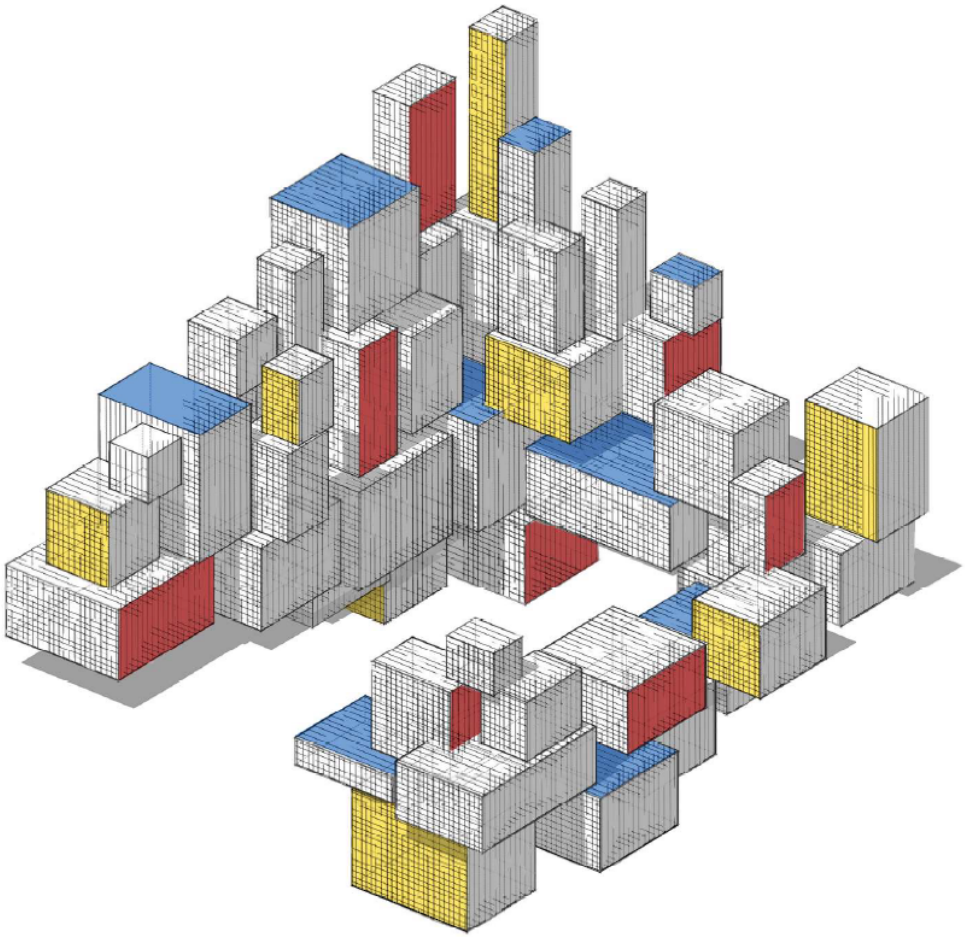
Logistic Regression and NN Classification



Purpose

In this lecture we will discuss:

- The multi-logit model
- Softmax classification
- Nearest Neighbor classification

Logistic Regression and Softmax Classification

We introduced the logistic (logit) regression model as a generalized linear model where, conditional on a p-dimensonal feature vector $\boldsymbol{x}$, the random response $Y \sim \mathsf{Ber}(h(\boldsymbol{x}^\top \boldsymbol{\beta}))$, with

$$h(u) = 1/(1 + \mathrm{e}^{-u}).$$

The parameter $\boldsymbol{\beta}$ is learned from the training data by maximizing the likelihood of the training responses or, equivalently, by minimizing the cross-entropy training loss:

$$-\frac{1}{n} \sum_{i=1}^{n} \ln g(y_i \,|\, \boldsymbol{\beta}, \boldsymbol{x}_i),$$

where $g(y = 1 \,|\, \boldsymbol{\beta}, \boldsymbol{x}) = \frac{1}{1+\mathrm{e}^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}$ and $g(y = 0 \,|\, \boldsymbol{\beta}, \boldsymbol{x}) = \frac{\mathrm{e}^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}{1+\mathrm{e}^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}$.
In particular, we have

$$\ln \frac{g(y = 1 \,|\, \boldsymbol{\beta}, \boldsymbol{x})}{g(y = 0 \,|\, \boldsymbol{\beta}, \boldsymbol{x})} = \boldsymbol{x}^\top \boldsymbol{\beta}. \tag{1}$$

Logit Classification

Hence, the decision boundary $\{\boldsymbol{x} : g(y = 0 \,|\, \boldsymbol{\beta}, \boldsymbol{x}) = g(y = 1 \,|\, \boldsymbol{\beta}, \boldsymbol{x})\}$ is the hyperplane $\boldsymbol{x}^\top \boldsymbol{\beta} = 0$.

If the constant feature is considered separately, that is $\boldsymbol{x} = [1, \widetilde{\boldsymbol{x}}^\top]^\top$, then the boundary is an affine hyperplane in $\widetilde{\boldsymbol{x}}$.

Suppose that training on $\tau = \{(\boldsymbol{x}_i, y_i)\}$ yields the estimate $\widehat{\boldsymbol{\beta}}$ with the corresponding learner $g_\tau(y = 1 \,|\, \boldsymbol{x}) = 1/(1 + \mathrm{e}^{-\boldsymbol{x}^\top \widehat{\boldsymbol{\beta}}})$.

The learner can be used as a pre-classifier from which we obtain the classifier $\mathbb{I}\{g_\tau(y = 1 \,|\, \boldsymbol{x}) > 1/2\}$ or, equivalently,

$$\widehat{y} := \operatorname*{argmax}_{j \in \{0,1\}} g_\tau(y = j \,|\, \boldsymbol{x}).$$

Multi-Logit Classification

The classification methodology for the logit model can be generalized to the multi-logit model where the response takes values in the set $\{0, \ldots, c-1\}$.

The key idea is to replace (1) with

$$\ln \frac{g(y = j \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x})}{g(y = 0 \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x})} = \boldsymbol{x}^\top \boldsymbol{\beta}_j, \quad j = 1, \ldots, c-1, \tag{2}$$

where the matrix $\mathbf{W} \in \mathbb{R}^{(c-1)\times(p-1)}$ and vector $\boldsymbol{b} \in \mathbb{R}^{c-1}$ reparameterize all $\boldsymbol{\beta}_j \in \mathbb{R}^p$ such that (recall $\boldsymbol{x} = [1, \widetilde{\boldsymbol{x}}^\top]^\top$):

$$\mathbf{W}\widetilde{\boldsymbol{x}} + \boldsymbol{b} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{c-1}]^\top \boldsymbol{x}.$$

Thus, the random response Y is assumed to have a conditional probability distribution for which the log-odds ratio with respect to class j and a "reference" class (in this case 0) is linear.

Multi-Logit Classification

The model (2) completely specifies the distribution of Y, namely:

$$g(y \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x}) = \frac{\exp(z_{y+1})}{\sum_{k=1}^{c} \exp(z_k)}, \quad y = 0, \ldots, c-1,$$

where z_1 is an arbitrary constant, say 0, corresponding to the "reference" class $y = 0$, and

$$[z_2, \ldots, z_c]^\top := \mathbf{W}\widetilde{\boldsymbol{x}} + \boldsymbol{b}.$$

Note: $g(y \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x})$ is the $(y+1)$-st component of $\boldsymbol{a} = \text{softmax}(\boldsymbol{z})$, where

$$\text{softmax} : \boldsymbol{z} \mapsto \frac{\exp(\boldsymbol{z})}{\sum_k \exp(z_k)}$$

is the softmax function and $\boldsymbol{z} = [z_1, \ldots, z_c]^\top$.

Multi-Logit Classification

Finally, we can write the classifier as

$$\widehat{y} = \underset{j \in \{0,\ldots,c-1\}}{\operatorname{argmax}} \; a_{j+1}.$$

In summary, we have the sequence of mappings transforming the input $\boldsymbol{x}$ into the output $\widehat{y}$:

$$\boldsymbol{x} \to \mathbf{W}\widetilde{\boldsymbol{x}} + \boldsymbol{b} \to \operatorname{softmax}(\boldsymbol{z}) \to \underset{j \in \{0,\ldots,c-1\}}{\operatorname{argmax}} \; a_{j+1} \to \widehat{y}.$$

In the context of neural networks, $\mathbf{W}$ is called a weight matrix and $\boldsymbol{b}$ is called a bias vector.

Training Loss

The parameters $\mathbf{W}$ and $\boldsymbol{b}$ have to be learned from the training data, which involves minimization of the (supervised version of) the cross-entropy training loss:

$$\frac{1}{n}\sum_{i=1}^{n}\text{Loss}(f(y_i \mid \boldsymbol{x}_i), g(y_i \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x}_i)) = -\frac{1}{n}\sum_{i=1}^{n}\ln g(y_i \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x}_i).$$

Using the softmax function, the cross-entropy loss can be simplified to:

$$\text{Loss}(f(y \mid \boldsymbol{x}), g(y \mid \mathbf{W}, \boldsymbol{b}, \boldsymbol{x})) = -z_{y+1} + \ln\sum_{k=1}^{c}\exp(z_k).$$

K-Nearest Neighbors Classification

Let $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be the training set, with $y_i \in \{0, \ldots, c-1\}$, and let $\boldsymbol{x}$ be a new feature vector.

Define $\boldsymbol{x}_{(1)}, \boldsymbol{x}_{(2)}, \ldots, \boldsymbol{x}_{(n)}$ as the feature vectors ordered by closeness to $\boldsymbol{x}$ in some distance $\text{dist}(\boldsymbol{x}, \boldsymbol{x}_i)$, e.g., the Euclidean distance $\|\boldsymbol{x} - \boldsymbol{x}'\|$.

Let $\tau(\boldsymbol{x}) := \{(\boldsymbol{x}_{(1)}, y_{(1)}) \ldots, (\boldsymbol{x}_{(K)}, y_{(K)})\}$ be the subset of τ that contains K feature vectors $\boldsymbol{x}_i$ that are closest to $\boldsymbol{x}$.

Then the K-nearest neighbors classification rule classifies $\boldsymbol{x}$ according to the most frequently occurring class labels in $\tau(\boldsymbol{x})$.

If two or more labels receive the same number of votes, the feature vector is classified by selecting one of these labels randomly with equal probability.

Voronoi Tessellation

For the case $K = 1$ the set $\tau(\boldsymbol{x})$ contains only one element, say $(\boldsymbol{x}', y')$, and $\boldsymbol{x}$ is classified as y'. This divides the space into n regions

$$\mathcal{R}_i = \{\boldsymbol{x} : \text{dist}(\boldsymbol{x}, \boldsymbol{x}_i) \leqslant \text{dist}(\boldsymbol{x}, \boldsymbol{x}_j), j \neq i\}, \quad i = 1, \ldots, n.$$

For a feature space $\mathbb{R}^p$ with the Euclidean distance, this gives a Voronoi tessellation of the feature space.

The following figure and Python program show how nearest neighbor classification works for 80 randomly simulated points above and below the line $x_2 = x_1$. Points above the line $x_2 = x_1$ have label 0 and points below this line have label 1.

Example: Nearest Neighbor Classification

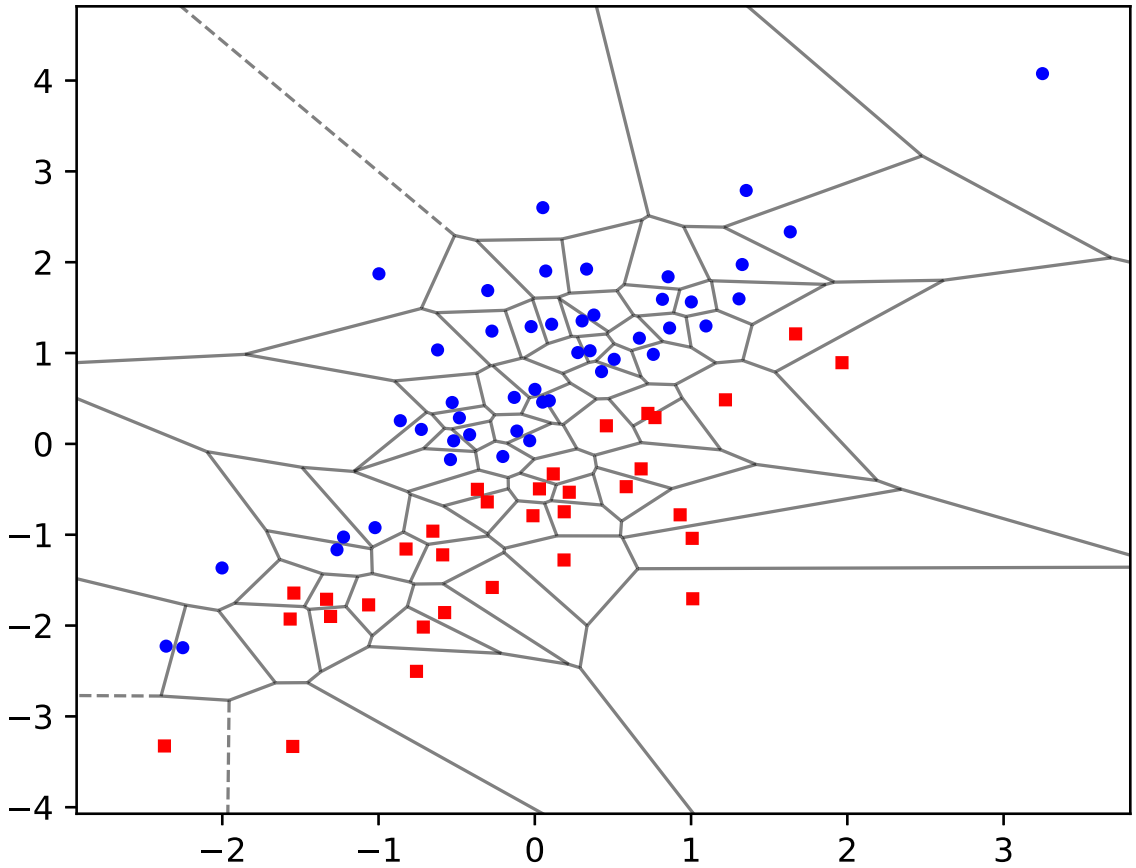


Figure: The 1-nearest neighbor algorithm divides up the space into Voronoi cells.

nearestnb.py

```
import numpy as np
from numpy.random import rand,randn
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d

np.random.seed(12345)
M = 80
x = randn(M,2)
y = np.zeros(M) # pre-allocate list

for i in range(M):
    if rand()<0.5:
        x[i,1], y[i] = x[i,0] + np.abs(randn()), 0
    else:
        x[i,1], y[i] = x[i,0] - np.abs(randn()), 1

vor = Voronoi(x)
plt_options = {'show_vertices':False, 'show_points':False,
               'line_alpha':0.5}
fig = voronoi_plot_2d(vor, **plt_options)
plt.plot(x[y==0,0], x[y==0,1],'bo',
         x[y==1,0], x[y==1,1],'rs', markersize=3)
```