Fitting Classifiers by Empirical Risk Minimization

Sanjay Lall and Stephen Boyd

EE104
Stanford University

Embedding and un-embedding a categorical

Embedding the categorical output v

- we embed raw output $v \in \mathcal{V}$ into $\mathbf{R}^m$ as $y = \psi(v) \in \mathbf{R}^m$
- we can describe ψ by the K vectors $\psi_1 = \psi(v_1), \ldots, \psi_K = \psi(v_K)$ (*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 $\{\psi_1, \ldots, \psi_K\}$ the *constellation*
- examples:
  - $\textsc{true} \mapsto 1$, $\textsc{false} \mapsto -1$
  - $\textsc{true} \mapsto 1$, $\textsc{false} \mapsto 0$
  - $\textsc{yes} \mapsto 1$, $\textsc{maybe} \mapsto 0$ $\textsc{no} \mapsto -1$
  - $\textsc{yes} \mapsto (1,0)$, $\textsc{maybe} \mapsto (0,0)$, $\textsc{no} \mapsto (0,1)$
  - $\textsc{apple} \mapsto (1,0,0)$, $\textsc{orange} \mapsto (0,1,0)$, $\textsc{banana} \mapsto (0,0,1)$
  - (Horse 3, Horse 1, Horse 2) $\mapsto (3,1,2)$
  - `word2vec` (maps 1M words to vectors in $\mathbf{R}^{300}$)

One-hot and reduced one-hot embedding

- one-hot embedding of K classes into $\mathbf{R}^K$: $\psi(v_i) = \psi_i = e_i$
- *e.g.*, for Booleans: $\psi_1 = (1, 0)$, $\psi_2 = (0, 1)$
- reduced one-hot embedding 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 $e_1, \ldots, e_{K-1} \in \mathbf{R}^{K-1}$
- for Booleans:
  - one-hot embedding is $\psi_1 = (1, 0)$, $\psi_2 = (0, 1)$
  - reduced one-hot embedding is $\psi_1 = 0$, $\psi_2 = 1$
- example: $\mathcal{V} = \{\text{MAYBE}, \text{YES}, \text{NO}\}$, with default MAYBE
- reduced one-hot embedding is $\psi(\text{MAYBE}) = (0, 0)$, $\psi(\text{YES}) = (1, 0)$, $\psi(\text{NO}) = (0, 1)$

Classifying by un-embedding a prediction

- embed raw input to feature vector as $x = \phi(u) \in \mathbf{R}^d$
- embed raw output to representative as $y = \psi(v) \in \mathbf{R}^m$
- create predictor $g : \mathbf{R}^d \to \mathbf{R}^m$ with $\hat{y} = g(x)$
- we hope that $\hat{y} = g(x) \approx y = \psi(v)$ ($\hat{y}$ and y are vectors, so this means $||\hat{y} - y||_2$ small)
- to get the prediction, we *un-embed* $\hat{y}$ to get $\hat{v}$: $\hat{v} = \psi^\dagger(\hat{y})$
- $\psi^\dagger : \mathbf{R}^m \to \mathcal{V}$ is the *un-embedding function*
- the final classifier has the form $\hat{v} = G(u) = \psi^\dagger(g(\phi(u)))$
- can write as $G = \psi^\dagger \circ g \circ \phi$
- in words: *embed*; *predict*; *un-embed*

Nearest neighbor un-embedding

- given prediction $\hat{y} \in \mathbf{R}^m$, we *un-embed* to get $\hat{v}$
- we denote our un-emdedding using the symbol $\psi^\dagger : \mathbf{R}^m \to \mathcal{V}$
- we will use *nearest neighbor un-embedding*:

$$\psi^\dagger(\hat{y}) = \underset{v \in \mathcal{V}}{\mathrm{argmin}} \; ||\hat{y} - \psi(v)||_2$$

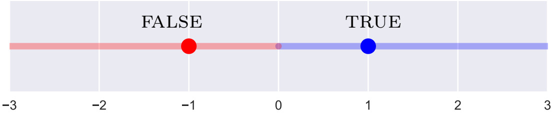
(we can break ties any way we like)

- *i.e.*, we choose the raw value associated with the nearest representative to $\hat{y}$

Un-embedding Boolean

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

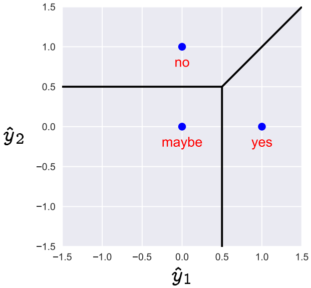
$$\psi^\dagger(\hat{y}) = \begin{cases} \text{TRUE} & \hat{y} \geq 0 \\ \text{FALSE} & \hat{y} < 0 \end{cases}$$



Un-embedding one-hot

- take $\mathcal{V} = \{1, \ldots, K\}$
- one-hot embedding: $\psi_i = e_i$, $i = 1, \ldots, K$
- un-embed via $\psi^\dagger(\hat{y}) = \mathbf{argmin}_i \, ||y - e_i||_2$
- can be expressed as $\psi^\dagger(\hat{y}) = \mathbf{argmax}_i \, \hat{y}_i$
- *i.e.*, we guess class associated with the largest entry in $\hat{y}$
- reason:
  - $||\hat{y} - e_i||_2^2 = ||\hat{y}||_2^2 + 1 - 2\hat{y}^T e_i = ||\hat{y}||_2^2 + 1 - 2\hat{y}_i$
  - first two terms don't depend on i, so we just choose i to maximize $\hat{y}_i$

Un-embedding yes, maybe, no

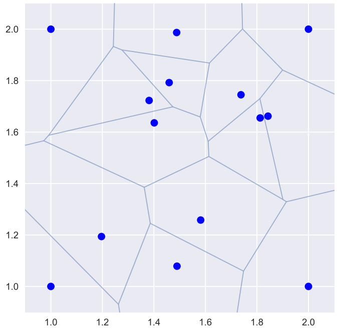


- embed $\textsc{yes} \mapsto (1,0)$, $\textsc{maybe} \mapsto (0,0)$, $\textsc{no} \mapsto (0,1)$ (reduced one-hot)
- un-embed via

$$\psi^\dagger(\hat{y}) = \begin{cases} \textsc{yes} & \hat{y}_1 > 1/2,\ \hat{y}_1 > \hat{y}_2 \\ \textsc{maybe} & \hat{y}_1 < 1/2,\ \hat{y}_2 < 1/2 \\ \textsc{no} & \hat{y}_2 > 1/2,\ \hat{y}_1 < \hat{y}_2 \end{cases}$$

(can choose any value on boundaries)

Voronoi diagram



- $\psi^\dagger$ *partitions* $\mathbf{R}^m$ into the K regions $\{y \mid \psi^\dagger(y) = v_i\}$, for $i = 1, \ldots, K$
- regions are *polyhedra* (of points closer to one representative than all others)
- called *Voronoi diagram*
- boundaries between regions are perpendicular bisectors between pairs of representatives ψ_i, ψ_j

Loss function and empirical risk

Parametrized predictor

- we use parametrized predictor $g_\theta : \mathbf{R}^d \to \mathbf{R}^m$
- θ is a parameter that we can choose
- predictor g_θ gives classifier $\hat{v} = G(u) = \psi^\dagger(g_\theta(\psi(u)))$

- we'll choose θ using ERM and a training data set
- we validate the predictor by performance metric on a test data set

Examples of parametrized predictors for classification

- tree-based predictor (called a *classification tree*)
  - θ encodes tree, feature to split at each node, threshold, leaf values
  - each leaf has a value of $\hat{y}$

- neural network
  - θ gives offset and weights in the different layers
  - $\hat{y}$ is output of last layer

- linear predictor
  - θ is a $d \times m$ *parameter matrix*
  - $\hat{y} = g_\theta(x) = \theta^\mathsf{T} x$

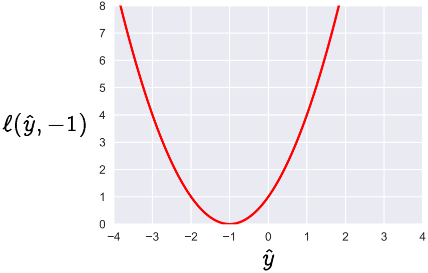
Loss function for classifiers

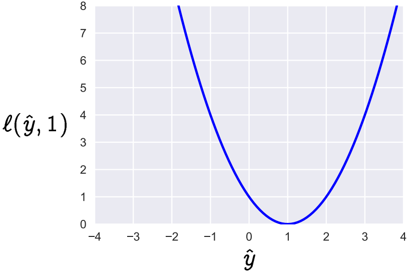
- we use a loss function $\ell : \mathbf{R}^m \times \mathcal{V} \to \mathbf{R}$
- $\ell(\hat{y}, y)$ is how much prediction $\hat{y} \in \mathbf{R}^m$ bothers us when observed value is $y \in \{\psi_1, \ldots, \psi_K\}$
- the only possible values of y are $\psi_1, \ldots, \psi_K$, so we can simply give the K functions of $\hat{y}$

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

- $\ell(\hat{y}, \psi_j)$ is how much we dislike predicting $\hat{y}$ when $y = \psi_j$
- typically $\ell(\hat{y}, \psi_j)$ is nonnegative, and small when $\hat{y} \approx \psi_j$
- *square loss*: $\ell(\hat{y}, \psi_j) = ||\hat{y} - \psi_j||_2^2$
- we'll see far better loss functions for classifiers later

Square loss for Boolean classification





ERM and RERM

- we are given a training data set $x^1, \dots, x^n$, $y^1, \dots, y^n$, and a parametrized predictor g_θ
- empirical risk associated with loss function ℓ is

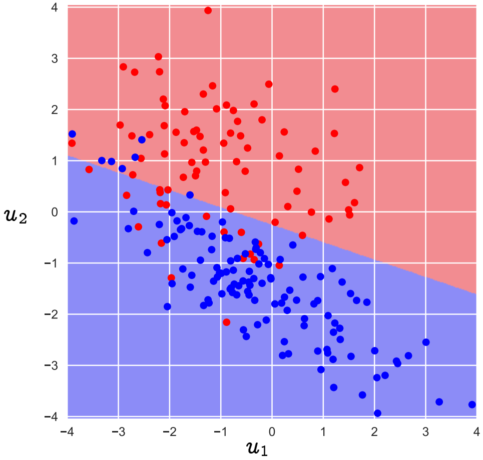
$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{y}^i, y^i) = \frac{1}{n} \sum_{i=1}^{n} \ell(g_\theta(x^i), y^i)$$

- ERM: choose θ to minimize $\mathcal{L}(\theta)$
- in most cases, we need to resort to numerical optimization to find θ
- regularized ERM: choose θ to minimize $\mathcal{L}(\theta) + \lambda r(\theta)$
- r is the regularizer and $\lambda > 0$ is the regularization hyper-parameter

Least squares classifier

- linear predictor $\hat{y} = \theta^\mathsf{T} x$
- square loss $\ell(\hat{y}, \psi_j) = ||\hat{y} - \psi_j||_2^2$
- square regularizer $r(\theta) = ||\theta||_F^2$
- called *least squares classifier*
- can solve RERM problem exactly using least squares
- we'll see better losses for classifiers later

Example



- $u \in \mathbf{R}^2$, embedded as $x = (1, u_1, u_2)$; $v \in \{-1, 1\}$, embedded as $y = v$
- square loss and regularizer

ERM for Neyman-Pearson metric

Neyman-Pearson meetric

- suppose we care about the Neyman-Pearson metric, $\sum_{j=1}^{K} \kappa_j E_j$
- E_j is rate of mistaking v_j for some other class; κ is a weight vector
- κ_j is how much we care about mistaking v_j, relative to others
- to reflect different costs for different errors, we scale the losses by κ_i

- if $\tilde{\ell}(\hat{y}, \psi_j)$, $j = 1, \ldots K$ are the unweighted losses, we use

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

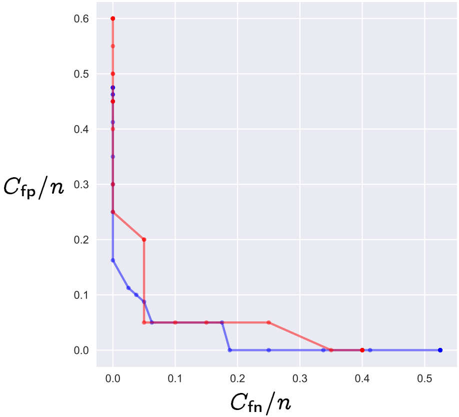
Example

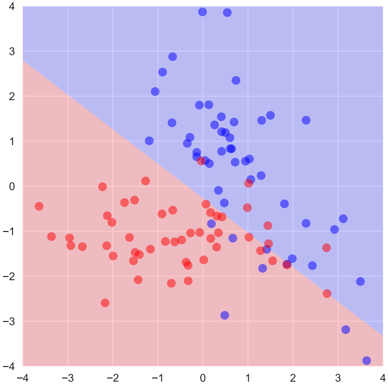
- Boolean classifier, with $\psi_1 = -1$, $\psi_2 = 1$
- we care about Neyman-Pearson metric, $\kappa E_{\mathsf{fn}} + E_{\mathsf{fp}}$
- $\kappa > 0$ is how much we care about false negatives relative to false positive
- we use loss function

  $$\ell(\hat{y}, y) = \begin{cases} (\hat{y} - y)^2 & \text{if } y = -1 \\ \kappa(\hat{y} - y)^2 & \text{otherwise} \end{cases}$$

  which gives more weight to deviating from the positive representative ψ_2

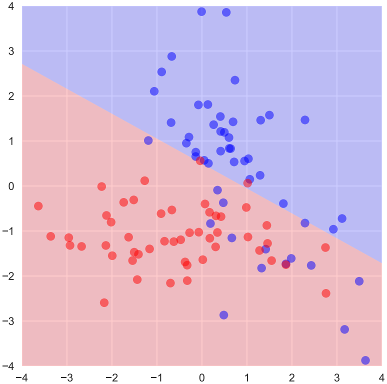
Example

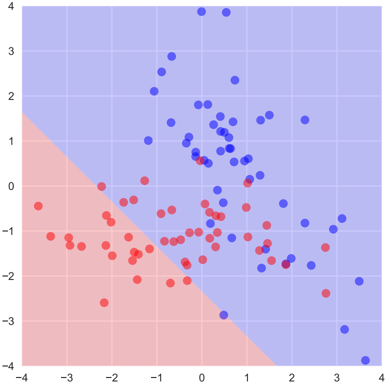




- square loss, sum squares regularizer
- left hand plot shows training errors in blue, test errors in red
- right hand plot shows minimum-error classifier (*i.e.*, $\kappa = 1$)

Example





- left hand plot shows predictor when $\kappa = 0.4$
- right hand plot shows predictor when $\kappa = 4$

Summary

Summary

a *classifier* is a predictor, when the raw output is categorical $v \in \mathcal{V} = \{v_1, \ldots, v_K\}$

- called a *Boolean classifier* when $|\mathcal{V}| = K = 2$, *multi-class classifier* when $K > 2$
- judged by various *error rates*, summarized in a *confusion matrix*, on test data

fitting a classifier to a training data set via ERM or RERM

- we embed the raw output v into $\mathbf{R}^m$ using ψ, with $\psi_i = \psi(v_i)$ the *representative* of class i
- we build a predictor for y, given x
- we *un-embed* a prediction $\hat{y} \in \mathbf{R}^m$ to a class prediction $\hat{v} = \psi^\dagger(\hat{y})$, using nearest neighbor
- there are special loss functions for fitting classifiers, that we'll see later