k(x_n)$$

$$\min_{w_j} \varepsilon\big((w_j)\big)$$

- The gradient of the error function w.r.t one of the parameter vectors:

$$\frac{\partial}{\partial w_j} \varepsilon(w_1, \dots, w_k) = \frac{\partial}{\partial w_j} [-\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln f_k(x_n)]$$

Multiclass Logistic Regression

$$\frac{\partial}{\partial w_j} \varepsilon(w_1, \ldots, w_k) = \frac{\partial}{\partial w_j} [-\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln f_k(x_n)]$$

- The derivatives of the softmax function:

$$\frac{\partial}{\partial a_k} f_k = \frac{\partial}{\partial a_k} \frac{e^{a_k}}{\sum_j e^{a_j}} = \frac{e^{a_k} \sum_j e^{a_j} - e^{a_k} e^{a_k}}{\left(\sum_j e^{a_j}\right)^2} = f_k - f_k^2 = f_k(1 - f_k)$$

- Thus, for $j \neq k$

$$\frac{\partial}{\partial a_j} f_k = \frac{\partial}{\partial a_j} \frac{e^{a_k}}{\sum_j e^{a_j}} = \frac{-e^{a_k} e^{a_j}}{\left(\sum_j e^{a_j}\right)^2} = -f_k f_j$$

- Compact expression (I_{kj} are the elements of the identity matrix)

$$\frac{\partial}{\partial a_j} f_k = f_k(I_{kj} - f_j)$$

Multiclass Logistic Regression

$$\nabla_{w_j}\varepsilon(w_1, \dots, w_k) = \sum_{n=1}^{N}\left(f_{nj} - y_{nj}\right)x_n$$

- It can be shown that ε is a convex function of w. Thus, it has a unique minimum.
- Online solution (SGD):

$$w_j^{t+1} = w_j^t - \eta\nabla_{w_j}\varepsilon_n(w) = w_j^t - \eta\left(f_{nj} - y_{nj}\right)x_n$$

Summary of Today’s Lecture

- Model Selection
- Evaluation Metrics for Classification
- Probabilistic Generative Models
- Logistic Regression