Solution to Homework 3

Shoeb Mohammed and Zhuo Chen

February 21, 2016

1 MAP and MLE parameter estimation

$\mathcal{D} = \{x^{(i)} | 1 \leq i \leq m\}$ where $x^{(i)} \sim$ i.i.d $Ber(\theta)$

1.1

If m_1 are number of heads, m_0 are number of tails and $m_0 + m_1 = m$ then the likelihood and MLE for θ are

$$p(\mathcal{D}|\theta) = \theta^{m_1}(1-\theta)^{m_0} \tag{1}$$

$$\begin{aligned} \theta_{MLE} &= argmax_\theta \theta^{m_1}(1-\theta)^{m_0} \\ &= argmax_\theta m_1 \log\theta + m_0 \log(1-\theta) \end{aligned} \tag{2}$$

θ_{MLE} satisfies (first derivative of the likelihood equals zero)

$$\frac{m_1}{\theta_{MLE}} - \frac{m_0}{1-\theta_{MLE}} = 0 \tag{3}$$

Thus,

$$\begin{aligned} \theta_{MLE} &= \frac{m_1}{m_0 + m_1} \\ &= \frac{m_1}{m} \end{aligned} \tag{4}$$

1.2

The prior is

$$\begin{aligned} p(\theta) &= Beta(\theta|a,b) \\ &\propto \theta^{(a-1)}(1-\theta)^{(b-1)} \end{aligned} \tag{5}$$

Thus, the posterior is

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\ &= \frac{p(\mathcal{D}|\theta)p(\theta)}{\sum_{\theta'} p(\mathcal{D}|\theta')p(\theta')} \\ &\propto \theta^{m_1+a-1}\theta^{m_0+b-1} \end{aligned} \tag{6}$$

Thus,

$$\begin{aligned}\theta_{MAP} &= argmax_\theta \theta^{m_1+a-1}\theta^{m_0+b-1} \\ &= argmax_\theta (m_1+a-1)\log\theta + (m_0+b-1)\log(1-\theta)\end{aligned} \tag{7}$$

Equation 7 is similar to MLE estimation. Thus,

$$\begin{aligned}\theta_{MAP} &= \frac{m_1+a-1}{m_0+m_1+a+b-2} \\ &= \frac{m_1+a-1}{m+a+b-2}\end{aligned} \tag{8}$$

It is clear from equations 8 and 4 that $\theta_{MAP} = \theta_{MLE}$ when $a = b = 1$.

2 Logistic regression and Gaussian Naive Bayes

2.1

$$\begin{aligned}p(y=1|x) &= g(\theta^T x) \\ p(y=0|x) &= 1 - g(\theta^T x)\end{aligned} \tag{9}$$

2.2

With naives Bayes assumption,

$$\begin{aligned}p(y=1|x) &= \frac{p(x|y=1)p(y=1)}{p(x)} \\ &= \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \\ &= \frac{p(x|y=1)\gamma}{p(x|y=1)\gamma+p(x|y=0)(1-\gamma)} \quad \text{since } y \sim Ber(\gamma) \\ &= \frac{\prod_{j=1}^{d}\mathcal{N}(\mu_j^1,\sigma_j^2)\gamma}{\prod_{j=1}^{d}\mathcal{N}(\mu_j^1,\sigma_j^2)\gamma+\prod_{j=1}^{d}\mathcal{N}(\mu_j^0,\sigma_j^2)(1-\gamma)} \quad \text{since } p(x_j|y=1) \sim \mathcal{N}(\mu_j^1,\sigma_j^2) \text{ and } p(x_j|y=0) \sim \mathcal{N}(\mu_j^0,\sigma_j^2) \\ &= \frac{\mathcal{N}(\mu^1,\Sigma)\gamma}{\mathcal{N}(\mu^1,\Sigma)\gamma+\mathcal{N}(\mu^0,\Sigma)(1-\gamma)} \quad \text{where } \mu_0 = (\mu_1^0 \cdots \mu_d^0)^T \text{ , } \mu_1 = (\mu_1^1 \cdots \mu_d^1)^T, \Sigma = diag(\sigma_1^2 \cdots \sigma_d^2)\end{aligned} \tag{10}$$

$$\begin{aligned}p(y=0|x) &= \frac{p(x|y=0)p(y=0)}{p(x)} \\ &= \frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \\ &= \frac{\mathcal{N}(\mu^0,\Sigma)\gamma}{\mathcal{N}(\mu^1,\Sigma)\gamma+\mathcal{N}(\mu^0,\Sigma)(1-\gamma)} \quad \text{where } \mu_0 = (\mu_1^0 \cdots \mu_d^0)^T \text{ , } \mu_1 = (\mu_1^1 \cdots \mu_d^1)^T, \Sigma = diag(\sigma_1^2 \cdots \sigma_d^2)\end{aligned} \tag{11}$$

2.3

Proof. With uniform class priors, equation 10 gives

$$\begin{aligned}
p(y=1|x) &= \frac{\mathcal{N}(\mu^1,\Sigma)}{\mathcal{N}(\mu^1,\Sigma)+\mathcal{N}(\mu^0,\Sigma)} \\
&= \frac{1}{1+\frac{\mathcal{N}(\mu^0,\Sigma)}{\mathcal{N}(\mu^1,\Sigma)}} \\
&= \frac{1}{1+\frac{\exp(\frac{1}{2}(x-\mu^1)^T\Sigma^{-1}(x-\mu^1))}{\exp(\frac{1}{2}(x-\mu^0)^T\Sigma^{-1}(x-\mu^0))}} \\
&= \frac{1}{1+\frac{\exp((x-\mu^1)^T\Lambda^2(x-\mu^1))}{\exp((x-\mu^0)^T\Lambda^2(x-\mu^0))}} \quad \text{where } \Lambda = diag\left(\frac{1}{\sqrt{2}\sigma_1}\cdots\frac{1}{\sqrt{2}\sigma_d}\right) \\
&= \frac{1}{1+\frac{\exp((\Lambda(x-\mu^1))^T(\Lambda(x-\mu^1)))}{\exp((\Lambda(x-\mu^0))^T(\Lambda(x-\mu^0)))}} \\
&= \frac{1}{1+\frac{\exp((\Lambda(z+a))^T(\Lambda(z+a)))}{\exp((\Lambda(z-a))^T(\Lambda(z-a)))}} \quad \text{where } a = \frac{\mu^0-\mu^1}{2} \text{ and } z = x - \frac{\mu^0+\mu^1}{2} \\
&= \frac{1}{1+\exp\left((\Lambda(z+a))^T(\Lambda(z+a))-(\Lambda(z-a))^T(\Lambda(z-a))\right)} \\
&= \frac{1}{1+\exp\left(4(\Lambda a)^T(\Lambda z)\right)} \\
&= \frac{1}{1+\exp\left(4a^T\Sigma^{-1}(x-\frac{\mu^0+\mu^1}{2})\right)} \\
&= g(\theta^T x') \quad \text{where } \theta^T = [(\mu^0-\mu^1)^T\Sigma^{-1}(\mu^0+\mu^1), 2(\mu^1-\mu^0)^T\Sigma^{-1}] \text{ and } x' = [1,x]
\end{aligned} \tag{12}$$

□

3 Softmax regression and OVA logistic regression

3.1 Implementing the loss function for softmax regression (naive version)

3.2 Implementing the gradient of loss function for softmax regression(naive version)

Implemented the `softmax_loss_naive` method in file `softmax.py`:

```
for i in range(0,m):
p=np.zeros(max(y)+1)
for j in range(0,max(y)+1):
po=0
for jj in range(0,max(y)+1):
po=po+np.exp(theta[:,jj].dot(X[i,:])-theta[:,j].dot(X[i,:]))
```

```
p[j]=1/po
grad[:,j]-=X[i,:]*(float(y[i]==j)-p[j])/m
J=J+np.log(p[y[i]])
J=-J/m+reg*np.sum(theta**2)
grad=grad+2*theta*reg
```

result:

```
Training data shape: (49000, 3072)
Validation data shape: (1000, 3072)
Test data shape: (10000, 3072)
Training data shape with bias term: (49000, 3073)
Validation data shape with bias term: (1000, 3073)
Test data shape with bias term: (10000, 3073)
loss: 2.33181510664 should be close to 2.30258509299
numerical: 1.846291 analytic: 1.846291, relative error: 1.620672e-08
numerical: 0.402461 analytic: 0.402461, relative error: 1.300510e-07
numerical: 2.983793 analytic: 2.983793, relative error: 9.064330e-09
numerical: 0.277037 analytic: 0.277037, relative error: 7.767378e-08
numerical: 1.066744 analytic: 1.066744, relative error: 5.981913e-08
numerical: -0.718366 analytic: -0.718366, relative error: 6.584340e-08
numerical: -0.298495 analytic: -0.298495, relative error: 1.193483e-07
numerical: 2.824531 analytic: 2.824531, relative error: 2.177955e-08
numerical: -0.617456 analytic: -0.617456, relative error: 1.193407e-08
numerical: 0.150777 analytic: 0.150777, relative error: 5.651458e-08
```

It performs as expected.

3.3 Implementing the loss function for softmax regression (vectorized version)

3.4 Implementing the gradient of loss function for softmax regression(vectorized version)

Implemented the `softmax_loss_vectorized` method in file `softmax.py`:

```
xt=X.dot(theta)
Pt=np.exp(xt-np.max(xt,1).reshape([m,1]).dot(np.ones([1,theta.shape[1]])))
P=Pt/Pt.sum(1).reshape([m,1]).dot(np.ones([1,theta.shape[1]]))
J=-1.0/m*np.sum(np.multiply(np.log(P),convert_y_to_matrix(y)))+reg*np.sum(theta**2)
grad=-1.0/m*X.T.dot((convert_y_to_matrix(y)-P))+2*theta*reg
```

result:

```
naive loss: 2.331815e+00 computed in 2945.336793s
vectorized loss: 2.331815e+00 computed in 7.681536s
```

Loss difference: 0.000000
Gradient difference: 0.000000

we can see vectorized method is about 400 times faster then using for-loop because numpy has optimation for operating matrices, and it can get the same result.

3.5 Implementing mini-batch gradient descent

Implemented `train` and `predict` method of `SoftmaxClassifier` class in file `softmax.py`.

index=np.random.choice(range(0,len(y)),size=batch_size)
X_batch=X[index,:]
y_batch=y[index]

self.theta-=grad*learning_rate

y_pred=np.argmax(X.dot(self.theta),1)

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

3.7 Training a softmax classifier with the best hyperparameters