Project-2 of "introduction to Statistical Learning and Machine Learning"

Yanwei Fu

October 27, 2017

Abstract

(1) This is the second project of our course. The project is released on Oct 27, 2017. The deadline is 5:00pm, Nov 19, 2016. Please send the report to 582195191@qq.com, or 17210240267@fudan.edu.cn.

(2) The goal of your write-up is to document the experiments you've done and your main findings. So be sure to explain the results. The report can be written by Word or Latex. Generate a single pdf file of your mini-projects and turned in along with your code. ackage your code and a copy of the write-up pdf document into a zip or tar.gz file called Project2-*your-student-id*_your_name.[zip|tar.gz]. Only include functions and scripts that you modified. Also put the names and Student ID in your paper. To submit the report, email the pdf file to 582195191@qq.com, or 17210240267@fudan.edu.cn.

(3) About the deadline and penalty. In general, you should submit the paper according to the deadline of each mini-project. The late submission is also acceptable; however, you will be penalized 10% of scores for each week's delay.

(4) Note that if you are not satisfied with the initial report, the updated report will also be acceptable given the necessary score penalty of late submission. The good thing is that we will compare and choose the higher score from several submissions as your final score of this project.

(5) The problems in Sec. 3 are extra-bonus, which means that you can get additional awards toward your scores of this project; yet your score won't get punished if you just skip this problem.

(6) OK! That's all. Please let me know if you have any additional doubts of this project. Enjoy!

1 (30 points) Logistic Regression

1.1 (10 points) Bayes' Rule

Suppose you have a D-dimensional data vector $x = (x_1, \cdots, x_D)^T$ and an associated class variable $y \in \{0, 1\}$ which is Bernoulli with parameter α(i.e. $p(y = 1) = \alpha$ and $p(y = 0) = 1 - \alpha$). Assume that the dimensions of x are conditionally independent given y, and that the conditional likelihood of each x_i is Gaussian with μ_{i0} and μ_{i1} as the means of the two classes and σ_i as their shared standard deviation.

Use Bayes' rule to show that $p(y = 1 \mid x)$ takes the form of a logistic function:

$$p(y = 1 \mid x) = \sigma\left(w^T x + b\right) = \frac{1}{1 + exp\left(-\Sigma_{i=1}^{D} w_i x_i - b\right)}$$

Derive expressions for the weights $\mathbf{w} = (w_i, \cdots, w_D)^T$ and the bias b in terms of the parameters of the class likelihoods and priors (i.e., μ_{i0}, μ_{i1},σ_i and α).

1.2 (10 points) Maximum Likelihood Estimation

Now suppose you are given a training set $D = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \cdots, \left(\mathbf{x}^{(N)}, y^{(N)}\right)\right\}$. Consider a binary logistic regression classifier of the same form as before,

$$p\left(y = 1 \mid \mathbf{x}^{(n)}, \mathbf{w}, b\right) = \sigma\left(\mathbf{w}^T \mathbf{x}^{(n)} + b\right) = \frac{1}{1 + exp\left(-\Sigma_{i=1}^{D} w_i x_i^{(n)} - b\right)}$$

Derive an expression for the negative log-likelihood ($E(\mathbf{w}, b)$) of $y^{(1)}, \cdots, y^{(N)}$ given $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}$ and the model parameters, under the i.i.d. assumption. Then derive expressions for the derivatives of E with respect to each of the model parameters.

1.3 (10 points) L2 Regularization

Now assume that a Gaussian prior is placed on each element of $\mathbf{w}$, and b such that $p(w_i) = \mathcal{N}(w_i \mid 0, 1/\lambda)$ and $p(b) = \mathcal{N}(b \mid 0, 1/\lambda)$. Derive an expression that is proportional to $p(\mathbf{w}, b \mid \mathcal{D})$, the posterior distribution of $\mathbf{w}$ and b, based on this prior and the likelihood defined above. The expression you derive must contain all terms that depend on $\mathbf{w}$ and b. Define $L(\mathbf{w}, b)$ to be the negative logarithm of the expression you derive. Show that $L(\mathbf{w}, b)$ takes the following form:

$$L(\mathbf{w}, b) = E(\mathbf{w}, b) + \frac{\lambda}{2}\Sigma_{i=1}^{D} w_i^2 + \frac{\lambda}{2} b^2 + C(\lambda)$$

where $C(\lambda)$ is a term that depends on λ but not on either $\mathbf{w}$ or b . What are the derivatives of L with respect to each of the model parameters?

2 (70 points) Digit Classification

In this section, you will compare the performance and characteristics of different classifiers, namely k-nearest neighbours, logistic regression, and naive Bayes. You will extend the provided code and experiment with these extensions. Note that you should understand the code first instead of using it as a black box. Both Matlab and Python versions (Python 2.7 with both the Numpy and Matplotlib packages installed) of the code have been provided. You are free to work with whichever you wish. The data you will be working with are hand-written digits, 4s and 9s, represented as $28x28$ pixel arrays. There are two training sets: *mnist_train*, which contains 80 examples of each class, and mnist train small, which contains 5 examples of each class. There is also a validation set mnist valid that you should use for model selection, and a test set *mnist_test*. Code for visualizing the datasets has been included in *plot_digits*.

2.1 (10 points) k-Nearest Neighbours

Use the supplied kNN implementation to predict labels for *mnist_valid*, using *mnist_train* as the training set. Write a script that runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$ and plots the classification rate on the validation set (number of correctly predicted cases, divided by total number of data points) as a function of k.

Comment on the performance of the classifier and argue which value of k you would choose. What is the classification rate for $k^\star$, your chosen value of k? Also compute the rate for $k^\star + 2$ and $k^\star - 2$. Does the test performance for these values of k correspond to the validation performance? (Note that in general you shouldn't peek at the test set multiple times, but for the purposes of this question it can be an illustrative exercise.) Why or why not?

2.2 (20 points) Logistic regression

Look through the code in *logistic_ regression_ template* and *logistic*. Complete the implementation of logistic regression by providing the missing part of *logistic*. Use *checkgrad* to make sure that your gradients are correct.

Run the code on both *mnist_ train* and *mnist_ train_ small*. You will need to experiment with the hyperparameters for the learning rate, the number of iterations (if you have a smaller learning rate, your model will take longer to converge), and the way in which you initialize the weights. If you get Nan/Inf errors, you may try to reduce your learning rate or initialize with smaller weights. Report which hyperparameter settings you found worked the best and the final cross entropy and classification error on the training, validation and test sets. Note that you should only compute the test error once you have selected your best hyperparameter settings using the validation set.

Next look at how the cross entropy changes as training progresses. Submit 2 plots, one for each of *mnist_ train* and *mnist_ train_ small*. In each plot show two curves: one for the training set and one for the validation set. Run your code several times and observe if the results change. If they do, how would you choose the best parameter settings?

2.3 (20 points) Penalized logistic regression

Implement the penalized logistic regression model you derived in 1.3 by modifying *logistic* to include a regularizer. Call the new function *logistic_ pen*. You should only penalize the weights and not the bias term, as it only controls the height of the function but not its complexity. Note that you can omit the $C(\lambda)$ term in your error computation, since its derivative is 0 w.r.t. the weights and bias. Use *checkgrad* to verify the gradients of your new *logistic_ pen* function.

Repeat part 2.2, but now with different values of the penalty parameter λ. Try $\lambda \in \{0.001, 0.01, 0.1, 1.0\}$. At this stage you should not be evaluating on the test set as you will do so once you have chosen your best λ.

To do the comparison systematically, you should write a script that includes a loop to evaluate different values of λ automatically. You should also re-run logistic regression at least 10 times for each value of λ with the weights randomly initialized each time.

So you will need two nested loops: The outer loop is over values of λ. The inner loop is over multiple re-runs. Average the evaluation metrics (cross entropy and classification error) over the different re-runs. In the end, plot the average cross entropy and classification error against λ. So for each of *mnist_ train* and *mnist_ train_ small* you will have 2 plots: one plot for cross entropy and another plot for classification error. Each plot will have two curves: one for training and one for validation.

How do the cross entropy and classification error change when you increase λ? Do they go up, down, first up and then down, or down and then up? Explain why you think they behave this way. Which is the best value of λ, based on your experiments? Report the test error for the best value of λ.

Compare the results with and without penalty. Which one performed better for which data set? Why do you think this is the case?

2.4 (15 points) Naive Bayes

In this question you will experiment with a binary naive Bayes classifier. In a naive Bayes classifier, the conditional distribution for example $\mathbf{x} \in \mathbb{R}^d$ to take on class c (out of K different classes) is defined by

$$p(c \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid c)\, p(c)}{\Sigma_{k=1}^{K} p(\mathbf{x} \mid k)\, p(k)}$$

where $p(\mathbf{x} \mid c) = \prod_{i=1}^{d} p(x_i \mid c)$ according to the naive Bayes assumption. In this question, we model $p(x_i \mid c)$ as a Gaussian for each i as

$$p\left(x_i \mid c\right) = \mathcal{N}\left(x_i \mid \mu_{ic}, \sigma_{ic}^2\right) = \frac{1}{\sqrt{2\pi\sigma_{ic}^2}} exp\left(-\frac{\left(x_i - \mu_{ic}\right)^2}{2\sigma_{ic}^2}\right)$$

The prior distribution $p(c)$ and parameters $\mu_c = (\mu_{1c}, \cdots, \mu_{dc})^T$, $\sigma_c^2 = \left(\sigma_{1c}^2, \cdots, \sigma_{dc}^2\right)$ for all c are learned on a training set using maximum likelihood estimation.

Code for training this binary naive Bayes classifier is included. The main components are:

MATLAB

***train_nb.m*:** trains a naive Bayes classifier given some data.

***test_nb.m*:** tests a trained naive Bayes classifier on some test digits.

Python

***nb.py*:** includes code to train and test naive Bayes classifiers.

You are required to fill in run nb.m in MATLAB or the *main* method of *nb.py* in Python to complete the pipeline of training, testing a naive Bayes classifier and visualize learned models. The code you need to fill in should be less than 10 lines. Report the training and test accuracy using the naive Bayes model, and show the visualization of the mean and variance vectors μ_c and σ_c^2 for both classes. Hint: Note that each dimension of feature is corresponding to one pixel in original image; thus you can generate the mean image of each number by re-ordering the dimensions of images. Briefly comment on the visualization results.

2.5 (5points) Compare k-NN, Logistic Regression, and Naive Bayes

Compare the results of k-NN on the digit classification task with those you got using logistic regression and naive Bayes. Briefly comment on the differences between these classifiers.

3 (Extra Bonus: 20 points)Stochastic Subgradient Methods

Stochastic gradient methods are an appealing training strategy when the number of training examples n is very large. They have appealing theoretical properties in terms of the training and testing objective, even when n is infinite. However, they can prove difficult to get working in practice, and are much reliable than the methods above. In this quesiton, you'll explore how dierent techniques affect the performance of stochastic gradient method. You'll explore using different step size selection schemes and using the average of pervious iterations.

3.1 Averaging and Step-size Strategies

Consider minimizing the SVM objective function,

$$\min_w f\left(w\right) = \frac{\lambda}{2} \parallel w \parallel^2 + \frac{1}{n} \Sigma_{i=1}^n \max\left\{0, 1 - y_i w^T x_i\right\} \tag{1}$$

This objective function is λ-strongly convex but non-smooth so it is a good candidate for stochastic subgradient methods. The function *example_svm* tries to minimize this function (on a problem arising from quantum physics) using a stochastic subgradient method, using a step-size of $\frac{1}{\lambda t}$, and plots the objective function against the number of "passes" through the data as it goes. Notice that it performs terribly: it actually *increases* the objective function quite substantially on the first iteration and typically never gets down to its initial value. In this question we'll explore methods for improving the performance.

1. (4 points) In the analysis of the stochastic subgradient method (below), the convergence result is discussed below (referring to reading slides in Sec.3.2 for the average of the w^t variables rather than the final value. Make a new function, *svmAvg*, that reports the performance based on the running average of the w^t values rather than the current value. Hand in the modied part of the code and a plot of the performance with averaging.

2. (8 points) While averaging all the iterations smoothes the performance, the final performance isn't too much better. This is because it places just as much weight on the early iterations (w^0, w^1, and so on) as it does on the later iterations. A common variant is to exclude the early iterations from the average. For example, we can start averaging once we have get half way to *maxIter*. Modify your *svmAvg* code to do this, and hand in the modied parts of the code and a plot of the performance this "second-half" averaging.

3. (8 points) The modication above allows you to find a better solution than the initial w^0, and it achieves the theoretically-optimal rate in terms of the accuracy ϵ, but the practical performance is still clearly bad. Modify the *svm* function to try to optimize its performance. hand in your new code, a description of the modications you found most eective, and a plot of the performance with your modications.

3.2 Reading Material

3.2.1 Stochastic Vs. Deterministic Gradient Methods

In general, when we consider minimizing $f(x) = \frac{1}{n}\Sigma_{i=1}^{n} f(x_i)$, the *deterministic gradient method* [Cauchy, 1847] will optimize

$$x^{t+1} = x^t - \alpha_t \nabla f\left(x^t\right) = x^t - \frac{\alpha_t}{n}\Sigma_{i=1}^{n} \nabla f_i\left(x^t\right)$$

The iterative cost is linear in n; and it will converge with constant α_t or line-search. As for the *stochastic gradient method* [Robbins & Monro, 1951], the algorithm step is

1. Random selection of i_t from $\{1, 2, \cdots, n\}$;

2. Direction is an unbiased estimate of true gradient,

$$\mathbf{E}\left(f_{i_t}^{'}(x)\right) = \frac{1}{n}\Sigma_{i=1}^{n} \nabla f_i(x) = \nabla f(x)$$

The iteration cost is independent of n. The convergence requires that $\alpha_t \to 0$.

The convergence of two methods is intuitively illustrated in Fig. 1.

3.2.2 Sub-gradient of SVM

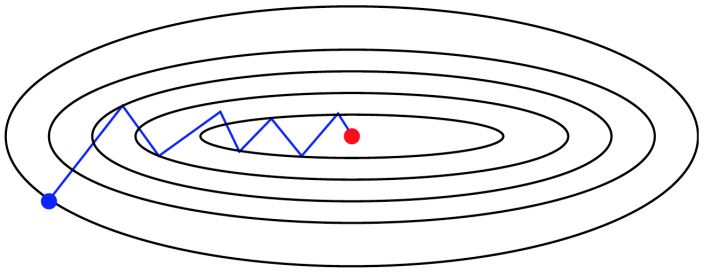
As for the SVM in Eq (1), the sub-gradient is

$$\frac{1}{n}\Sigma_{i=1}^{n} d_i + \lambda w, \quad with\, d_i = \begin{cases} -y_i x_i & if\, 1 - y_i\left(w^T x_i\right) > 0 \\ 0 & otherwise \end{cases}$$

The stochastic subgradient method could be

$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t \quad with\, g_{i_t} = \begin{cases} -y_i x_i & if\, 1 - y_i\left(w^T x_i\right) > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

- Deterministic gradient method [Cauchy, 1847]:



- Stochastic gradient method [Robbins & Monro, 1951]:

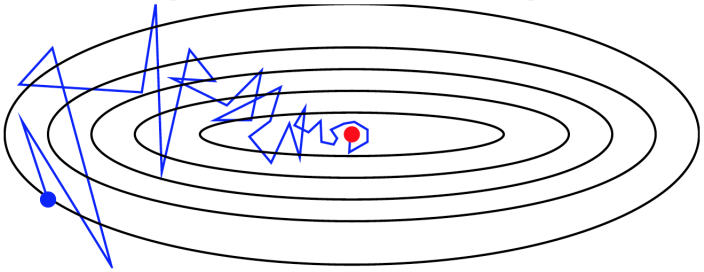


Figure 1: Determinstic gradient method Vs. Stochastic gradient method.

3.2.3 About the convergence of sub-gradient method

To get convergence, we need a *decreasing* step size in Eq (2). Note that (1) region that we converge to shrinks over time; (2) But it can't shrink too quickly or we may never reach the best solution. The classical approach to choose α_t such that

$$\Sigma_{t=1}^{\infty}\alpha_t = \infty,\ \Sigma_{t=1}^{\infty}\alpha_t^2 < \infty$$

which suggests that $\alpha_t = \mathcal{O}\left(\frac{1}{t}\right)$.
We can obtain convergence rates with decreasing steps:
if $\alpha_t = \frac{1}{\mu_t}$, we can show

$$\begin{aligned}\mathbf{E}\left[f\left(\bar{x}^t\right) - f\left(x^\star\right)\right] &= \mathcal{O}\left(\log\left(t\right)/t\right)\\ &= \mathcal{O}\left(1/t\right)\end{aligned}$$

for the average iteration $\bar{x}^t = {}^1/_k\Sigma_{k=1}^{T} x_{k-1}$.

3.2.4 Other reading material

Please read the slides "*Stochastic gradient methods for machine learning*" (attached in the file).