Solution to Homework 2

Shoeb Mohammed and Zhuo Chen

February 21, 2016

1 MAP and MLE parameter estimation

1.1

$$Likelihood(\theta|D) = P(D|\theta) = \theta^n(1-\theta)^{m-n} \tag{1}$$

Where n is the times of $X = 1$ in D.
According to MLE method we need to derive the maximum point of $Likelihood(\theta|D)$:

$$Likelihood'(\theta|D) = n\theta^{n-1}(1-\theta)^{m-n} - (m-n)\theta^n(1-\theta)^{m-n-1} \tag{2}$$

set the derivative equals to 0, we get

$$Likelihood'(\theta|D) = n\theta^{n-1}(1-\theta)^{m-n} - (m-n)\theta^n(1-\theta)^{m-n-1} = 0 \tag{3}$$

$$\theta = 0 \quad | \quad \theta = 1 \quad | \quad \theta = \frac{n}{m} \tag{4}$$

Since we know $0 <= \theta <= 1$, it's easy to verify that $\theta = \frac{n}{m}$ is the maximum point. So $\theta = \frac{n}{m}$ is a MLE for θ.

1.2

According to Bayesian statistics, we have:(since P(D) is a Constant)

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \propto Beta(\theta|a,b)Likelihood(\theta|D) \propto \theta^{n+a-1}(1-\theta)^{m-n+b-1} \tag{5}$$

It's very easy to show that the MAP estimation of θ is:

$$\theta = \frac{n+a-1}{m+a+b-2} \tag{6}$$

When $a = b = 1$, it equals to MLE estimation.

2 Logistic regression and Gaussian Naive Bayes

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