Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**

Machine Learning: Chenhao Tan

University of Colorado Boulder

LECTURE 21: REVIEW SESSION

Precision

Confusion matrix:

		predicted labels	
		positive (1)	negative (0)
true labels	positive (1)	true positive (TP)	false negative (FN)
	negative (0)	false positive (FP)	true negative (TN)

Precision

Confusion matrix:

		predicted labels	
		1	0
true labels	1	TP	FN
	0	FP	TN

Precision measures how accurate the predicted positive class are (exactness).

$$\text{precision} = \frac{TP}{TP + FP}$$

Recall

		predicted labels	
		1	0
true labels	1	TP	FN
	0	FP	TN

Recall measures the fraction of positives that are correctly identified (completeness).

Recall

		predicted labels	
		1	0
true labels	1	TP	FN
	0	FP	TN

Recall measures the fraction of positives that are correctly identified (completeness).

$$\text{recall} = \frac{TP}{TP + FN}$$

F1

F1 score strikes a balance between precision and recall.

$$F1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F1 score of the minority class is usually used when evaluating classifiers on imbalanced datasets.

$F1$ is a special case of $F_\beta = (1 + \beta^2)\frac{\text{precision}\cdot\text{recall}}{\beta^2\cdot\text{precision}+\text{recall}}$.

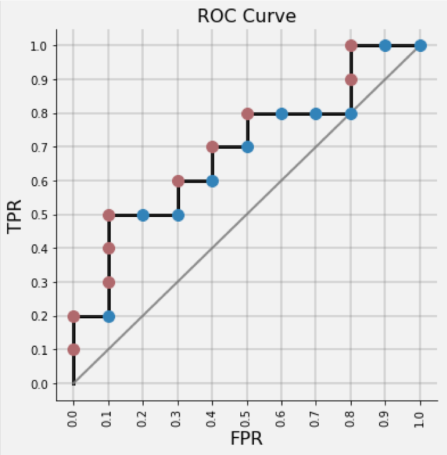
Constructing a ROC curve

You need a classifier that is able to rank examples by predicted score.

- Order all examples by prediction confidence
- Move threshold to each point, one at a time
- If point is true positive, move vertically (1/NP)
- If point is true negative, move horizontally (1/NN)

#	c	$\hat{p}$	#	c	$\hat{p}$
1	P	0.90	11	P	0.40
2	P	0.80	12	N	0.39
3	N	0.70	13	P	0.38
4	P	0.60	14	N	0.37
5	P	0.55	15	N	0.36
6	P	0.54	16	N	0.35
7	N	0.53	17	P	0.34
8	N	0.52	18	P	0.33
9	P	0.51	19	N	0.30
10	N	0.50	20	N	0.10

Constructing a ROC curve



#	c	$\hat{p}$	#	c	$\hat{p}$
1	P	0.90	11	P	0.40
2	P	0.80	12	N	0.39
3	N	0.70	13	P	0.38
4	P	0.60	14	N	0.37
5	P	0.55	15	N	0.36
6	P	0.54	16	N	0.35
7	N	0.53	17	P	0.34
8	N	0.52	18	P	0.33
9	P	0.51	19	N	0.30
10	N	0.50	20	N	0.10

ROC curve

ROC cares both about TPR and FPR, so it values both positive examples and negative examples.
If only positive examples are important, one can plot precision and recall curve.

Multi-class classification

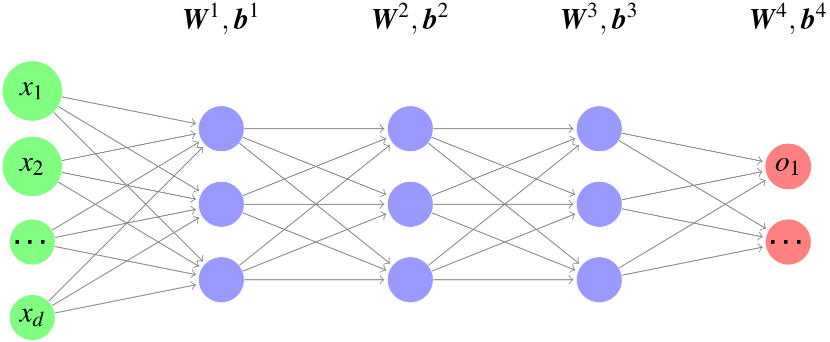
Assume training time is $\mathcal{O}(m^\alpha)$ and test time is $\mathcal{O}(c_t)$

	Training	Testing
One-against-all	$\mathcal{O}(Cm^\alpha)$	$\mathcal{O}(Cc_t)$
All-pairs	$\mathcal{O}\left(C^2\left(\frac{m}{C}\right)^\alpha\right)$	$\mathcal{O}\left(C^2 c_t\right)$

- One-against-all better for testing time
- All-pairs better for training
- All-pairs usually better for performance

Forward propagation algorithm

How do we make predictions based on a multi-layer neural network?
Store the biases for layer l in $\boldsymbol{b}^l$, weight matrix in $\boldsymbol{W}^l$



Forward propagation algorithm

Suppose your network has L layers
Make prediction for an instance $\boldsymbol{x}$

1: Initialize $\boldsymbol{a}^0 = \boldsymbol{x}$
2: **for** $l = 1$ to L **do**
3: $\quad \boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$
4: $\quad \boldsymbol{a}^l = g(\boldsymbol{z}^l)$
5: **end for**
6: The prediction $\hat{y}$ is simply $\boldsymbol{a}^L$

Network architecture

- Network architecture (a lot more then fully connected layers)
  - Convulutional layer
  - Recurrent layer

Back propagation

Back propagation allows for computing the gradients of the parameters and watch out for unstable gradients!

$\delta^L = \frac{\partial \mathscr{L}}{\partial a_j^L} \odot g'(\mathbf{z}^L)$ # Compute δ's on output layer

For $\ell = L, \ldots, 1$

$\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$ # Compute bias derivatives

$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot g'(\mathbf{z}^{\ell-1})$ # Back prop $\boldsymbol{\delta}$'s to previous layer

Practical issues

- Unstable gradients
- Weight initialization
- Dropout
- Batch size

Hard-margin SVM

Primal problem

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2$$

s.t. $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1, i \in [1, m]$

Dual problem

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_j \cdot \boldsymbol{x}_i)$$

s.t. $\alpha_i \geq 0, i \in [1, m]$

$$\sum_i \alpha_i y_i = 0$$

Karush-Kuhn-Tucker (KKT) conditions

Primal and dual feasibility

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1, \alpha_i \geq 0$$

Stationarity

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i, \sum_{i=1}^{m} \alpha_i y_i = 0$$

Complementary slackness

$$\alpha_i = 0 \vee y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$$

Soft-margin SVM

Primal problem

$$\min_{\boldsymbol{w},b,\xi} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1} \xi_i$$

s.t. $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, i \in [1, m]$

$\xi_i \geq 0, i \in [1, m]$

Dual problem

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j (\boldsymbol{x}_j \cdot \boldsymbol{x}_i)$$

s.t. $C \geq \alpha_i \geq 0, i \in [1, m]$

$$\sum_i \alpha_i y_i = 0$$

Karush-Kuhn-Tucker (KKT) conditions

Primal and dual feasibility

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0, C \geq \alpha_i \geq 0, \beta_i \geq 0$$

Stationarity

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i, \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i + \beta_i = C$$

Complementary slackness

$$\alpha_i [y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 + \xi_i] = 0, \beta_i \xi_i = 0$$

Kernels

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \qquad \max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

- Replace all dot product with kernel evaluations $K(x_1, x_2)$
- Makes computation more expensive, overall structure is the same
- K is a Gram matrix