HW3. Logistic Regression

Jungpyo Hong (jphong@ai.kaist.ac.kr)

April 10, 2016

1 Introduction

In this assignment, you will design the logistic regression classifier on two image data sets, handwritten digit images and face images, which are used in HW 1 and 2.

2 Project Instruction

2.1 NumPy and Anaconda

In this assignment, we will use NumPy and Python 2.7, extending the previous homework base codes. Matrix operations including addition and multiplication are heavily used in implementation of logistic regression, so NumPy, which is scientific computing library on Python, will be helful. Also, as the same as before, Anaconda is stronly recommended if you don't have NumPy library yet. Following list is the link if you want to download programs for this homework:

- Python 2.7.11: https://www.python.org/
- NumPy 1.10.4: http://www.numpy.org/
- Anaconda: https://www.continuum.io/downloads/

2.2 Logistic Regression

Unlike with previous works (naive bayes classifier and gaussian discreminant analysis), logistic regression which we are dealing with is the discreminative approach for classification. Logistic regression tries to fit conditional probability $\Pr(y|\mathbf{x})$ directly from training data. Because the handwritten data set we are used is multi-class classification problems, the conditional probability may be modeled as a softmax function:

$$\Pr(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x})}$$

Given the training set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}, \mathbf{y} = \{y_1, y_2, ..., y_N\}$, while N is a number of training data, the conditional probability for one training data $\mathbf{x}_i$ to be classified to c can be written as μ_{ic}:

$$\Pr(y = c|\mathbf{x}_i, \mathbf{W}) = \mu_{ic} = \frac{\exp(\mathbf{w}_c^T \mathbf{x}_i)}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x}_i)}$$

The likelihood funtion for one training data $\mathbf{x}_i$ and y_i is:

$$\begin{aligned}\Pr(y_i|\mathbf{x}_i, \mathbf{W}) &= \Pr(y = y_i|\mathbf{x}_i) = \mu_{iy_i} \\ &= \mu_{i1}^0 \mu_{i2}^0 ... \mu_{iy_i}^1 ... \mu_{iy_N}^0 \\ &= \prod_{c=1}^{C} \mu_{ic}^{y_{ic}}\end{aligned}$$

, where $y_{ic} = \mathbb{I}(y_i = c)$. So, likelihood function given training set $\mathbf{X}$ and $\mathbf{y}$ is

$$\Pr(\mathbf{y}|\mathbf{X}, \mathbf{W}) = \prod_{i=1}^{N} \prod_{c=1}^{C} \mu_{ic}^{y_{ic}}$$

Its negative log likelihood (NLL) is so that:

$$\begin{aligned}NLL(\mathbf{W}) &= -\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{W}) = -\log \prod_{i=1}^{N} \prod_{c=1}^{C} \mu_{ic}^{y_{ic}} \\ &= -\sum_{i=1}^{N} \left[\sum_{c=1}^{C} y_{ic} \mathbf{w}_c^T \mathbf{x}_i - \log \sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x}_i) \right]\end{aligned}$$

The objective of logistic regression is to find a maximum likelihood estimation (MLE) of the parameter $\hat{\mathbf{W}}$ which maximize the likelihood function, or minimize the negative log likelihood function.

2.2.1 Regularization

Regularization aims to keep the size of parameters to be small. By discouraging the coefficients from reaching large values, we can prevent a phenomenon called over-fitting, which means that the model tries to fit only the training data so perfectly but this leads to a result of poor performance on other, general case of data. There are many ways to do regularization on logistic regression, but in this assignment we use l_2 regularization. l_2 regularization tries to reduce the l_2 norm of our parameter, $\|\mathbf{w}\|_2^2$:

$$\|\mathbf{w}\|_2^2 = \mathbf{w}^T \mathbf{w} \qquad (\mathbf{w} \text{ is vector})$$

$$\|\mathbf{W}\|_2^2 = \sum_{i=1}^{N} \sum_{j=1}^{M} (m_{ij}^2) \qquad (\mathbf{W} \text{ is (N, M) size matrix})$$

We can adapt this regularization to our negative log likelihood to penalize the parameters. It can be done by minimize the new objective function $E(\mathbf{W})$:

$$E(\mathbf{W}) = NLL(\mathbf{W}) + \lambda \|\mathbf{W}\|_2^2$$

λ is called regularization parameter, which controls the tradeoff between two goals:

- Fit the training set well
- Keep parameter value small

Note that if $\lambda = 0$, then $E(\mathbf{W}) = NLL(\mathbf{W})$ so the objective is just minimize NLL and the resulting parameters are MLE solution (Fit the training set well). On the other hand, if $\lambda \to \infty$, the objective function becomes $\|\mathbf{W}\|_2^2$, and the resulting parameters becomes zero (Keep parameter value small).

2.2.2 Gradient Descent

To find the global minimum point of our objective function, the most general way is to find a point $\hat{\mathbf{W}}$ which makes the gradient $\bigtriangledown E(\hat{\mathbf{W}}) = 0$:

$$\bigtriangledown_{\mathbf{w}_c} E(\mathbf{W}) = \sum_{i=1}^{N} (\mu_{ic} - y_{ic})\mathbf{x}_i + \lambda \mathbf{w}_c$$

However, there is no a closed-form solution for the logistic regression case, because of the non-linearity of the softmax function. One way to find the minimum point of objective function in such a case is a gradient descent method. In gradient descent, the parameters are updated iteratively with follwing rule:

$$\begin{aligned}
\mathbf{w}_c^{(new)} &= \mathbf{w}_c^{(old)} - \alpha \bigtriangledown_{\mathbf{w}_c} E(\mathbf{W}^{(old)}) \\
&= \mathbf{w}_c^{(old)} - \alpha \left[\sum_{i=1}^{N} (\mu_{ic} - y_{ic})\mathbf{x}_i + \lambda \mathbf{w}_c^{(old)} \right]
\end{aligned}$$

Here, α is called a learning rate. Because $E(\mathbf{W})$ is known to be convex, we can get the global minima of $E(\mathbf{W})$ repeatedly following gradient descent update, if α is small enough.
It is convention not to penalize the bias parameter $\mathbf{b}$. One reason to omit the bias parameter to regularize is that the origin of the target variable can effects to the result if we include it. (ex. adds c to all targets y_i leads $\mathbf{b}$ increasing by c as well, but not in regularized $\mathbf{b}$)
The result update rule is as follows:

$$\begin{aligned}
\mu_{ic} &= \frac{\exp\left(\mathbf{w}_c^{(old)T}\mathbf{x}_i + b_c^{(old)}\right)}{\sum_{c'=1}^{C} \exp\left(\mathbf{w}_{c'}^{(old)T}\mathbf{x}_i + b_{c'}^{(old)}\right)} \\
\bigtriangledown_{\mathbf{w}_c} E &= \left[\sum_{i=1}^{N} (\mu_{ic} - y_{ic})\mathbf{x}_i + \lambda \mathbf{w}_c^{(old)} \right] \\
\bigtriangledown_{b_c} E &= \left[\sum_{i=1}^{N} (\mu_{ic} - y_{ic}) \right] \\
\mathbf{w}_c^{(new)} &= \mathbf{w}_c^{(old)} - \alpha \bigtriangledown_{\mathbf{w}_c} E \\
b_c^{(new)} &= b_c^{(old)} - \alpha \bigtriangledown_{b_c} E
\end{aligned}$$

2.3 What to Do

You must fill in the portion of **logisticRegression.py** during the assignment. You will fill in the following functions for this assignment:

- *calculateCostAndGradient*
- *updateWeight*
- *validateWeight*
- *calculateConditionalProbability*

In *calculateCostAndGradient*, calculate the negative log likelihood and the gradient of the parameters based on current parameter values. Method *initializeWeight* initialized the parameters, giving some randomness on the value, and stored on the variable W and b. You will use this variables as current estimated parameters of gradient descent. The gradient value calculated in this method will be used at the next step, *updateWeight*.

In *updateWeight*, do the gradient descent by updating the parameter with calcualted gradients. Not only gradients you made before, you can get learning rate and regularization parameter in this method. Implement the gradient descent update using the given parameters
In *validateWeight*, you must choose the best parameters based on validation accuracy. Validation set in this assignment is used to choose the best model parameters (learning rate and regularization parameter). If the calcualted validation accuracy is higher then before, you must save the current parameter W and b to the bestParam variable. bestParam will be used to calculate $\Pr(y|\mathbf{x})$ while testing.
In *calculateConditionalProbability*, calculate the conditional probability $\Pr(y|\mathbf{x})$ for the input testing data $\mathbf{x}$. You must use the parameters which shows best performance on the validation set. If you implement *validateWeight* correctly, these will be stored on bestParam.

2.4 Some Hints

- You can check the negative log likelihood value to ensure that your graident implementation is right. For sufficiently small learning rate, the cost should be decreased every epoch (in train method)
- Note that gradient descent method claims the the learning rate should be 'small enough'. If the learning rate is too big, the gradient desecent 'overshoot' the minimum point and will be diverged. (https://wingshore.wordpress.com/2014/11/19/linear-regression-in-one-variable-gradient-descent-contd/)
- Accuracy can be obtained by applying numpy argmax function to your conditional probability. ($\arg\max_c \Pr(y = c|\mathbf{x})$) You can check how to calculate validation/test accuracy in **dataClassifier.py** and *classify* in **logisticRegression.py**.
- On the handwritten image dataset, logistic regression shows 78% accuracy on validation set and 79% on test set.
- DO NOT use any scientific library that abbreviate your implementation(i.e. scikit-learn)

2.5 What to Submit

Please submit **logisticRegression.py** file **only**. Any late submissions will not be accepted.

2.6 How to Run the Code

To try out the classification pipeline, run **dataClassifier.py** from the command line. This will classify the digit data using the default classifier (mostFrequent) which blindly classifies every example with the most frequent label.

```
python dataClassifier.py
```

To activate the logistic regression classifier, use -c logisticRegression, or -c lr:

```
python dataClassifier.py -c lr
```

To run on the face recognition dataset with different training data size, use -d faces and -t numTraining

```
python dataClassifier.py -c lr -d faces
```

```
python dataClassifier.py -c lr -t 1000
```