Longxiang Wang
12710172
wlongxiang1119@gmail.com

Homework Assignment 3
Machine Learning 1, 19/20

2019-10-02

1 Naive Bayes Text Classification

a) Write down the data likelihood $p(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta})$

Since we have N data points in the training dataset (i.i.d.), data likelihood can be written out as product of each data point:

$$p(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{t_n}, \boldsymbol{x_n}|\boldsymbol{\theta}) \tag{1}$$

According to product rule of joint probability $p(x, y) = p(x|y)p(y)$, equation 1 can be written using class conditional probability:

$$p(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{t_n}, \boldsymbol{x_n}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x_n}|\boldsymbol{t_n}, \boldsymbol{\theta})p(\boldsymbol{t_n}|\boldsymbol{\theta}) \tag{2}$$

Since $\boldsymbol{t_n}$ is one-hot encoded of dimension K, we only need to consider when class label presents, i.e., when $t_{nk} = 1$:

$$p(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{t_n}, \boldsymbol{x_n}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x_n}|\boldsymbol{t_n}, \boldsymbol{\theta})p(\boldsymbol{t_n}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\prod_{k=1}^{K}(p(\boldsymbol{x_n}|t_{nk} = 1, \boldsymbol{\theta})p(t_{nk} = 1|\boldsymbol{\theta}))^{t_{nk}} \tag{3}$$

By applying Naive Bayes assumption that all features are independent of each other, i.e., features like $x_1, x_2, ..., x_D$ (for a training sample) are not correlated, according to product rule (apply in chain), the class conditional probability can be written out as:

$$p(\boldsymbol{x_n}|t_{nk} = 1, \boldsymbol{\theta}) = p(x_{n1}, x_{n2}, ..., x_{nD}|t_{nk} = 1, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(x_{nd}|t_{nk} = 1, \boldsymbol{\theta}) \tag{4}$$

As we know, $\boldsymbol{\theta}$ describes the distribution of features (words), thus we have the following

$$p(\boldsymbol{x_n}|t_{nk} = 1, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(x_{nd}|t_{nk} = 1, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(x_{nd}|C_k, \theta_{dk}) \tag{5}$$

$$p(t_{nk} = 1|\boldsymbol{\theta}) = p(C_k) = \pi_k \tag{6}$$

Plugging in equations 5 and 6 into equation 3, we have the final form:

$$p(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\prod_{k=1}^{K}(\pi_k \prod_{d=1}^{D} p(x_{nd}|C_k, \theta_{dk}))^{t_{nk}} \tag{7}$$

b) How does number of parameters change under NB assumption?

Under NB assumption, for each class, there are D (number of features) parameters:

$$total\_parameters = KD \tag{8}$$

Without NB assumption, for each class, we also need to consider the covariance amongst all features, according to combinatorics, the total number of parameters become exponential:

$$total\_parameters \approx K(2^D) \tag{9}$$

Some words tend to apprear more often than others. For example, tulips, Dutch, windmill have high chance to appear together.

c) Write the log form of data likelihood for Bernoulli model

Under Bernoulli distribution, the class condition probability can be written as

$$p(\boldsymbol{x}|C_k, \theta_{1k}, \theta_{1k}, ..., \theta_{Dk}) = \prod_{d=1}^{D} p(x_d|C_k, \theta_{1k}, ..., \theta_{dk}) = \prod_{d=1}^{D} \theta_{dk}^{x_d}(1-\theta_{dk})^{1-x_d} \tag{10}$$

Thus the data likelihood from last question can be written as:

$$\begin{aligned} lnp(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta}) &= ln \prod_{n=1}^{N}\prod_{k=1}^{K}(\pi_k \prod_{d=1}^{D} p(x_{nd}|C_k, \theta_{dk}))^{t_{nk}} && (11)\\ &= ln \prod_{n=1}^{N}\prod_{k=1}^{K}\left(\pi_k \prod_{d=1}^{D} \theta_{dk}^{x_d}(1-\theta_{dk})^{1-x_d}\right)^{t_{nk}} && (12)\\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left(ln\pi_k + \sum_{d=1}^{D}\left(x_{nd}ln\theta_{dk} + (1-x_{nd})ln(1-\theta_{dk})\right)\right) && (13) \end{aligned}$$

d) Solve for MLE for θ_{dk}

We can solve for θ_{dk} independently for a class C_k, the partial derivative with respect to θ_{dk} can be simplified by discarding all other θ terms.

$$\begin{aligned} \frac{\partial lnp(\boldsymbol{T}, \boldsymbol{X}|\boldsymbol{\theta})}{\partial\theta_{dk}} &= \frac{\partial \sum_{n\in C_k}^{N}\left(x_{nd}ln\theta_{dk} + (1-x_{nd})ln(1-\theta_{dk})\right)}{\partial\theta_{dk}} && (14)\\ &= \sum_{n\in C_k}^{N}\left(\frac{x_{nd}}{\theta_{dk}} - \frac{1-x_{nd}}{1-\theta_{dk}}\right) = 0 && (15) \end{aligned}$$

From Equation 15, we can calculate parameter as below:

$$\sum_{n\in C_k}^{N}\frac{x_{nd}}{\theta_{dk}} = \sum_{n\in C_k}^{N}\frac{1-x_{nd}}{1-\theta_{dk}} \rightarrow \theta_{dk}\sum_{n\in C_k}^{N}(1-x_{nd}) = (1-\theta_{dk})\sum_{n\in C_k}^{N}x_{nd} \rightarrow \theta_{dk}\sum_{n\in C_k}^{N}1 = \sum_{n\in C_k}^{N}x_{nd} \rightarrow \theta_{dk} = \frac{\sum_{n\in C_k}^{N}x_{nd}}{N_k} \tag{16}$$

Where N_k represents number of documents in the training set which belongs to class k, and $\sum_{n\in C_k}^{N} x_{nd}$ represents number of document in the training set which contains word d.
Therefore, θ_{dk} can be interpreted as the frequency (or probability if data is large enough) of documents containing a certain word in all documents belonging to class k.

e) Write $p(C_1|\boldsymbol{x})$ for general k-class NB classifier

According to Bayes theorem, the probability of class 1 given observation $\boldsymbol{x}$ is given as

$$p(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x})} \tag{17}$$

According to sum rule and product rule, the evidence is given by

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} p(\boldsymbol{x}|C_k)p(C_k) \tag{18}$$

Under Naive Bayes assumption, the likelihood is given by

$$p(\boldsymbol{x}|C_1) = p(x_1, x_2, ...x_D|C_k) = \prod_{d=1}^{D} p(x_d|C_1) \tag{19}$$

And the prior is given by

$$p(C_1) = \pi_1 \tag{20}$$

Plugging Equation 20, Equation 19 and Equation 18 into 17, we have the final form

$$p(C_1|\boldsymbol{x}) = \frac{\pi_1\prod_{d=1}^{D}p(x_d|C_1)}{\sum_{k=1}^{K}p(\boldsymbol{x}|C_k)\pi_k} = \frac{\pi_1\prod_{d=1}^{D}p(x_d|C_1)}{\sum_{k=1}^{K}\pi_k\prod_{d=1}^{D}p(x_d|C_k)} = \frac{\pi_1\prod_{d=1}^{D}p(x_d|C_1, \theta_{d1})}{\sum_{k=1}^{K}\pi_k\prod_{d=1}^{D}p(x_d|C_k, \theta_{dk})} \tag{21}$$

f) Write $p(C_1|x)$ for general k-class NB classifier and Bernoulli model

Based on Equation 21, and plug in the Bernoulli assumption with Equation 10, we have:

$$p(C_1|\boldsymbol{x}) = \frac{\pi_1 \prod_{d=1}^{D} \theta_{d1}^{x_d}(1-\theta_{d1})^{1-x_d}}{\sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} \theta_{dk}^{x_d}(1-\theta_{dk})^{1-x_d}} \tag{22}$$

2 Multi-class logistic regression and multilayer perceptrons

a) Derive the matrix form for the gradient of the log-likelihood $\nabla_{\boldsymbol{w}_j} lnp(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K})$

We first write down the likelihood:

$$p(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) = \prod_{n=1}^{N} p(\boldsymbol{t_n}|\boldsymbol{\phi_n}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_k(\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{23}$$

Where N represents number of training samples, K represents number of classes. $t_{nk} = 1$ is used as a selector for class k in one hot encoded vector $\boldsymbol{t_n}$. And $y_{nk} = y_k(\phi_n)$.
Then the log-likelihood is given by

$$lnp(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln\Big(y_k(\phi_n)\Big) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln y_{nk} \tag{24}$$

We first derive the preliminaries for calculating the gradient with repspect to w_j. Using chain rule and sigmod function property $\frac{\partial sig(x)}{\partial x} = sig(x)(1 - sig(x))$, we have:

$$a_j = w_j^T \phi \longrightarrow \frac{\partial a_j}{\partial w_j} = \phi^T \tag{25}$$

$$\frac{\partial y_k}{\partial w_j} = \frac{\partial y_k}{\partial a_j}\frac{\partial a_j}{\partial w_j} = y_k(I_{kj} - y_j)\phi^T \tag{26}$$

Where I_{kj} are the elements of an identity matrix.
Now we can calculate the gradient:

$$\begin{aligned}
\nabla_{\boldsymbol{w}_j} lnp(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) &= \nabla_{\boldsymbol{w}_j}\Big(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln y_{nk}\Big) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{t_{nk}}{y_{nk}}\frac{\partial y_{nk}}{\partial w_j} \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{t_{nk}}{y_{nk}} y_{nk}(I_{kj} - y_{nj})\phi_n^T \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}(I_{kj} - y_{nj})\phi_n^T \\
&= \sum_{n=1}^{N}(t_{nj} - y_{nj})\phi_n^T
\end{aligned} \tag{27}$$

We can further write Euqation 27 into matrix form:

$$\boldsymbol{T} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1K} \\ t_{21} & t_{22} & \cdots & t_{2K} \\ \cdots & \cdots & \cdots & \cdots \\ t_{N1} & t_{N2} & \cdots & t_{NK} \end{bmatrix} \tag{28}$$

$$\boldsymbol{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1K} \\ y_{21} & y_{22} & \cdots & y_{2K} \\ \cdots & \cdots & \cdots & \cdots \\ y_{N1} & y_{N2} & \cdots & y_{NK} \end{bmatrix}, \quad where \quad y_{nk} = y_k(\phi_n) \tag{29}$$

$$\boldsymbol{\Phi} = [\boldsymbol{\phi_1}, \boldsymbol{\phi_2}, \ldots, \boldsymbol{\phi_N}]^T \tag{30}$$

$$\Downarrow$$

$$\nabla_{\boldsymbol{w}_j} lnp(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) = \sum_{n=1}^{N}(t_{nj} - y_{nj})\phi_n^T = (\boldsymbol{T}_{:,j} - \boldsymbol{Y}_{:,j})^T \boldsymbol{\Phi} \tag{31}$$

By rewriting the gradient into matrix form, we can easily derive the gradient for other parameters for other classes, and also it is easy to use modern machine learning framework to calculate vectorized gradients in parallel and optimized manner.

b) Write down the negative log-likelihood

The negative log-likelihood is given by

$$E(\boldsymbol{w}) = -lnp(\boldsymbol{T}|\boldsymbol{\phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_K}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln y_{nk} \tag{32}$$

The negative log-likelihood is the cross-entropy loss function. By adding a negative sign, maximizing likelihood is equivalent to minimizing the cross entropy loss function. When it comes to optimization and weights update, doing gradient ascend with respect to original log-likelihood is equivalent to do gradient descend to the cross entropy loss function.

c) Write down min-batch gradient descent algorithm for logistic regression

We take the average of gradients in one batch of size B to determine the direction of weight update ($\boldsymbol{w}_j$ as an example, all other weights should be updated the same way in each step):

$$\boldsymbol{w}_j^{t+1} = \boldsymbol{w}_j^t - \eta^t \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{w}_j} E_b(\boldsymbol{w})^T \tag{33}$$

where $j = 1, 2, \ldots, K$, t represents step number and η^t represents learning rate at step t.
The mini-batch GD algorithm is described below:

1. initialize random weights: $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K$
2. initialize learning rate η
3. define stop condition, one could choose to limit total number of training steps, or choose a threshold for $|\boldsymbol{w}^{t+1} - \boldsymbol{w}^t| < \epsilon$
4. initialize step $t = 0$
5. updates the weights iteratively until stop condition is met. e.g.:
   (a) choose a batch from $[1, n_B]$ either sequentially or randomly
   (b) update the weights fot $j = 1, 2, \ldots, K$: $\boldsymbol{w}_j^{t+1} = \boldsymbol{w}_j^t - \eta^t \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{w}_j} E_b(\boldsymbol{w})^T$
   (c) update step $t = t + 1$
   (d) decrease learning rate η^t (e.g. time-based decay: $\eta^{t+1} = \frac{\eta^t}{1+kt}$ where k is decay rate)
6. return calculated weights: $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K$

Mini-batch GD is generally more stable and easier to converge than SGD with single sample, according to the law of large numbers, using mini-batch has better approximation to actual steepest gradient direction thus giving better optimization performance.
Full GD has better approximation to the real gradient compared to mini-batch GD, but is suffers from longer training time due to the large number of gradients to calculate in a single step.

d) Multilayer perceptron

d.1) Example forward pass and loss evaluation

We know that softmax is defined as $softmax(\boldsymbol{x}) = \frac{e^x}{\sum_i e^x}$, therefore we have

$$\begin{aligned} a_1 &= W_1 x = [0.724, 0.412]^T \\ z_1 &= tanh(a_1) = tanh([0.724, 0.412]^T) = [0.622, 0.390]^T \end{aligned} \tag{34}$$

$$\begin{aligned} a_2 &= W_2 z_1 = [0.414, 0.631, 0.830]^T \\ y &= softmax(a_2) = softmax([0.414, 0.631, 0.830]^T) = [0.266, 0.331, 0.403]^T \end{aligned} \tag{35}$$

$$E_n(\boldsymbol{W}) = -\sum_{k=1}^{K} t_k ln y_k = -0ln(0.266) - 0ln(0.331) - 1ln(0.403) = 0.909 \tag{36}$$

d.2) Compute the derivative $\frac{\partial E}{\partial w_5}$

Let's calculate the derivative w.r.t. W_2 first:

$$\frac{\partial E}{\partial W_2} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial W_2} = \boldsymbol{\delta_k Z_1^T} = (\boldsymbol{y} - \boldsymbol{t}) \cdot \boldsymbol{Z_1}^T \tag{37}$$

If we write out Equation 37, we get:

$$\begin{bmatrix} \frac{\partial E}{\partial w_5} & \frac{\partial E}{\partial w_6} \\ \frac{\partial E}{\partial w_7} & \frac{\partial E}{\partial w_8} \\ \frac{\partial E}{\partial w_9} & \frac{\partial E}{\partial w_{10}} \end{bmatrix} = \begin{bmatrix} (y_1 - t_1)z_1 & (y_1 - t_1)z_2 \\ (y_2 - t_2)z_1 & (y_2 - t_2)z_2 \\ (y_3 - t_3)z_1 & (y_3 - t_3)z_2 \end{bmatrix} \tag{38}$$

With that, we have $\frac{\partial E}{\partial w_5} = (y_1 - t_1)z_1 = 0.166$

d.3) Perform a weight update

We update the weights as below:

$$\boldsymbol{w}_j^{t+1} = \boldsymbol{w}_j^t - \eta^t \frac{\partial E_b(\boldsymbol{w})^T}{\partial \boldsymbol{w}_j} \tag{39}$$

Therefore, we have new weights with $\eta = 0.05$:

$$W_2^{new} = W_2 - \eta\frac{\partial E}{\partial W_2} = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \\ w_9 & w_{10} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial E}{\partial w_5} & \frac{\partial E}{\partial w_6} \\ \frac{\partial E}{\partial w_7} & \frac{\partial E}{\partial w_8} \\ \frac{\partial E}{\partial w_9} & \frac{\partial E}{\partial w_{10}} \end{bmatrix} = \begin{bmatrix} 0.112 & 0.865 \\ 0.810 & 0.304 \\ 0.789 & 0.912 \end{bmatrix} \tag{40}$$

$$W_1^{new} = W_1 - \eta\frac{\partial E}{\partial W_1} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2} \\ \frac{\partial E}{\partial w_3} & \frac{\partial E}{\partial w_4} \end{bmatrix} = \begin{bmatrix} 0.401 & 0.873 \\ 0.583 & 0.346 \end{bmatrix} \tag{41}$$

d.4) Compute new loss after weights update

Repeating the process in solution d.1), replace weights with the new weights, we calculate the new loss as

$$\begin{aligned} a_1 &= W_1 x = [0.732, 0.417]^T \\ z_1 &= tanh(a_1) = tanh([0.732, 0.417]^T) = [0.624, 0.394]^T && (42) \\ a_2 &= W_2 z_1 = [0.411, 0.625, 0.852]^T \\ y &= softmax(a_2) = softmax([0.411, 0.625, 0.852]^T) = [0.264, 0.327, 0.410]^T && (43) \end{aligned}$$

$$E_n(\boldsymbol{W}) = -\sum_{k=1}^{K} t_k ln y_k = -0ln(0.264) - 0ln(0.327) - 1ln(0.413) = 0.892 \tag{44}$$

As we can see, the loss decreases after weights update. This means the optimization process is going towards a local minimal. Intuitively, we can see from y that the prediction for class 3 is increasing while the other 2 classes are decreasing.