

Multi-Class Classification

Jong-Han Kim

EE787 Fundamentals of machine learning
Kyung Hee University

Multi-class classification

Multi-class classification

- ▶ *classification* is *multi-class* when raw output variable v is a *categorical* $v \in \mathcal{V} = \{v_1, \dots, v_K\}$ with $K > 2$
- ▶ v_i are called *classes* or *labels*
- ▶ we'll also denote them as $1, \dots, K$
- ▶ examples:
 - ▶ $\mathcal{V} = \{\text{YES, MAYBE, NO}\}$
 - ▶ $\mathcal{V} = \{\text{ALBANIA, AZERBAIJAN, ...}\}$
 - ▶ $\mathcal{V} = \{\text{HINDI, TAMIL, ...}\}$
 - ▶ \mathcal{V} = set of English words in some dictionary
 - ▶ \mathcal{V} = set of $m!$ possible orders of m horses in a race
- ▶ a *classifier* predicts label \hat{v} given raw input u
- ▶ called a *K -class classifier*

Confusion matrix

Confusion matrix

- ▶ measure performance of a specific predictor on a data set with n records
- ▶ for each data record i , there are K^2 possible values of (\hat{v}^i, v^i)
- ▶ $K \times K$ *confusion matrix* is defined by

$$C_{ij} = \# \text{ records with } \hat{v} = v_i \text{ and } v = v_j$$

- ▶ entries in C add up to n
- ▶ column sums of C give number of records in each class in the data set
- ▶ C_{ii} is the number of times we predict v_i correctly
- ▶ C_{ij} for $i \neq j$ is the number of times we mistook v_j for v_i
- ▶ there are $K(K - 1)$ different types of errors we can make
- ▶ there are $K(K - 1)$ different *error rates*, C_{ij}/n , $i \neq j$

Neyman-Pearson error

- ▶ $E_j = \sum_{i \neq j} C_{ij}$ is number of times we mistook v_j for another class
- ▶ E_j/n is the error rate of mistaking v_j
- ▶ we will scalarize these K error rates using a weighted sum
- ▶ the *Neyman-Pearson error* is

$$\sum_{j=1}^K \kappa_j E_j = \sum_{i \neq j} \kappa_j C_{ij}$$

where κ is a weight vector with nonnegative entries

- ▶ κ_j is how much we care about mistaking v_j
- ▶ for $\kappa_j = 1$ for all i , Neyman-Pearson error is the *error rate*

Embedding

Embedding v

- ▶ we embed raw output $v \in \mathcal{V}$ into \mathbf{R}^m as $y = \psi(v) \in \mathbf{R}^m$
(cf. boolean classification, where we embed v into \mathbf{R})
- ▶ we can describe ψ by the K vectors $\psi_1 = \psi(v_1), \dots, \psi_K = \psi(v_K)$
(i.e., just say what vector in \mathbf{R}^m each $v \in \mathcal{V}$ maps to)
- ▶ we call the vector ψ_i the *representative* of v_i
- ▶ we call the set $\{\psi_1, \dots, \psi_K\}$ the *constellation*
- ▶ examples:
 - ▶ $\text{TRUE} \mapsto 1, \text{FALSE} \mapsto -1$
 - ▶ $\text{YES} \mapsto 1, \text{MAYBE} \mapsto 0, \text{NO} \mapsto -1$
 - ▶ $\text{YES} \mapsto (1, 0), \text{MAYBE} \mapsto (0, 0), \text{NO} \mapsto (0, 1)$
 - ▶ $\text{APPLE} \mapsto (1, 0, 0), \text{ORANGE} \mapsto (0, 1, 0), \text{BANANA} \mapsto (0, 0, 1)$
 - ▶ $(\text{Horse 3}, \text{Horse 1}, \text{Horse 2}) \mapsto (3, 1, 2)$
 - ▶ `word2vec` (maps 1M words to vectors in \mathbf{R}^{300})

One-hot embedding

- ▶ a simple generic embedding of K classes into \mathbf{R}^K
- ▶ $\psi(v_i) = \psi_i = e_i$
- ▶ variation (embedding K classes into \mathbf{R}^{K-1}):
 - ▶ choose one of the classes as the *default*, and map it to $0 \in \mathbf{R}^{K-1}$
 - ▶ map the others to the unit vectors $e_1, \dots, e_{K-1} \in \mathbf{R}^{K-1}$

Nearest neighbor un-embedding

- ▶ given prediction $\hat{y} \in \mathbf{R}^m$, we *un-embed* to get \hat{v}
- ▶ we denote our un-embedding using the symbol $\psi^\dagger : \mathbf{R}^m \rightarrow \mathcal{V}$
- ▶ we *define* the un-embedding function ψ^\dagger as

$$\psi^\dagger(\hat{y}) = \operatorname{argmin}_{v \in \mathcal{V}} \|\hat{y} - \psi(v)\|$$

(we can break ties any way we like)

- ▶ *i.e.*, we choose the raw value associated with the nearest representative
- ▶ called *nearest neighbor un-embedding*

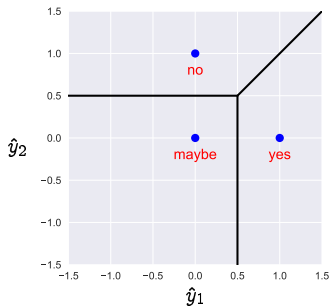
Un-embedding boolean

► embed $\text{TRUE} \mapsto 1 = \psi_1$ and $\text{FALSE} \mapsto -1 = \psi_2$

► un-embed via

$$\psi^\dagger(\hat{y}) = \begin{cases} \text{TRUE} & \hat{y} \geq 0 \\ \text{FALSE} & \hat{y} < 0 \end{cases}$$

Un-embedding yes, maybe, no



► embed YES $\mapsto (1, 0)$, MAYBE $\mapsto (0, 0)$, NO $\mapsto (0, 1)$

► un-embed via

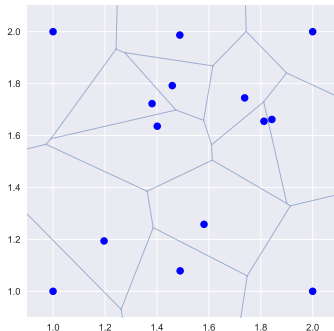
$$\psi^\dagger(\hat{y}) = \begin{cases} \text{YES} & \hat{y}_1 > 1/2, \hat{y}_1 > \hat{y}_2 \\ \text{MAYBE} & \hat{y}_1 < 1/2, \hat{y}_2 < 1/2 \\ \text{NO} & \hat{y}_2 > 1/2, \hat{y}_1 < \hat{y}_2 \end{cases}$$

(can choose any value on boundaries)

Un-embedding one-hot

- ▶ one-hot embedding: $\psi_i = e_i, i = 1, \dots, K$
- ▶ un-embed via $\psi^\dagger(y) = \operatorname{argmin}_i \|y - e_i\|_2 = \operatorname{argmax}_i y_i$
- ▶ intuition:
 - ▶ you can subtract one from one component of a vector
 - ▶ to get the smallest norm
 - ▶ best choice is the largest entry of the vector

Voronoi diagram



- ▶ ψ^\dagger *partitions* \mathbf{R}^m into the K regions $\{y \mid \psi^\dagger(y) = v_i\}$, for $i = 1, \dots, K$
- ▶ regions are *polyhedra*
- ▶ called *Voronoi diagram*
- ▶ boundaries between regions are perpendicular bisectors between pairs of representatives ψ_i, ψ_j

Margins

Margins and decision boundaries

- ▶ given prediction $\hat{y} \in \mathbf{R}^m$, we un-embed via $\hat{v} = \psi^\dagger(\hat{y})$
- ▶ $\psi^\dagger(\hat{y}) = v_j$ when \hat{y} is closer to ψ_j than the other representatives, i.e.,

$$\|\hat{y} - \psi_j\| < \|\hat{y} - \psi_i\| \text{ for } i \neq j$$

- ▶ define the *negative margin* function M_{ij} by

$$\begin{aligned} M_{ij}(\hat{y}) &= (\|\hat{y} - \psi_j\|^2 - \|\hat{y} - \psi_i\|^2) / (2\|\psi_i - \psi_j\|) \\ &= \frac{2(\psi_i - \psi_j)^\top \hat{y} + \|\psi_j\|^2 - \|\psi_i\|^2}{2\|\psi_i - \psi_j\|} \end{aligned}$$

- ▶ so $\psi^\dagger(\hat{y}) = v_j$ when $M_{ij}(\hat{y}) < 0$ for all $i \neq j$

Margins and decision boundaries

- ▶ linear equation

$$M_{ij}(\hat{y}) = 0$$

defines a *hyperplane* called the *perpendicular bisector* between ψ_i and ψ_j

- ▶ it is the *decision boundary* between ψ_i and ψ_j
- ▶ \hat{y} is the correct prediction, when $v = v_j$, if

$$\max_{i \neq j} M_{ij}(\hat{y}) < 0$$

Margins and decision boundaries

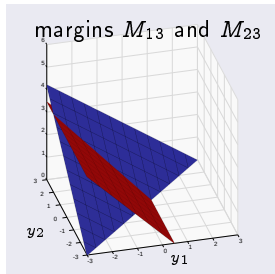
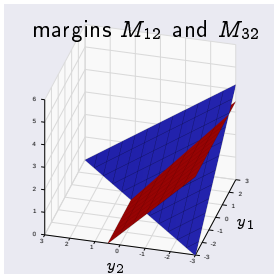
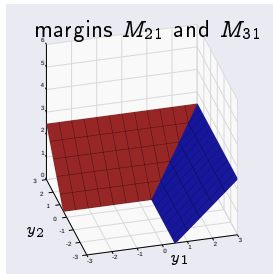
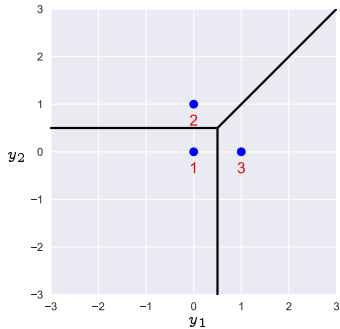
- ▶ boolean: $\psi_1 = -1$ and $\psi_2 = 1$ and

$$M_{21}(\hat{y}) = \hat{y} \quad M_{12}(\hat{y}) = -\hat{y}$$

- ▶ one-hot: $\psi_j = e_j$ for all j , so

$$M_{ij} = \frac{y_i - y_j}{\sqrt{2}}$$

Margins



Vector ERM

Vector prediction

- ▶ after embedding raw data u and v we have data pair (x, y)
- ▶ the target y is a *vector* (which takes only the values ψ_1, \dots, ψ_K)
- ▶ predictor is a function $g : \mathbf{R}^d \rightarrow \mathbf{R}^m$
- ▶ our final (raw) prediction is $\hat{v} = \psi^\dagger(\hat{y})$

Vector linear predictor

- ▶ *vector linear predictor* has form $\hat{y} = g(x) = \theta^\top x$
- ▶ same form as when y is a scalar, but here θ is a $d \times m$ *parameter matrix*
- ▶ θ_{23} is how much x_2 affects \hat{y}_3
- ▶ reduces to the usual parameter vector when $m = 1$ (i.e., y is scalar)

Vector ERM

- ▶ linear model $\hat{y} = \theta^\top x$, $\theta \in \mathbf{R}^{d \times m}$
- ▶ choose parameter matrix θ to minimize $\mathcal{L}(\theta) + \lambda r(\theta)$
- ▶ $\mathcal{L}(\theta)$ is the empirical risk

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(\hat{y}^i, y^i) = \frac{1}{n} \sum_{i=1}^n \ell(\theta^\top x^i, y^i)$$

with loss function $\ell : \mathbf{R}^m \times \mathbf{R}^m \rightarrow \mathbf{R}$ (i.e., ℓ takes two arguments, each in \mathbf{R}^m)

- ▶ $\lambda \geq 0$ is regularization parameter
- ▶ $r(\theta)$ is the regularizer

Derivative of the empirical risk

- ▶ loss $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(\theta^\top x^i, y^i)$
- ▶ we'd like to apply the gradient method
- ▶ $D\mathcal{L}(\theta)$ is the derivative of \mathcal{L} with respect to θ (a matrix)
- ▶ we have

$$(D\mathcal{L}(\theta))_{ij} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta_{ij}}$$

- ▶ then the first-order Taylor approximation is

$$\mathcal{L}(\theta + \delta\theta) \approx \mathcal{L}(\theta) + \text{trace}(D\mathcal{L}(\theta)^\top \delta\theta)$$

- ▶ we have

$$D\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n x^i (\nabla_1 \ell(\theta^\top x^i, y^i))^\top$$

where ∇_1 means the gradient with respect to the first argument

Matrix regularizers

Matrix regularizers

- ▶ general penalty regularizer: $r(\theta) = \sum_{i=1}^d \sum_{j=1}^m q(\theta_{ij})$
- ▶ sum square regularizer: $r(\theta) = \|\theta\|_F^2 = \sum_{i=1}^d \sum_{j=1}^m \theta_{ij}^2$
- ▶ the *Frobenius norm* of a matrix θ is $\left(\sum_{i,j} \theta_{ij}^2\right)^{1/2}$
- ▶ sum absolute or ℓ_1 regularizer: $r(\theta) = \|\theta\|_1 = \sum_{i=1}^d \sum_{j=1}^m |\theta_{ij}|$

Multi-class loss functions

Multi-class loss functions

- ▶ $\ell(\hat{y}, y)$ is how much prediction \hat{y} bothers us when observed value is y
- ▶ but the only possible values of y are ψ_1, \dots, ψ_K
- ▶ so we can simply give the K functions of \hat{y}

$$\ell(\hat{y}, \psi_j), \quad j = 1, \dots, K$$

- ▶ $\ell(\hat{y}, \psi_j)$ is how much we dislike predicting \hat{y} when $y = \psi_j$

Neyman-Pearson loss

- ▶ Neyman-Pearson loss is

$$\ell^{\text{NP}}(\hat{y}, \psi_j) = \begin{cases} 0 & \text{if } \max_{i \neq j} M_{ij} < 0 \\ \kappa_j & \text{otherwise} \end{cases}$$

- ▶ Neyman-Pearson risk $\mathcal{L}^{\text{NP}}(\theta)$ is the Neyman-Pearson error
- ▶ but $\nabla \mathcal{L}^{\text{NP}}(\theta)$ is either zero or undefined
- ▶ so there's no gradient to tell us which way to change θ to reduce $\mathcal{L}(\theta)$

Proxy loss

- ▶ we will use a *proxy loss* that
 - ▶ approximates, or at least captures the flavor of, the Neyman-Pearson loss
 - ▶ is more easily optimized (e.g., is convex or has nonzero derivative)
- ▶ we want a proxy loss function
 - ▶ with $\ell(\hat{y}, \psi_j)$ small whenever $M_{ij} < 0$ for $i \neq j$
 - ▶ and not small otherwise
 - ▶ which has other nice characteristics, e.g., differentiable or convex

Multi-class hinge loss

- ▶ *hinge loss* is

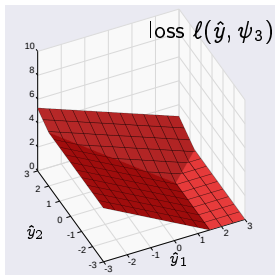
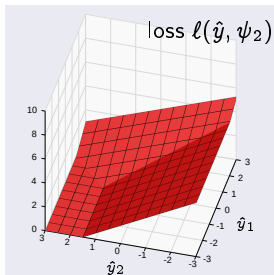
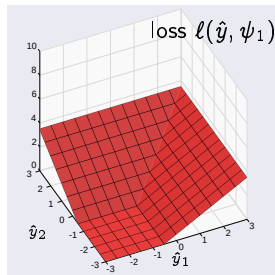
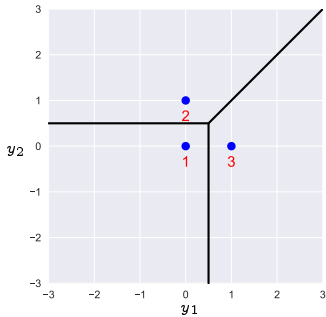
$$\ell(\hat{y}, \psi_j) = \kappa_j \max_{i \neq j} (1 + M_{ij}(\hat{y}))_+$$

- ▶ $\ell(\hat{y}, \psi_j)$ is zero when the correct prediction is made, with a margin at least one
- ▶ convex but not differentiable
- ▶ for boolean embedding with $\psi_1 = -1$, $\psi_2 = 1$, reduces to

$$\ell(\hat{y}, -1) = \kappa_1 (1 + \hat{y})_+, \quad \ell(\hat{y}, 1) = \kappa_2 (1 - \hat{y})_+$$

usual hinge loss when $\kappa_1 = 1$

Multi-class hinge loss



Multi-class logistic loss

- ▶ *logistic loss* is

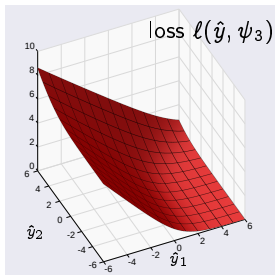
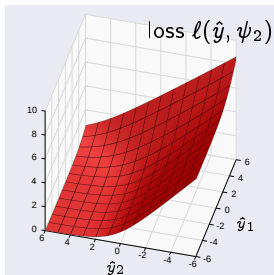
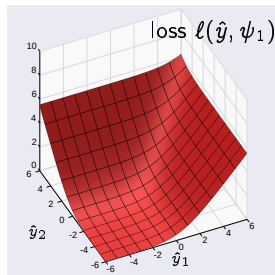
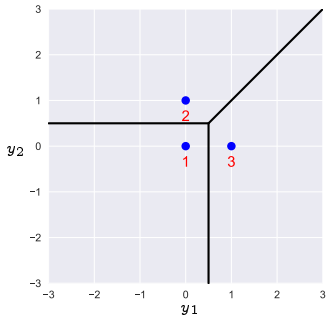
$$\ell(\hat{y}, \psi_j) = \kappa_j \log \left(\sum_{i=1}^K \exp(M_{ij}(\hat{y})) \right)$$

- ▶ recall that $M_{jj} = 0$
- ▶ convex and differentiable
- ▶ for boolean embedding with $\psi_1 = -1$, $\psi_2 = 1$, reduces to

$$\ell(\hat{y}, -1) = \kappa_1 \log(1 + e^{\hat{y}}), \quad \ell(\hat{y}, 1) = \kappa_2 \log(1 + e^{-\hat{y}})$$

usual logistic loss when $\kappa_1 = 1$

Multi-class logistic loss



Soft-max function

- ▶ the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$

$$f(x) = \log \sum_{i=1}^n \exp(x_i)$$

is called the *log-sum-exp* function

- ▶ it is a convex differentiable approximation to the max function
- ▶ we have

$$\max\{x_1, \dots, x_n\} \leq f(x) \leq \max\{x_1, \dots, x_n\} + \log(n)$$

Example: Iris

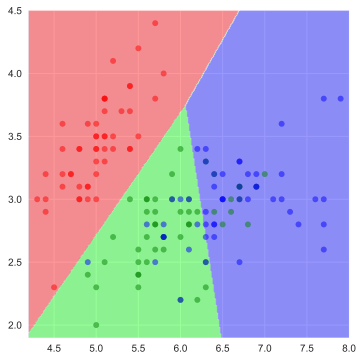
Example: Iris

- ▶ famous example dataset by Fisher, 1936
- ▶ measurements of 150 plants, 50 from each of 3 species
- ▶ iris setosa, iris versicolor, iris virginica
- ▶ four measurements: sepal length, sepal width, petal length, petal width

Example: Iris



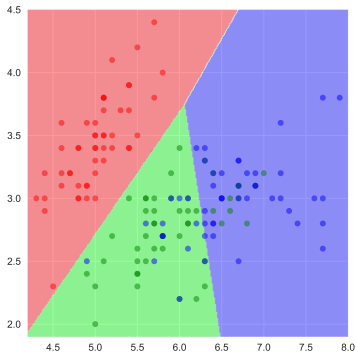
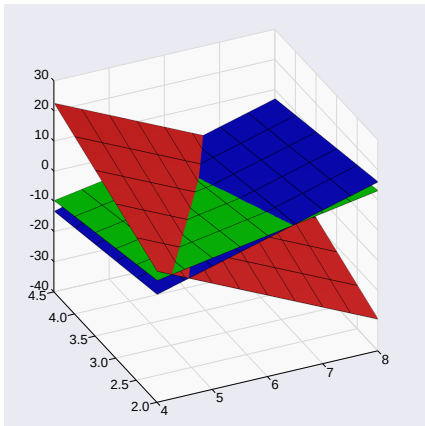
Classification with two features



- ▶ using only `sepal_length` and `sepal_width`
- ▶ one-hot embedding, multi-class logistic loss

▶ confusion matrix $C = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 38 & 13 \\ 0 & 12 & 37 \end{bmatrix}$

Classification with two features



- ▶ let θ_i be the i th column of θ
- ▶ plot shows $\theta_i^T \phi(u)$ as function of u
- ▶ one-hot embedding of v , so un-embedding is $\hat{v} = \arg \max_i \theta_i^T x$

Example: Iris confusion matrix

- ▶ we train using multi-class logistic loss, with $\kappa_i =$ for all i
- ▶ for this example, train using all the data
- ▶ resulting confusion matrix is

$$C = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 49 & 1 \\ 0 & 1 & 49 \end{bmatrix}$$