Report

Pingxuan Huang

11/19/2017

Contents

1 Logistic Regression

Bayes Rule

1.1

According to the question:

$$\mathrm{P}(\mathrm{x}_i \mid y=1) = \frac{1}{\sqrt{2\pi}\delta_i}\exp\{-\frac{1}{2}\frac{(x_i-\mu_{i1})^2}{\delta_i^2}\}$$
$$\mathrm{P}(\mathrm{x}_i \mid y=1) = \frac{1}{\sqrt{2\pi}\delta_i}\exp\{-\frac{1}{2}\frac{(x_i-\mu_{i0})^2}{\delta_i^2}\}$$
$$\mathrm{P}(\mathrm{x} \mid y=1) = \frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{\prod_{i=1}^{D}\delta_i}\exp\{-\frac{1}{2}\sum_{i=0}^{D}\frac{(x_i-\mu_{i1})^2}{\delta_i^2}\}$$
$$\mathrm{P}(\mathrm{x} \mid y=1) = \frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{\prod_{i=1}^{D}\delta_i}\exp\{-\frac{1}{2}\sum_{i=0}^{D}\frac{(x_i-\mu_{i0})^2}{\delta_i^2}\}$$
$$\mathrm{P}(\mathrm{y}{=}1 \mid x) = \frac{P(x|y=1)\alpha}{P(x|y=0)(1-\alpha)+P(x|y=1)\alpha}$$
$$\mathrm{P}(\mathrm{y}{=}1 \mid x) = \frac{\exp\{-\frac{1}{2}\sum_{i=0}^{D}\frac{(x_i-\mu_{i1})^2}{\delta_i^2}\}\alpha}{\exp\{-\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i-\mu_{i0})^2}{\delta_i^2}\}(1-\alpha)+\exp\{-\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i-\mu_{i1})^2}{\delta_i^2}\}\alpha}$$

$$\mathrm{P(y{=}1 \mid } x) = \frac{1}{1+\frac{1-\alpha}{\alpha}\exp\{-\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i-\mu_{i0})^2}{\delta_i^2}+\log^{1-\alpha}+\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i-\mu_{i1})^2}{\delta_i^2}-\log^{\alpha}\}}$$

$$\mathrm{P(y{=}1 \mid } x) = \frac{1}{1+\exp\{-\sum_{i=1}^{D} W_i X_i - b\}}$$

where $\mathrm{W}_i = \frac{\mu_{i1}-\mu i0}{\delta_i^2}, b = 2\log\frac{\alpha}{1-\alpha} + \frac{1}{2}\sum_{i=1}^{D}\frac{\mu_{i1}^2-\mu i0^2}{\delta_i^2}$

1.2

1.2. $E_1(w,b) = \prod_{i,y_i=1} P(y=1|x_i,w,b) \cdot \prod_{i,y_i=0} P(y=0|x_i,w,b)$

$= \prod_{i,y_i=1} P(y=1|x_i,w,b) \cdot \prod_{i,y_i=0} (1-P(y=1|x_i,w,b))$

$\ln[E_1(w,b)] = \sum_{i,y_i=1} \ln P(y=1|x_i,w,b) + \sum_{i,y_i=0} (1-y_i) P(y=0|x_i,w,b)$

$= \sum_{i=1}^{N} y_i \cdot \ln P(y=1|x_i,w,b) + (1-y_i)\ln[1-P(y=1|x_i,w,b)]$

$E(w,b) = -\ln[E_1(w,b)]$

$= -\sum_{i=1}^{N} y_i \cdot \ln P(y=1|x_i,w,b) + (1-y_i)\ln[1-P(y=1|x_i,w,b)]$

$= -\sum_{i=1}^{N} y_i \cdot \ln\left(\frac{1}{1+\exp(-w^T x-b)}\right) + (1-y_i)\cdot\ln\left(\frac{\exp(-w^T x-b)}{1+\exp(-w^T x-b)}\right)$

$= -\sum_{i=1}^{N}[(1-y_i)\cdot(-w^T x-b) - \ln(1+\exp(-w^T x-b))]$

$E(w,b) = \sum_{i=1}^{N}[(1-y_i)\cdot(w^T x+b) + \ln(1+\exp(-w^T x-b))]$

let $\hat{w} = (w;b)$; $\hat{x} = (x;1)$

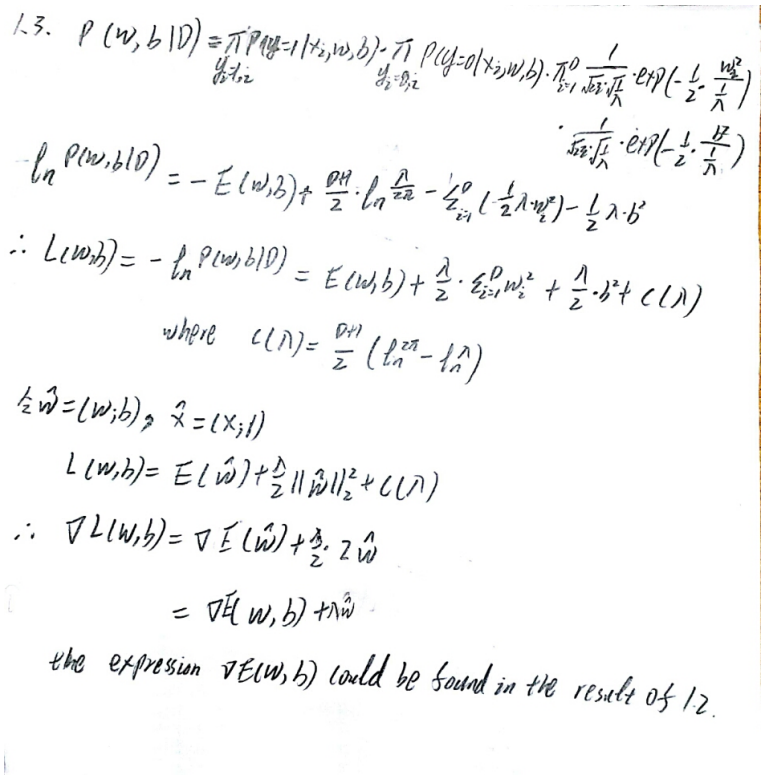
then.

$E(w,b)E(\hat{w}) = \sum_{i=1}^{n}[(1-y_i)(\hat{w}^T\cdot\hat{x}) + \ln(1+\exp(-\hat{w}^T\hat{x}))]$

$\nabla E(w,b) = \frac{\partial E(\hat{w})}{\partial(\hat{w})} = \sum_{i=1}^{n}\left[(1-y_i) - \frac{\exp(-\hat{w}^T\hat{x})}{1+\exp(-\hat{w}^T\hat{x})}\right]\cdot\hat{x}$

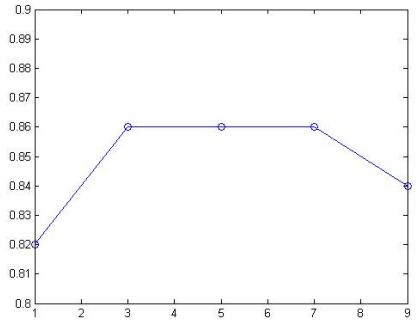
where $\hat{w} = (w;b)$, $\hat{x} = (x;1)$

1.3



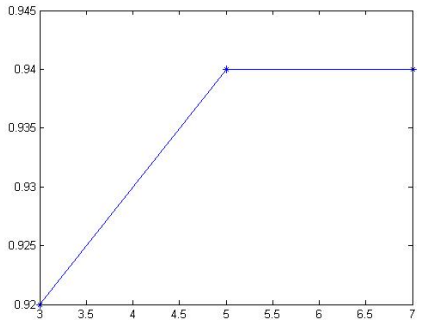
2 Digit Classification

2.1 k-Nearest Neighbours



Experiment Analysis:

As can be seen from the figure, when the value of k increases, accuracy is a process that first becomes better, then stabilizes, and then decreases. When k is 3, 5, and 7, accuracy is best (0.86) because both the median and mean of these values are 5. Therefore, when k* is 5, the stability of the result will be the best, so k* takes 5.



Experiment Analysis:

It can be seen from the figure that when k is taken as k*-2, k*, k*+2, the accuracy on the test set is 0.92, 0.94, 0.94, respectively. There is an under-fitting phenomenon (the result on the test is better than the result on the validation), indicating that the structure of test set is closer to that of train set than the validation set (the data similarity is more High), or it can be said that the test data itself is more categorizable.

2.2 Logistic regression

(I)experiment procedure:

Code for evaluating function

```
ce = sum(-log(y).*targets);
correct = sum((y >= 0.5) == targets);
frac_correct = correct/size(y ,1);
```

The lower the calculated cross entropy, the closer the model is to the real model.

Code for Logistic function:

```
y = logistic_predict(weights,data);
[N,M] = size(data);

%calculate likehood
data = [data'; ones(1,N)]'; %Expand the dimensions of the data
Y = zeros(1,N);
for i = 1:N
    tem = - data(i,:) * weights;
    tem = (1 - targets(i))*tem - log(1 + exp(tem));
    Y(i) = -tem;
end
f = sum(Y);

df = zeros(M+1 ,1);
for i = 1:N
    tem = - data(i,:) * weights;
    tem = (1 - targets(i)) - exp(tem)/(1+exp(tem));
    df = df + tem*(data(i,:))';
end
```

end

The logistic_regression_template part was finished, noting that the Checkgrad function is equivalent to calculating the partial derivative by the definition of the partial derivative.

Experiments conducted by adjusting Hyper parameters:

(II) Experimental results and analysis:

- 使用 mnist_train 进行训练

Learning rate	Number iterations	weights	Accuracy on training set(%)	Accuracy on validation set(%)
0.01	1000	使用 randn 随机生成	85	68
0.01	5000	使用 randn 随机生成	99.38	84
0.001	10000	使用 randn 随机生成	86.88	82
0.05	2000	使用 randn 随机生成	100	88
0.1	1000	使用 randn 随机生成	100	88
0.01	2000	使用 randn 随机生成/100	100	88
0.01	5000	使用 randn 随机生成/100	99.8	90
0.05	1000	0 向量	100	88

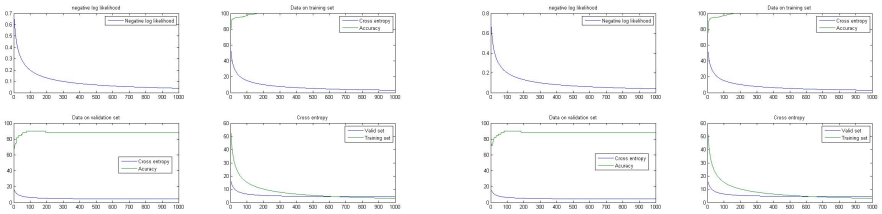
- 使用 mnist_train_small 进行训练

Learning rate	Number iterations	weights	Accuracy on training set(%)	Accuracy on validation set(%)
0.05	2000	使用 randn 随机生成 /100	100	72
0.01	5000	使用 randn 随机生成 /100	100	72
0.001	10000	使用 randn 随机生成 /100	100	68

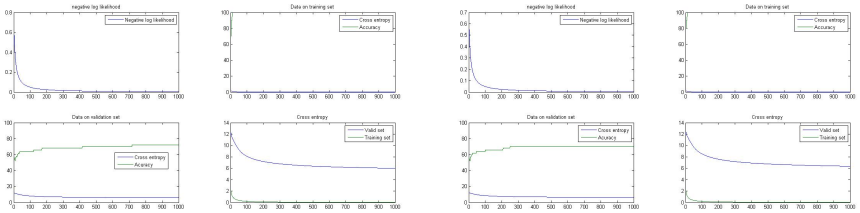
After considering the convergence speed and the correct rate on the two sets, set the parameters as:

Step size: 0.05, number of iterations: 1000

Test results:



(a) mnist_train 1 (b) mnist_train 2



(c) mnist_train_small 1 (d) mnist_train_small 2

Figure 1: Analysis of data

使用 mnist_train_small 进行训练

	cross entropy	classification error
training	0.007648	0%
validation	6.168312	28%
test	2.518233	22%

使用 mnist_train 进行训练

	cross entropy	classification error
training	3.397260	0%
validation	4.759911	12%
test	1.625505	8%

The change of Cross-Entropy is shown in Figure 1:

It can be observed that the cross entropy of the fitting result gradually reduced in both the validation set and the train set, which indicates that the degree of fitting of the model obtained by the iterative operation to the original model is gradually improved. At the same time, it can be observed that the accuracy of the validation always has a process of first rising and then falling, indicating that there is a certain over-fitting.

The result of training multiple times on the same training set and on different sets will be different, but the overall change trend is not much different from the final result (when training on the same training set), so we can make a choice according to the overall situation.

2.3 Penalized logistic regression

(I) Experimental process Code the logistic_pen function, here we need to notice that because the problem requires only penalty for weights, not for bias, so we need to change the formula deduced in 1.3, the specific method is to set the final position of 1.3 as 0.

(II) Experiment and analysis (Remarks, 20 regression for each λ in this experiment)

Experiment data:

- 使用 mnist_train_small 进行训练

Learning rate	Number iterations	weights	Cross entropy on training set	Cross Entropy on validation set	Error On Training set	Error On validation set
0.05	2500	使用	0.0079	5.9975	0	28
		randn 随	0.0104	5.9710	0	27.9
		机生成	0.0405	6.2794	0	28
		/100	0.2409	8.2753	0	26
0.01	9000	使用	0.0109	6.0814	0	28
		randn 随	0.0132	6.1026	0	28.1
		机生成	0.0409	6.3400	0	28
		/100	0.2402	8.2111	0	30
0.1	1500	使用	0.0066	5.9641	0	27.9
		randn 随	0.0093	5.9041	0	28
		机生成	0.0404	6.2712	0	28
		/100	0.2413	8.3257	0	24

Note: The classification error increases at the end of the second set (λ:0.01, validation: 9000). This should be due to insufficient fitting times, so in the subsequent experiments, The number of training increases to 14,000

- 使用 mnist_train 进行训练

Learning rate	Number iterations	weights	Cross entropy on training set	Cross Entropy on validation set	Error On Training set	Error On validation set
0.05	2500	使用	1.3971	4.7522	0	12
		randn 随	1.4102	4.7448	0	12
		机生成	1.5434	4.7013	0	12
		/100	2.9639	4.4500	0	12
0.01	14000	使用	1.3477	4.7515	0	12
		randn 随	1.3611	4.7590	0	12
		机生成	1.4968	4.6870	0	12
		/100	2.9417	4.4532	0	12
0.1	1500	使用	1.1819	4.8201	0	12
		randn 随	1.1956	4.7946	0	12
		机生成	1.3382	4.7271	0	12
		/100	2.8733	4.4164	0	12

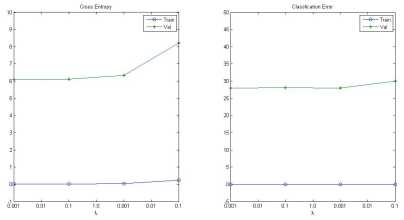
The curve is shown in Figure 2 (mnist_train_small), as shown in Figure 3 (mnist_train_small):

Data Analysis: Excluding the under-fitting situation that occurred during the second training on mnist_train_small, the overall trend is as follows:

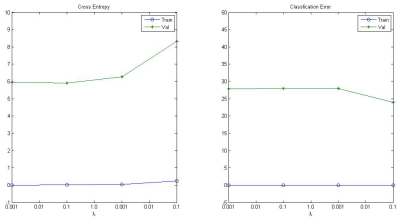
1. On the training set, when λ increases, the cross entropy that is trained by mnist_train or mnist_train_small goes up. This is due to the fact that the increase of the penalty term leads to a decrease in the degree of fitting on the training set (increased under-fitting), and what the cross entropy describe is the degree of fitting. The correctness of the classification on the training set has not changed, which indicates that the penalty has little effect on the final classification effect.

2. The results of the data on the validation set need to be discussed separately:

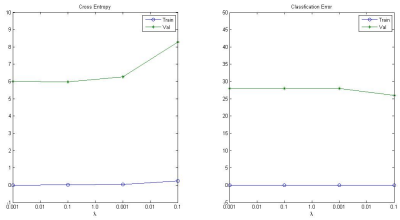
   (a) When using mnist_train set, as the λ increases, the cross entropy on the verification set keeps decreasing, which means that the addition of the penalty term alleviated the overfitting phenomenon (which is mentioned in the analysis of 2.2). However, the classification accuracy rate on the verification set has not changed, which indicates that



(a) λ:0.01, iteration:9000

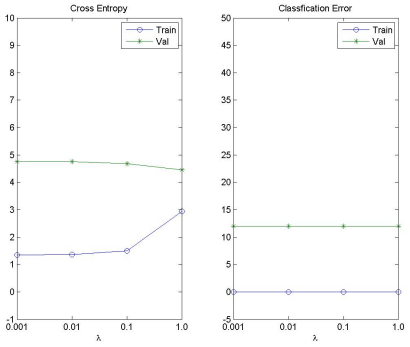


(b) λ:0.1, iteration:1500

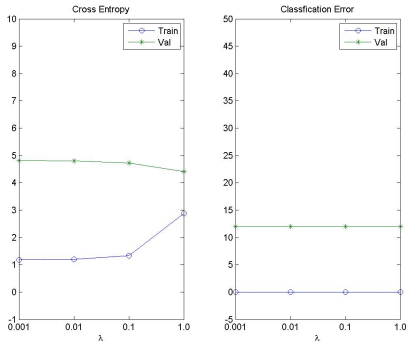


(c) λ:0.05, iteration:2500

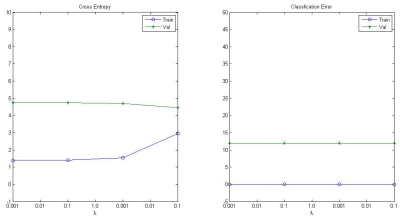
Figure 2: mnist_train_small



(a) λ:0.01, iteration:14000



(b) λ:0.1, iteration:1500



(c) λ:0.05, iteration:2500

Figure 3: mnist_train

the mitigation of this over-fitting does not have much influence on the classification results;

(b) When training with mnist_train_small, as the λ increases, the cross entropy on the verification set keeps rising. This is a very interesting phenomenon. I think there are two explanations: The verification set is more similar to mnist_train_small; (2) the original data set is too small, and a certain over-fitting does not have a big impact on the final verification. What is even more surprising is that when the lambda is equal to 1.0, the classification accuracy of the verification set is improved, which shows that the mitigation of the over-fitting phenomenon has an effect on the final classification result, which contradicts to the analysis of the cross entropy. So, we can only suppose that because the mnist_train_small is too small, few special data will have a big impact.

Considering the above experimental results, we is decided to set lambda as 1.0, and other hyperparameters is the same as before (set: 0.05, number of iterations: 2500). The classification result on the test set after selecting hyperparameters is:

- 在测试集上的分类结果为：

Training_set	cross entropy	classification error
mnist_train	1.150558	8%
Mnist_train_small	5.745978	22%

- 与不加入惩罚项时进行对比

Training_set	cross entropy	classification error	penelarization
mnist_train	1.150558	8%	√
Mnist_train_small	5.745978	22%	√
mnist_train	1.625505	8%	
Mnist_train_small	2.518233	22%	

It can be seen that before and after the penalty is added, there is no influence on the classification accuracy on the test set, but in the mnist_train training, the cross entropy of the test set reduced after the add of penalty, which indicates that the over-fitting phenomenon is obtained. At the same time, when using mnist_train_small training, the cross entropy of the test set is increased after the penalty is added. This phenomenon is still difficult to explain and can only be interpreted as the mnist_train_small data are somehow special, and its sample size is so small that magnified this particularity .

2.4 Naive Bayes

The code added is:

```
% Add your code here (it should be less than 10 lines)
% Trainning
[log_prior, class_mean, class_var] =
train_nb(train_inputs_small, train_targets_small);

% Forcast on train set
[prediction_train, accuracy_train] =
test_nb(train_inputs_small, train_targets_small,
log_prior, class_mean, class_var);
```

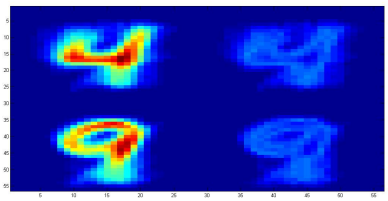


Figure 4: mnist_train

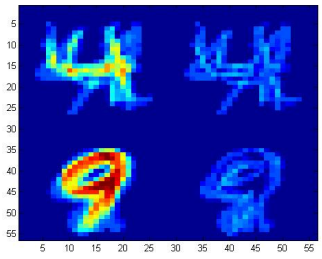


Figure 5: mnist_train_small

```
% Forcast on test set
[prediction_test, accuracy_test] =
test_nb(test_inputs, test_targets, log_prior,
 class_mean, class_var);

fprintf('Correct rate on the train set:%2.2f\nCorrect rate on the test set:%2.2
accuracy_train*100,accuracy_test*100);

Figure = zeros(56);
Figure(1:28,1:28) = reshape(class_mean( 1,:), 28, 28)';
Figure(1:28,29:56) = reshape(class_var( 1,:), 28, 28)';
Figure(29:56,1:28) = reshape(class_mean( 2,:), 28, 28)';
Figure(29:56,29:56) = reshape(class_var( 2,:), 28, 28)';
imagesc(Figure);
```

The according images are Figure4, Figure5:
The classification results are as follows:

分类准确度为：

Set	Accuracy
Mnist_train	86.25%
Test	80%
Mnist_train_small	100%
Test	66%

Visualization evaluation

1. After visualization, both the mean and the variance images can roughly be seen the shape of 4 and 9;

2. The main difference between the two mean images is in the upper part (there is a gap in the upper part of the mean image of 4), which can be predicted to be one of the main features for classification;

3. Both images are blurry in the lower part, as can be seen from the mean and variance images.This shows that the handwriting 4 and 9 are more random in the place where the stroke ends, and it is not easy to use as the discriminant criterion;

4. The Variance image is much lighter than the mean image, which means that the variance is a small value relative to the mean, and it can be found that the edge color of the variance is significantly deeper than the inside,

which indicates that the edge distribution of the handwritten number is more uncertain than the inside;

5. Here is a brief description of the classifier trained by mnist_train_small. From the visual image, we can verify the previous hypothesis. The data of 4 in the mnist_train_small is very specific (the mean/variance image is similar to 'X' instead of 4), while the data related to handwriting 9 is highly recognizable, plus the amount of data in mnist_train_small is too small. , sp it produced a special case of the previous classification.

2.5 Compare k-NN, Logistic Regression, and Naive Bayes

结果比较

Methods	Training set	Accuracy on test set
Knn	Mnist_train	94%
	Mnist_train_small	66%
Logistic	Mnist_train	92%
	Mnist_train_small	78%
Logistic with penalization	Mnist_train	92%
	Mnist_train_small	78%
NB	Mnist_train	80%
	Mnist_train_small	66%

Due to the particularity of the mnist_train_small data (previous analysis), we will mainly analysis the data obtained through the training of mnist_train. Generally speaking, superiority: KNN ¿ Logistic ¿ NB ,Specific analysis:

1. Because KNN is a nonlinear model and Logistic provides linear classification boundaries, the data results of KNN and Logistic indicate that the handwritten digit recognition task has certain nonlinearity, but this property is not obvious;

2. Both NB and KNN are nonlinear models, but the results of these two are quite different. This is because the smoothing method is needed when training a NB model, but the large amount of data in the handwritten digit recognition task is 0, which leads to simple Addition Smoothing can make a big difference to the original model;

3. The use of penalty terms has little effect on the final classification of Logistic, which means that the noise of mnist_train has little effect on over-fitting.

3 Stochastic Subgradient Methods

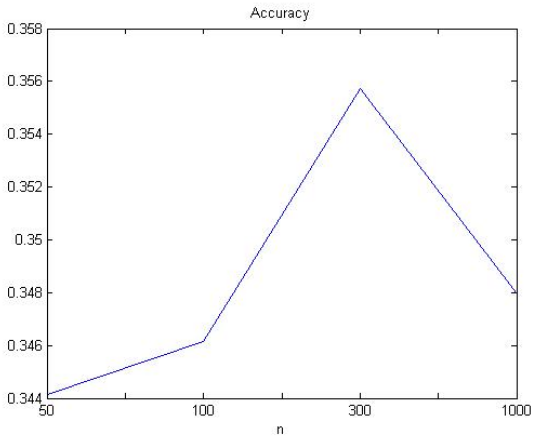
3.1

Modified Code:

```
tem = zeros(d+1,1);
for t = 1:maxIter
    % Painning part has been deleted
    i = ceil(rand*n);
    [f,sg] = hingeLossSubGrad(w,Xt,y,lambda,i);
    alpha = 1/(lambda*t);
```

```
    w = w - alpha*(sg + lambda*w);

    tem = tem + w;
end

model.w = tem / maxIter;
model.predict = @predict;
```

Result

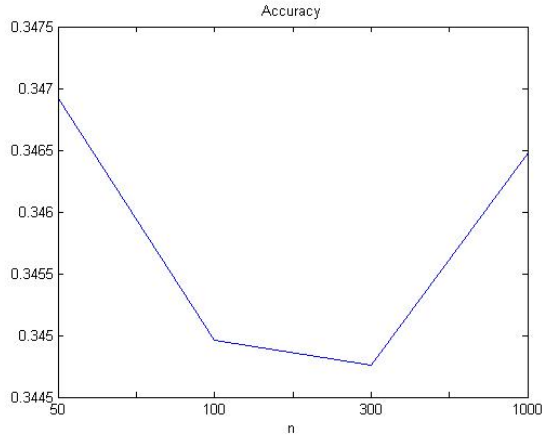


3.2

Modified Code

```
tem = zeros(d+1,1);
helf = fix(maxIter/2);
% Apply stochastic gradient method
for t = 1:maxIter
    % Painning part has been deleted
    i = ceil(rand*n);
    [f,sg] = hingeLossSubGrad(w,Xt,y,lambda,i);
    alpha = 1/(lambda*t);
    w = w - alpha*(sg + lambda*w);


    if t >= helf
        tem = tem + w;
    end
end

model.w = tem/( helf+1 );
```

Result



It can be seen that compared with the simple average method, the overall correct rate of the method is improved, but the volatility is still very large, and the training result is not simply increased when the number of iterations increases.

3.3

New Code

```
function [model] = svm_2(X,y,lambda,maxIter)
[n,d] = size(X);
X = [ones(n,1) X];
Xt = X';
w = zeros(d+1,1);
tem = zeros(d+1,1);
helf = fix(maxIter/2);

objValues = 100; %Store the best objective function value
w_tem = zeros(d+1,1); %Store the weights corresponding to the best objective fu

% Apply stochastic gradient method
for t = 1:maxIter
    i = ceil(rand*n);
    [f,sg] = hingeLossSubGrad(w,Xt,y,lambda,i);
    alpha = 1/(lambda*t);
    w = w - alpha*(sg + lambda*w);

    %Sampling
    if mod(t-1,(n/10)) == 0
        objtem = (1/n)*sum(max(0,1-y.*(X*w))) + (lambda/2)*(w'*w);
        %The value of the objective function under the weights
        if objtem < objValues
            objValues = objtem;
            w_tem = w;
        end
    end
```

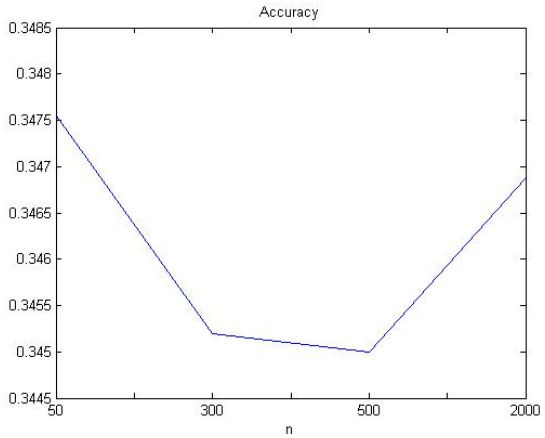
```
    %Late half-average method
    if t >= helf
        tem = tem + w;
    end
end

% Judging which is the best data collected method
tem = tem/(helf+1);
% Final iteration result
obj1 = (1/n)*sum(max(0,1-y.*(X*w))) + (lambda/2)*(w'*w);
% Results obtained by the mean method
obj2 = (1/n)*sum(max(0,1-y.*(X*tem))) + (lambda/2)*(tem'*tem);
% Results obtained using sampling
obj3 = (1/n)*sum(max(0,1-y.*(X*w_tem))) + (lambda/2)*(w_tem'*w_tem);
least = min([obj1,obj2,obj3]);
if least == obj1
    model.w = w;
else
    if least == obj2
        model.w = tem;
    else
        model.w = w_tem;
    end
end
model.predict = @predict;
end

function [yhat] = predict(model,Xhat)
[t,d] = size(Xhat);
Xhat = [ones(t,1) Xhat];
w = model.w;
yhat = sign(Xhat*w);
end

function [f,sg] = hingeLossSubGrad(w,Xt,y,lambda,i)

[d,n] = size(Xt);

% Function value
wtx = w'*Xt(:,i);
loss = max(0,1-y(i)*wtx);
f = loss;
% Subgradient
if loss > 0
    sg = -y(i)*Xt(:,i);
else
    sg = sparse(d,1);
end
end
```

Thought of new code

The purpose of gradient descent is to achieve the best classification result through minimizing the value of the objective function, so if we check whether the result is better before updating the weights (if not then there is no update) every time , then the convergence effect of the objective function should be better. But considering the existence of the locally optimal (random gradient descent is used to solve this problem), weights cannot be updated according to this standard. However, we can compare the current convergence effect with the optimal one after each iteration to determine whether to store the current weights as an alternative optimal value. If the value of the objective function under the weights is less than the current optimal value, the optimal weights are updated, otherwise only the w value is changed. Considering the number of iterations is so large, the sampling method is employed to detect the convergence effect. Specifically, each n value is averaged by extracting $10 * n$ convergence data for judgment.

While using the above sampling method, the second half averaging method of 3.2 is used, and finally the results obtained by the two methods and the last calculated w are compared to select the optimal value as the weights of the model.

Final Result:



It can be seen that there is still some volatility in accuracy. There are two reasons for this. For one thing, the uncertainty of the stochastic gradient drop itself will affect, and the second is because the number of iterations is not sufficient. When the number of iterations is sufficient, The reduction step size ($\alpha_t = \frac{1}{\lambda t}$) is supplemented by the above-mentioned sampling method and the latter half-average method, which can theoretically make the convergence effect better.