

COMP 540      HW 3  
Peiguang Wang, Xinran Zhou  
Due: 1/18/2018

## 1: MAP and MLE parameter estimation

1. Estimate for  $\theta$  using MLE

**Solution.** The maximum likelihood estimation of  $D$  given  $\theta$  is that

$$MLE = l(D|\theta) = p(x^{(i)}) = \prod_i^m \theta^{x^{(i)}} (1 - \theta)^{1-x^{(i)}}$$

take the NLL of MLE

$$NLL = - \sum_i^m [x^{(i)} \log \theta + (1 - x^{(i)}) \log(1 - \theta)]$$

take the derivative of NLL and make it equal to zero

$$\frac{\partial NLL}{\partial \theta} = - \sum_i^m [x^{(i)} \frac{1}{\theta} - \frac{1}{(1 - \theta)} (1 - x^{(i)})] = 0$$

by computing this equation, we can get  $\theta_{MLE}$

$$\theta_{MLE} = \frac{1}{m} \sum_i^m x^{(i)}$$

2. Compare the MAP and MLE estimates of  $\theta$

**Solution.** If we add a conjugate prior and use both the D and this prior to make a estimation of  $\theta$ , we can have this

$$\begin{aligned} MAP &= l(D|\theta) \text{Beta}(D|a, b) \\ &= [\prod_i^m \theta^{x^{(i)}} (1 - \theta)^{1-x^{(i)}}] \theta^{a-1} (1 - \theta)^{b-1} \end{aligned}$$

take the derivative of  $\theta$  and make it equal to zero, we can get  $\theta_{MAP}$

$$\theta_{MAP} = \frac{\sum_i^m x^{(i)} + a + 1}{m + a + b - 2}$$

if  $a = b = 1$  then

$$\theta_{MAP} = \theta_{MLE} = \frac{1}{m} \sum_i^m x^{(i)}$$

## 2: Logistic regression and Gaussian Naive Bayes

1. For logistic regression, what is the posterior probability for each class, i.e.,  $P(y = 1|x)$  and  $P(y = 0|x)$ ? Write the expression in terms of the parameter  $\theta$  and the sigmoid function.

**Solution.**

$$P(y = 1|x) = h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}}$$

$$P(y = 0|x) = 1 - h_{\theta}(X) = \frac{e^{-\theta^T X}}{1 + e^{-\theta^T X}}$$

2. Derive the posterior probabilities for each class

**Solution.** The Gaussian distribution and Bernoulli distribution that we assume

$$P(y = 1) = \gamma$$

$$p(x_j|y = 1) = N(\mu_j^1, \sigma_j^2)$$

$$P(x_j|; y = 0) = N(\mu_j^0, \sigma_j^2)$$

Naïve Bayes model

$$p(x|y) = \prod_j^d P(x_j|y)$$

Bayes rule

$$P(y = 1|x) = \frac{P(y = 1)P(x|y = 1)}{\sum P(y = i)P(x|y = i)}$$

so these are what we can use now, then we will derive the posterior probabilities

$$\begin{aligned} p(y = 1|x) &= \frac{P(y = 1)P(x|y = 1)}{P(y = 0)P(x|y = 0) + P(y = 1)P(x|y = 1)} \\ &= \frac{\gamma \prod_{j=1}^d N(\mu_j^1, \sigma_j^2)}{\gamma \prod_{j=1}^d N(\mu_j^1, \sigma_j^2) + (1 - \gamma) \prod_{j=1}^d N(\mu_j^0, \sigma_j^2)} \\ &= \frac{1}{1 + \frac{1-\gamma}{\gamma} \prod_{j=1}^d \exp(\frac{(x_j - \mu_j^1)^2 - (x_j - \mu_j^0)^2}{2\sigma_j^2})} \end{aligned}$$

the probability  $P(y = 0|x)$  can be derived using the same method or just subtract it from 1.

3. part 3

**Solution.** Class 1 and class 0 are equally likely, that means  $\gamma = \frac{1}{2}$  the probability equation can be written as

$$P(y = 1|x) = \frac{1}{1 + \frac{1-\gamma}{\gamma} \prod_{j=1}^d \exp(\frac{(x_j - \mu_j^1)^2 - (x_j - \mu_j^0)^2}{2\sigma_j^2})}$$

if we set  $\mu_j^0 = -\mu_j^1$  then we have

$$P(y = 1|x) = \frac{1}{1 + \prod_{j=1}^d e^{(\frac{2x_j\mu_j^0}{\sigma_j^2})}}$$

obviously it has the same form as logistic regression, if we see in this way

$$\theta = [\frac{2\mu_1^0}{\sigma_1^2}, \frac{2\mu_2^0}{\sigma_2^2}, \dots, \frac{2\mu_d^0}{\sigma_d^2}]^T$$

$$X = [x_1, x_2, \dots, x_d]^T$$

the equation can be rewritten using  $\theta$  and  $X$

$$P(y = 1|x) = \frac{1}{1 + e^{-\theta^T X}}$$

### 3: Reject option in classifiers

1. part 1

**Solution.** The loss of choosing a class  $j$  is

$$loss = \lambda_s(1 - P(y = j|x))$$

then if we want to decide  $y = j$ , we need to make  $loss \leq \lambda_r$ , which means that

$$\lambda_s(1 - P(y = j|x)) \leq \lambda_r$$

so

$$P(y = j|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

2. When  $\frac{\lambda_r}{\lambda_s}$  approaches to 0, the cost of falsely classifying is extremely large, which means that the cost of rejects is small, so we tend to choose rejection. As  $\frac{\lambda_r}{\lambda_s}$  increases, the cost of rejection increases. And when  $\frac{\lambda_r}{\lambda_s} = 1$  we would never reject.

### 4: Kernelizing k-nearest neighbors

#### 5: Constructing kernels

1.  $k(x, x') = Ck_1(x, x')$

**Solution.**

$$Ck_1(x, x') = C\Phi_1(x)^T \Phi_1(x') = (\sqrt{C}\Phi_1(x)^T)(\sqrt{C}\Phi_1(x')^T)$$

if  $C \geq 0$  then  $k(x, x')$  is a valid kernel.

2.  $k(x, x') = f(x)k_1(x, x')f(x')$

**Solution.**

$$f(x)k_1(x, x')f(x') = \langle f(x)\Phi_1(x), f(x')\Phi_1(x') \rangle$$

since it satisfies the Mercer's theorem,  $k(x, x')$  is valid

3.  $k(x, x') = k_1(x, x') + k_2(x, x')$

**Solution.** Because  $k_1(x, x')$  and  $k_2(x, x')$  both are valid kernels, so by Mercer's theorem they both satisfy

$$\int_d k(x, x')f(x)f(x') \geq 0$$

then  $k(x, x') = \Phi(x)^T \Phi(x')$  exists. So

$$k_1(x, x') + k_2(x, x') = \int_d k_1(x, x')f(x)f(x') + \int_d k_2(x, x')f(x)f(x')$$

Because  $\int_d k_1(x, x')f(x)f(x') > 0$  and  $\int_d k_2(x, x')f(x)f(x') > 0$ , so

$$k_1(x, x') + k_2(x, x') \geq 0$$

## 6: One vs all logistic regression

1. Implementing and predicting a one-vs-all classifier for the CIFAR-10 dataset.

```
one_vs_all on raw pixels final test set accuracy: 0.361700
[[463  58  22  24  19  35  26  60 203  90]
 [ 69 464  17  34  23  31  44  50  95 173]
 [123  64 193  77  96  89 151  88  72  47]
 [ 67  86  78 161  48 192 171  51  63  83]
 [ 65  38 103  64 234  90 194 129  35  48]
 [ 50  64  81 128  81 274 113  87  67  55]
 [ 31  52  67 102  86  78 457  51  30  46]
 [ 53  62  51  46  69  84  66 407  48 114]
 [144  79   8  25  10  34  22  19 544 115]
 [ 59 208  14  22  23  29  59  56 110 420]]
```

Figure 1: confusion matrix of ova

2. Visualizing the learned parameter matrix

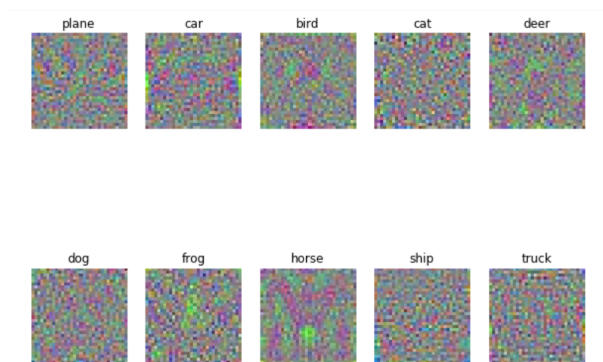


Figure 2: learned one-vs-all classifier visualization

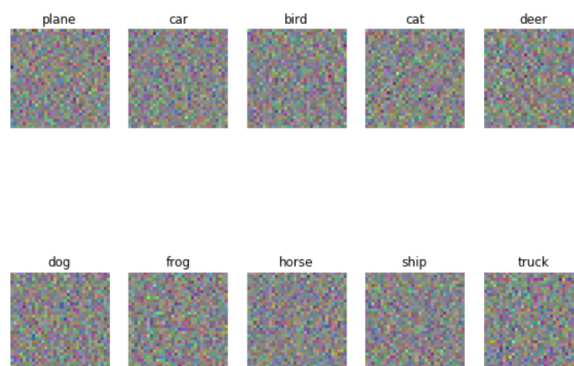


Figure 3: sklearn OVA classifier visualization

## 7: Softmax Regression

1. loss function for softmax regression (naive version)

**Solution.** When implemented in naive version, the softmax regression results are shown in below. Note it took 14.744 seconds to compute the loss.

numerical: -3.709464 analytic: -3.709464, relative error: 8.806310e-09  
 numerical: -0.474544 analytic: -0.474544, relative error: 5.447825e-08  
 numerical: 2.137769 analytic: 2.137769, relative error: 1.418326e-08  
 numerical: 2.231634 analytic: 2.231634, relative error: 5.038103e-08  
 numerical: 0.342416 analytic: 0.342416, relative error: 3.329287e-08  
 numerical: 2.323395 analytic: 2.323395, relative error: 1.629854e-08  
 numerical: -0.570532 analytic: -0.570532, relative error: 2.058362e-08  
 numerical: 1.082908 analytic: 1.082908, relative error: 5.096985e-10  
 numerical: 0.440409 analytic: 0.440409, relative error: 3.809953e-08  
 numerical: 2.232702 analytic: 2.232702, relative error: 2.706612e-08  
 naive loss: 2.352202e+00 computed in 14.744000s

2. loss function for softmax regression (vectorized version)

**Solution.** When implemented in naive version, the softmax regression results are shown in below.  
 Note it took 0.483000 seconds to compute the loss.  
 vectorized loss: 2.352202e+00 computed in 0.483000s  
 Loss difference: 0.000000  
 Gradient difference: 0.000000

3. Compare naive version and vectorized version

**Solution.** It took 14.744000s for naive version to compute the result; and 0.483000 for vectorized version. The vectorized version is much faster than the naive version.

4. Using a validation set to select regularization lambda and learning rate for gradient descent.

**Solution.** We choose mini patch size 400, and set the number of iteration to 4000. The program result is: Best validation accuracy achieved during cross-validation: 0.418000

5. Evaluating the best softmax classifier on the test set and visualizing the coefficients

**Solution.** Using the selected best softmax model on testing dataset. The program result is:  
 softmax on raw pixels final test set accuracy: 0.405100  
 Visualize the coefficients in Figure 4.

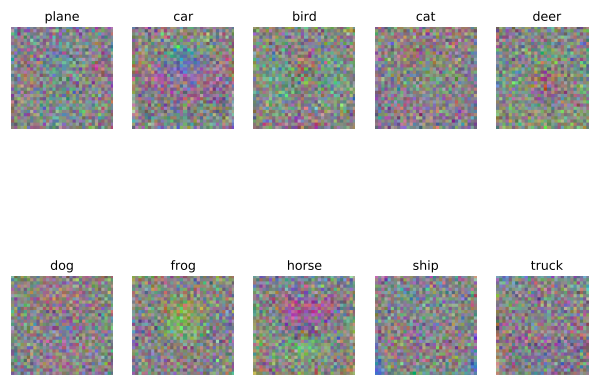


Figure 4: Visualization of the coefficients