Notes on Classification

Jens Behley

April 26, 2021

Again, some more formal notes on the derivations that we encounter in the lecture.

1 Notation

Some words on the notation:

- Vectors $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{y} \in \mathbb{R}^{1\times D}$ and matrices $\mathbf{X} \in \mathbb{R}^{M\times N}$ are bold. Vectors are usually column vectors if not defined explicitly as row vectors, like $\mathbf{y} \in \mathbb{R}^{1\times D}$
- Sets are calligraphic letters: $\mathcal{X}, \dots,$ where the corresponding letter $x_i \in \mathcal{X}$ usually refers to elements from the set.
- Blackboard bold letters refer to the usual domains: $\mathbb{R}$ are real numbers, $\mathbb{N}$ are natural numbers including 0.
- We write $\mathbf{x}_{1:N}$ or $y_{1:N}$, when we refer to $\mathbf{x}_1, \dots, \mathbf{x}_N$ or $y_1, \dots, y_N$, respectively.
- We generally refer to $\mathbf{x} \in \mathbb{R}^D$ for the feature vector and y for the label, where $y \in \mathbb{R}$ for regression and $y \in \{0, \dots, K-1\}$ for classification. We use $\theta \in \mathbb{R}^D$ for the parameters.[1]
- For the normal distribution aka Gaussian, we use $\mathcal{N}(x|\mu, \sigma^2)$ to refer to a normal distribution with mean μ and variance σ^2.

2 Naive Bayes

In the first lecture, we discussed k-nearest neighbor (kNN) classification, which simply determined the class of a point by searching for k-nearest neighbors from the training set. However, the kNN is quite space intensive as we have to store the whole training data and can be very compute intensive if we have to search for nearest neighbors in very high dimensional data.

[1] We intentional use here 0 for the first class as it is more consistent with the notation in Prince book. Sorry, for the inconsistency in the first lecture.

Now we want to go beyond simply storing all training data and here we now use a generative model with a specific assumption that makes learning easier. We now also want to have a maximum likelihood estimate and therefore wish to find parameters that maximize the likelihood $P(y_{1:N}|x_{1:N}, \theta)$.

As mentioned before, we can use Bayes rule to express $P(y|\mathbf{x})$ as

$$\begin{aligned} P(y|\mathbf{x}) &= \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} && (1)\\ &= \eta P(\mathbf{x}|y)P(y), && (2) \end{aligned}$$

where $\eta = \sum_{k=0}^{K-1} P(\mathbf{x}|y = k)P(y = k)$ is a normalization constant that can be computed from $P(\mathbf{x}|y)$ and $P(y)$.

In Naive Bayes, we assume that the features x_d are independent from each other, and therefore

$$P(\mathbf{x}|y) = \prod_{d=1}^{D} P(x_d|y) \tag{3}$$

For our classification model, we now assume that $P(x_d|y)$ and $P(y)$ are given by a normal distribution and a categorical distribution, respectively.

$$\begin{aligned} P(x_d|y = k) &= \mathcal{N}(x_d|\mu_{k,d}, \sigma_{k,d}^2) && (4)\\ P(y = k) &= \lambda_k && \text{with } \sum_{k=1}^{K} \lambda_k = 1 \quad (5) \end{aligned}$$

In the following, we will derive all terms $\mu_{k,d}$ and $\sigma_{k,d}^2$ for a specific label $y = k$ and drop k in the mean and standard deviation. But the other classes follow the same derivation and we say that our training set has $N = \sum_{k=1}^{K} N_k$ examples with N_k examples for class k.

For the maximum likelihood estimate, we want to find θ^* that maximizes $P(\theta|y_{1:N}, \mathbf{x}_{1:N})$ with uniform prior $P(\theta)$. Therefore, we can simplify the problem as follows:

$$\begin{aligned} \theta* = \arg\max_{\theta} P(\theta|y_{1:N}, \mathbf{x}_{1:N}) &= \arg\max_{\theta} \frac{P(y_{1:N}|\mathbf{x}_{1:N}, \theta)P(\theta)}{P(y_{1:N}|\mathbf{x}_{1:N})} && (6)\\ &\propto \arg\max_{\theta} P(y_{1:N}|\mathbf{x}_{1:N}, \theta) && (7)\\ &\propto \arg\max_{\theta} \prod_{i=1}^{N} P(y_i|\mathbf{x}_i, \theta), && (8) \end{aligned}$$

where we exploited that we have i.i.d. training examples. We see that we only have to deal for the maximum likelihood with the likelihood.

Using the generative model with the Naive Bayes, we arrive at:

$$\theta* = \arg\max_{\theta} \prod_{i=1}^{N} \prod_{d=1}^{D} P(x_{i,d}|y_i, \theta)P(y_i|\theta) \qquad \text{using (2) and (3),} \tag{9}$$

where $x_{i,d}$ refers to the d-th index of feature vector $\mathbf{x}_i$.

Using the negative logarithm to turn the maximization into a minimization problem, we arrive at:

$$\theta* = \arg\min_\theta -\log \prod_{i=1}^{N} \prod_{d=1}^{D} P(x_{i,d}|y_i, \theta) P(y_i|\theta) \tag{10}$$

$$= \arg\min_\theta \underbrace{-\sum_{i=1}^{N} \sum_{d=1}^{D} \log P(x_{i,d}|y_i, \theta) + \log P(y_i|\theta),}_{\mathcal{L}(\theta)} \tag{11}$$

where $\mathcal{L}(\theta)$ is the negative log-likelihood (NLL) as introduced in the regression part, but here for the Naive Bayes objective.

We now derive the equations for μ_d and σ_d^2 first and there the second term, $P(y_i|\theta)$ is irrelevant. Furthermore, the other $P(x_{i,d}|y_i, \theta)$ are independent and can be therefore also dropped. Plugging in Eq. 4 and using the independent of the features, we get:

$$-\sum_{i=1}^{N} \log P(x_{i,d}|y_i, \theta) = -\sum_{i=1}^{N} \log \mathcal{N}(x_d|\mu_{y_i,d}, \sigma_{y_i,d}^2) \tag{12}$$

$$\tag{13}$$

As noted before, we concentrate here on a specific class model, $y_i = k$, and therefore have to account for N_k examples where $y_i = k$. Using the definition of a uni-variate normal distribution, we get:

$$-\sum_{i=1}^{N_k} \log \mathcal{N}(x_d|\mu_d, \sigma_d^2) = -\sum_{i=1}^{N_k} \log \left((2\pi\sigma_d^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \frac{(x_d - \mu_d)^2}{\sigma_d^2} \right) \right) \tag{14}$$

$$= -\sum_{i=1}^{N_k} \left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_d^2) - \frac{1}{2}\frac{(x_d - \mu_d)^2}{\sigma_d^2} \right) \tag{15}$$

$$= \sum_{i=1}^{N_k} \frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_d^2) + \frac{1}{2}\frac{(x_d - \mu_d)^2}{\sigma_d^2} \tag{16}$$

Applying the usual strategy, we now derive the gradient with respect to μ_d

and σ_d and set the gradient to 0 to get the minimum.

$$\frac{\partial \mathcal{L}(\theta)}{\partial \mu_d} = \frac{1}{\partial \mu_d} \sum_{i=1}^{N_k} \frac{1}{2} \log(2\pi) + \frac{1}{2} \log(\sigma_d^2) + \frac{1}{2} \frac{(x_d - \mu_d)^2}{\sigma_d^2} \tag{17}$$

$$= \sum_{i=1}^{N_k} \frac{1}{\sigma_d^2} (x_d - \mu_d) \tag{18}$$

$$= \frac{1}{\sigma_d^2} \left(\sum_{i=1}^{N_k} x_d - \sum_{i=1}^{N_k} \mu_d \right) \tag{19}$$

$$= \frac{1}{\sigma_d^2} \left(\sum_{i=1}^{N_k} x_d - N_k \mu_d \right) \tag{20}$$

Setting the last term to zero, results in the ML estimate:

$$\mu_d = \frac{1}{N_k} \sum_{i=1}^{N_k} x_d \tag{21}$$

The same, we can perform for the standard deviation:

$$\frac{\partial \mathcal{L}(\theta)}{\partial \sigma_d^2} = \frac{1}{\partial \sigma_d^2} \sum_{i=1}^{N_k} \frac{1}{2} \log(2\pi) + \frac{1}{2} \log(\sigma_d^2) + \frac{1}{2} \frac{(x_d - \mu_d)^2}{\sigma_d^2} \tag{22}$$

$$= \sum_{i=1}^{N_k} \left(\frac{1}{2\sigma_d^2} - \frac{1}{2(\sigma_d^2)^2} (x_d - \mu_d)^2 \right) \tag{23}$$

$$= \frac{N_k}{2\sigma_d^2} - \frac{1}{2(\sigma_d^2)^2} \sum_{i=1}^{N_k} (x_d - \mu_d)^2 \tag{24}$$

$$\tag{25}$$

Setting this to zero, results in the ML estimate of σ_d^2:

$$\sigma_d^2 = \frac{1}{N_k} \sum_{i=1}^{N_k} (x_d - \mu_d)^2 \tag{26}$$

For the derivation of $P(y_i \theta)$, we can ignore the likelihood term in Eq. 11, i.e., $\sum_{d=1}^{D} \log P(x_{i,d} | y_i, \theta)$, and therefore we want to minimize:

$$\mathcal{L}(\lambda_{1:K}) := - \sum_{i=1}^{N} \log P(y_i = k | \lambda_k) \tag{27}$$

$$= - \sum_{i=1}^{N} \log \left(\lambda_{y_i} \right) + \nu \left(\sum_{k=1}^{K} \lambda_k - 1 \right), \tag{28}$$

where the second term $\nu\left(\sum_{k=1}^{K}\lambda_k - 1\right)$ is Lagrange multiplier to enforce that the parameters of the categorical distribution sum to 1. (Later we use a different function that ensures this.)

Using the same receipt as before, one can derive the gradient, we will arrive at

$$\lambda_k = \frac{N_k}{\sum_{j=0}^{K-1} N_j} \tag{29}$$

Thus, the maximum likelihood estimate for the categorical distribution is a histogram over the number of entries per class.

Intuitively, a Naive Bayes classifier defines for each class "regions" of good features, where the variance σ_d determines the range of acceptable values.

However, the independence assumption of features is usually not fulfilled, when we look at data such as images. There, neighboring pixels are correlated. Furthermore, assuming only a single parameter range that is acceptable restricts the classifier to very simple decision rules. Can find a better classifier that allows for more complex distribution on "good" feature values?

3 Decision Trees

A decision tree allows us have more complex functions that determine good features. Still decision trees are conceptually simple. Later we will see how to combine multiple trees into a better model.

But let's start with the Decision Tree model: A decision tree is a binary tree, where inner node have a split function $h(\mathbf{x}|\theta) = \{0, 1\}$ that returns 0 indicate that the left branch must be traversed next, and 1 that the right branch must be traversed. Leaves contain models for $P(y|\mathbf{x})$ that represent the prediction of the tree for $\mathbf{x}$.

For the split functions, we can design very complex functions, but often a simple thresholding on a feature x_d is used (also known as stump):

$$h(\mathbf{x}|d,\tau) = \begin{cases} x_d < \tau, & 0 \\ x_d \geq \tau, & 1 \end{cases} \tag{30}$$

Decision Trees are build starting with the root in a greedy manner, i.e., at each node is always the best split function for examples arriving at the node selected. Unless the maximal depth is reached (or a single example is left), a new inner node is created where the current relevant subset $\mathcal{S} \subseteq \mathcal{X}_{\text{train}}$ is split by the split function into subsets $\mathcal{S}_L \subset \mathcal{S}$ and $\mathcal{S}_R \subset \mathcal{S}$, where $\mathcal{S}_L \cap \mathcal{S}_R = \emptyset$:

$$\mathcal{S}_L = \{(\mathbf{x}, y) \in \mathcal{S} | h(\mathbf{x}, \theta) = 0\} \tag{31}$$

$$\mathcal{S}_R = \{(\mathbf{x}, y) \in \mathcal{S} | h(\mathbf{x}, \theta) = 1\} \tag{32}$$

The split function is often selected from splits at features x_d, i.e.,

$$\mathcal{H}_{\mathcal{S}} = \{h(\mathbf{x}|d, \tau)|d \in \{1, \ldots, D\}, \tau \in \{x_d|(\mathbf{x}, y) \in \mathcal{S}\}\} \tag{33}$$

A common criterion for selecting the split function is the information gain, defined as:

$$I(\mathcal{S}, \mathcal{S}_L, \mathcal{S}_R) = H(\mathcal{S}) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}_i|}{|\mathcal{S}|} H(\mathcal{S}_i), \tag{34}$$

where $H(\mathcal{S})$ is the entropy defined as:

$$H(\mathcal{S}) = -\sum_{k=1}^{K} P(y=k) \log(P(y=k)) \quad \text{with} \quad P(y=k) = |\{(\mathbf{x}, y) \in \mathcal{S}|y=k\}| \cdot |\mathcal{S}|^{-1} \tag{35}$$

For each split, we determine the information gain and select the split that maximizes the gain. Thus, splits that lead to more peaked distributions of classes in the subset, i.e., more pure subsets.

Decisions trees and the split functions induce decision boundaries in the feature space. But can't we directly learn decision boundaries?

4 Logistic Regression

We now turn to modeling directly the decision boundary. For this we will now develop a geometric intuition and then use special functions to map distances to probabilities.

Geometric intuition: We define linear decision boundaries, where distance to the boundary tells us how certain we are about the prediction. Thus, we use $\theta^T \mathbf{x} + \theta_0$ as hyperplane (line in 1D, plane in 2D) with a (un-normalized) normal θ as decision boundary between classes. With the length $||\theta||$ of θ, we can scale the distance to the decision boundary and therefore scale the confidence accordingly.

We use the again the trick that we add 1 to the feature vector $\mathbf{x}$ to include θ_0 in the parameters θ. Thus, we have $\mathbf{x} = (1, \mathbf{x}^T)^T$ and $\theta = (\theta_0, \theta_1, \ldots, \theta_D)$ as parameters.

However, $P(y|\mathbf{x})$ should be valid probability distribution and therefore values should be in $P(y|\mathbf{x}) \in [0, 1]$ and $\sum_{k=0}^{K-1} P(y=k|\mathbf{x}) = 1$ should hold. We want to have that $P(y=k|\mathbf{x})$ is largest for the decision boundary that produces the largest distance and the confidence should account for the distances of the other decision boundaries.

4.1 Binary Classification

First, we look at binary classification, where $P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x})$. Using the geometric idea, we now want to turn the distance in a probability in

$[0,1]$ and we can use the sigmoid function $\sigma : \mathbb{R} \mapsto [0,1]$, defined as

$$\sigma(x) = \frac{1}{1+\exp(-x)}. \tag{36}$$

Logistic regression uses now the following model that combines our geometric idea with the sigmoid in a probabilistic discriminative model:

$$\begin{align} P(y=1|\mathbf{x}) &= \sigma(\theta^T \mathbf{x}) \tag{37}\\ &= \frac{1}{1+\exp(-\theta^T \mathbf{x})} \tag{38}\\ P(y=0|\mathbf{x}) &= 1 - P(y=1|\mathbf{x}) \tag{39}\end{align}$$

For deriving the maximum likelihood estimate, we follow the same receipt as before: (1) determine NLL, (2) determine gradient, and (3) solve for 0.

Let $\mathbf{1}\{a\}$ be the indicator function that returns 1 if a is true and 0 otherwise. The negative log-likelihood can be derived as follows:

$$\begin{align} \mathcal{L}(\theta) - \log \prod_{i=1}^{N} P(y_i|\mathbf{x}_i) \tag{40}\\ = -\sum_{i=1}^{N} \log P(y_i|\mathbf{x}_i) \tag{41}\\ = -\sum_{i=1}^{N} \underbrace{\mathbf{1}\{y_i=0\}}_{1-\mathbf{1}\{y=1\}} \log P(y_i=1|\mathbf{x}_i) + \mathbf{1}\{y_i=1\} \log P(y_i=1|\mathbf{x}_i) \tag{42}\\ = -\sum_{i=1}^{N} (1-\mathbf{1}\{y=1\}) \log\left(1 - \frac{1}{1+\exp(-\theta^T\mathbf{x})}\right) \nonumber\\ + \mathbf{1}\{y_i=1\} \log\left(\frac{1}{1+\exp(-\theta^T\mathbf{x})}\right) \tag{43}\\ = -\sum_{i=1}^{N} (1-\mathbf{1}\{y=1\}) \log\left(\frac{1+\exp(-\theta^T\mathbf{x})}{1+\exp(-\theta^T\mathbf{x})} - \frac{1}{1+\exp(-\theta^T\mathbf{x})}\right) \nonumber\\ + \mathbf{1}\{y_i=1\} \log\left(\frac{1}{1+\exp(-\theta^T\mathbf{x})}\right) \tag{44}\end{align}$$

$$= -\sum_{i=1}^{N} \log\left(\frac{\exp(-\theta^T\mathbf{x})}{1+\exp(-\theta^T\mathbf{x})}\right) \\ + \mathbf{1}\{y_i = 1\}\left[\log\left(\frac{1}{1+\exp(-\theta^T\mathbf{x})}\right) - \log\left(\frac{\exp(-\theta^T\mathbf{x})}{1+\exp(-\theta^T\mathbf{x})}\right)\right] \tag{45}$$

$$= -\sum_{i=1}^{N} -\theta^T\mathbf{x} - \log\left(1+\exp(-\theta^T\mathbf{x})\right) \\ + \mathbf{1}\{y_i = 1\}\left[\log(1) - \log(1+\exp(-\theta^T\mathbf{x})) - (-\theta^T\mathbf{x} - \log(1+\exp(-\theta^T\mathbf{x})))\right] \tag{46}$$

$$= -\sum_{i=1}^{N} -\theta^T\mathbf{x} - \log\left(1+\exp(-\theta^T\mathbf{x})\right) \\ + \mathbf{1}\{y_i = 1\}\left[-\log(1+\exp(-\theta^T\mathbf{x})) + \theta^T\mathbf{x} + \log(1+\exp(-\theta^T\mathbf{x}))\right] \tag{47}$$

$$= -\sum_{i=1}^{N} -\theta^T\mathbf{x} - \log\left(1+\exp(-\theta^T\mathbf{x})\right) + \mathbf{1}\{y_i = 1\}\left(\theta^T\mathbf{x}\right) \tag{48}$$

$$= -\sum_{i=1}^{N} (\mathbf{1}\{y_i = 1\} - 1)\theta^T\mathbf{x} - \log\left(1+\exp(-\theta^T\mathbf{x})\right) \tag{49}$$

Next, we determine the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ with respect to θ:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{\partial \theta}\left(-\sum_{i=1}^{N} (\mathbf{1}\{y_i = 1\} - 1)\theta^T\mathbf{x} - \log\left(1+\exp(-\theta^T\mathbf{x})\right)\right) \tag{50}$$

$$= -\sum_{i=1}^{N} (\mathbf{1}\{y_i = 1\} - 1)\mathbf{x} - \frac{1}{1+\exp(-\theta^T\mathbf{x})}\exp(-\theta^T\mathbf{x})(-\mathbf{x}) \tag{51}$$

$$= -\sum_{i=1}^{N} (\mathbf{1}\{y_i = 1\} - 1)\mathbf{x} + \underbrace{\frac{\exp(-\theta^T\mathbf{x})}{1+\exp(-\theta^T\mathbf{x})}}_{1-\sigma(\theta^T\mathbf{x})}\mathbf{x} \tag{52}$$

$$= -\sum_{i=1}^{N} \mathbf{1}\{y_i = 1\}\mathbf{x} - \mathbf{x} + (1 - \sigma(\theta^T\mathbf{x}))\mathbf{x} \tag{53}$$

$$= -\sum_{i=1}^{N} \mathbf{1}\{y_i = 1\}\mathbf{x} - \mathbf{x} + \mathbf{x} - \sigma(\theta^T\mathbf{x})\mathbf{x} \tag{54}$$

$$= -\sum_{i=1}^{N} \mathbf{1}\{y_i = 1\}\mathbf{x} - \sigma(\theta^T \mathbf{x})\mathbf{x} \tag{55}$$

$$= \sum_{i=1}^{N} \sigma(\theta^T \mathbf{x})\mathbf{x} - \mathbf{1}\{y_i = 1\}\mathbf{x} \tag{56}$$

$$= \sum_{i=1}^{N} (\sigma(\theta^T \mathbf{x}) - \mathbf{1}\{y_i = 1\})\mathbf{x} \tag{57}$$

$$= \sum_{i=1}^{N} (P(y_i = 1|\mathbf{x}) - \mathbf{1}\{y_i = 1\})\mathbf{x} \tag{58}$$

Intuitively, gradient pushes the boundary in the direction such that $P(y_i = 1|\mathbf{x})$ reaches 1 for examples where $y_i = 1$ and oppositely if $y_i = 0$.

However, the gradient cannot be solved for zero to get the θ in closed form. Therefore, we have to use the gradient to nudge the solution in the right way.

4.2 Multi-class Logistic Regression

For multiple classes, we want to have multiple decision boundaries and have a θ_k^T for each class $k = \{0, \dots, K-1\}$. Now, we want to choose θ_k such that the distance for the correct class is maximized. But we have to turn the class-wise distances (as with the Logistic Regression) into a probability distribution.

As it turns out, the softmax function softmax : $\mathbb{R}^D \mapsto [0,1]^D$ does exactly this. The softmax is a "soft" version of the argmax function that sums to 1:

$$\text{softmax}(\mathbf{x}) = \mathbf{s} \quad \text{with} \quad s_i = \frac{\exp(x_i)}{\sum_{d=1}^{D} \exp(x_d)} \tag{59}$$

With this, we can define the multi-class logistic regression, which we now call simply Softmax Regression, as follows:

$$P(y = k|\mathbf{x}) = \frac{\exp(\theta_k^T \mathbf{x})}{\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}} \tag{60}$$

As done before with the Logistic Regression, we derive the Negative Log-Likelihood and then determine the gradient:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \log(P(y_i|\mathbf{x}_i)) \tag{61}$$

$$= -\sum_{i=1}^{N} \log \frac{\exp(\theta_{y_i}^T \mathbf{x}_i}{\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}_i)} \tag{62}$$

$$= -\sum_{i=1}^{N} \theta_{y_i}^T \mathbf{x}_i - \log\left(\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}_i)\right) \tag{63}$$

$$= \sum_{i=1}^{N} \log(\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}_i)) - \theta_{y_i}^T \mathbf{x}_i \tag{64}$$

The gradient with respect to θ_j is then given by:

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \frac{1}{\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}_i)} \exp(\theta_j^{\mathbf{x}})\mathbf{x}_i - \mathbf{1}\{y_i = j\}\mathbf{x}_i \tag{65}$$

$$= \sum_{i=1}^{N} \left(\frac{\exp(\theta_j^T \mathbf{x})}{\sum_{j=0}^{K-1} \exp(\theta_j^T \mathbf{x}_i)} - \mathbf{1}\{y_i = j\} \right) \mathbf{x}_i \tag{66}$$

$$= \sum_{i=1}^{N} \left(P(y = j|\mathbf{x}) - \mathbf{1}\{y_i = j\}\right) \mathbf{x}_i \tag{67}$$

Again, this can be optimized with gradient descent, where we simply stack the gradient vectors to update $\theta = (\theta_0^T, \theta_1^T, \ldots, \theta_K^T)^T$