Project 3: Movement Decoding for Brain Computer Interfaces

Name : Jiaxin Chen

Email: jiaxinc@andrew.cmu.edu

Abstract

The project can classify left movement and right movement to fulfill movement decoding for brain computer interfaces, which implement interior point method, Newton method, linear search and cross validation to develop a two-class linear support vector machine in MATLAB.

1 Overview

This project requires us to train a SVM to classify the brain signals in MATLAB.

Firstly, in order to solve the convex optimization problem, we can utilize the interior point algorithm to construct logarithmic barrier. At each ieration, apply Newton method to solve an unconstrained optimization with logarithmic barrier. At each Newton iteration, choose optimal stepsize by backtracking line search.

Furthermore, we can utilize 2-level cross validation to avoid overfitting. The 1st-level implements six-fold generate the test train data separately for optimization. The 2nd-level implements five-fold cross validation to obtain the optimal λ for Newton method.

Eventually, we can apply interior point algorithm again with the optimal λ to test the accuracy of each fold as well as outputing the mean accuracy and standard derivation and ploting the channel weights.

2 Mathematical Formulation

1) Logarithmic Barrier

The decision function for the two-clas linear SVM:

$$f(X) = W^T X + C \begin{cases} \geq 0 & (ClassA) \\ < 0 & (ClassB) \end{cases} \tag{1}$$

In order to determine W and C with maximum margin, we can construct the convex optimization problem:

$$\begin{aligned} \min_{W,C,\zeta} \quad & \sum \zeta_i + \lambda \cdot W^T W \\ S.T. \quad & y_i \cdot (W^T X_i + C) \geq 1 - \zeta_i \\ & \zeta_i \geq 0 \\ & (i = 1, 2, \cdots, N) \end{aligned} \tag{2}$$

And by utilizing Logarithmic barrier, we can convert constrained optimization (2) into the unconstrained nonlinear optimization problem:

$$\begin{aligned} \min_{W,C,\zeta} \quad & \sum \zeta_i + \lambda \cdot W^T W - \frac{1}{t} \sum_{i=1}^{N} log(\zeta_i) \\ & - \frac{1}{t} \sum_{i=1}^{N} log(W^T X_i \cdot y_i + C \cdot y_i + \zeta_i - 1) \\ & t \to 0 \end{aligned} \tag{3}$$

2) Interior Point Method

In order to slove the SVM problem(2), we can implement interior point method in the following steps:

(a) Select an initial value of t and an initial guess $[W^{(0)} \quad C^{(0)} \quad \zeta^{(0)}]$:

$$\begin{aligned} & w_i^{(0)} = 1 \qquad C^{(0)} = 0 \qquad (i = 1, 2, \cdots, N) \\ & \zeta_i^{(0)} = max\{1 - y_i \cdot (W^{(0)T} X_i + C^{(0)}), 0\} + 0.001 \end{aligned} \tag{4}$$

(b) Solve the unconstrained nonlinear optimization(3) to find the optimal solution $[W^* \quad C^* \quad \zeta^*]$.

(c) Set $[W^{(0)} \quad C^{(0)} \quad \zeta^{(0)}] = [W^* \quad C^* \quad \zeta^*]$ and update $t = \beta t$ where $\beta = 15$.

(d) Repeat the step (b) and (c) until $t \geq T_{max}$ where $T_{max} = 1000000$.

3) Newton Method

In order to solve the unconstrained nonlinear optimization problem(3) inside the IPM at each ieration, we can apply New Method to solve the unconstrained nonlinear optimization problem with logarithmic barrier(3) in following steps:

(a) Start from an initial value:

$$Z^{(0)} = [W^{(0)} \quad C^{(0)} \quad \zeta^{(0)}] \tag{5}$$

(b) Compute the Gradient $\nabla f(Z)$ and Hessian matrix $\nabla^2 f(Z)$ to obtain the Newton step and decrement:

$$\begin{aligned} \Delta Z &= -\nabla^2 f(Z) \cdot \nabla f(Z) \\ \sigma &= \nabla f(Z)^T \cdot \nabla^2 f(Z)^{-1} \nabla f(Z) \\ &= -\nabla f(Z)^T \cdot \Delta Z \end{aligned} \tag{6}$$

(c) Choose the step size s by backtracking line search and update $Z = Z + s \cdot \Delta Z$.

(d) Repeat step (b) and (c) until $\sigma/2 \leq \varepsilon$ where $\varepsilon = 0.000001$.

4) **Gradient and Hessian Matrix**

In order to generate the Gradient $\nabla f(Z)$ and Hessian matrix $\nabla^2 f(Z)$ in Newton method, we can apply 1st-order derivative and 2nd-order derivative in the following steps:

(a) Obtain F from the logarithmic barrier:

$$\begin{aligned} F = &\sum \zeta_i + \lambda \cdot W^T W - \frac{1}{t}\sum_{i=1}^{N} log(\zeta_i) \\ &- \frac{1}{t}\sum_{i=1}^{N} log(W^T X_i \cdot y_i + C \cdot y_i + \zeta_i - 1) \end{aligned} \tag{7}$$

And for convenience, we can set:

$$T_i = W^T X_i \cdot y_i + C \cdot y_i + \zeta_i - 1 \tag{8}$$

(b) Obtain G by computing the 1st-order derivative:

$$\begin{aligned} G_W &= \frac{\partial F}{\partial w_j} = 2\lambda \cdot w_j - \frac{1}{t} \cdot \sum_{i=1}^{N} \frac{x_{ij} y_i}{T_i} \\ G_C &= \frac{\partial F}{\partial C} = -\frac{1}{t} \cdot \sum_{i=1}^{N} \frac{y_i}{T_i} \\ G_\zeta &= \frac{\partial F}{\partial \zeta_i} = 1 - \frac{1}{t} \cdot \frac{1}{T_i} - \frac{1}{t} \cdot \frac{1}{\zeta_i} \end{aligned} \tag{9}$$

So we can construct the Gradient:

$$G = \begin{bmatrix} G_W & G_C & G_\zeta \end{bmatrix} \tag{10}$$

(c) Obtain H by computing 3 × 3 sub Hessian blocks from the 2nd-order derivative:

$$\begin{aligned} H_{WW} &= \frac{\partial^2 F}{\partial w_j \partial w_k} = \frac{1}{t} \cdot \sum_{i=1}^{N} \frac{x_{ij} x_{ik} y_i^2}{T_i^2} \\ &\left(\frac{\partial^2 F}{\partial w_j^2} = 2\lambda \cdot w_j + \frac{1}{t} \cdot \sum_{i=1}^{N} \frac{(x_{ij} y_i)^2}{T_i^2} \right) \\ H_{WC} &= (H_{CW})^T = \frac{\partial^2 F}{\partial w_j \partial C} = \frac{1}{t} \cdot \sum_{i=1}^{N} \frac{x_{ij} y_i^2}{T_i^2} \\ H_{W\zeta} &= (H_{\zeta W})^T = \frac{\partial^2 F}{\partial w_j \partial \zeta_i} = \frac{1}{t} \cdot \frac{x_{ij} y_i}{T_i^2} \\ H_{CC} &= \frac{\partial^2 F}{\partial C^2} = \frac{1}{t} \cdot \sum_{i=1}^{N} \frac{y_i^2}{T_i^2} \\ H_{C\zeta} &= (H_{\zeta C})^T = \frac{\partial^2 F}{\partial C \partial \zeta_i} = \frac{1}{t} \cdot \frac{y_i}{T_i^2} \\ H_{\zeta\zeta} &= \frac{\partial^2 F}{\partial \zeta_i \zeta_j} = 0 \quad \left(\frac{\partial^2 F}{\partial \zeta_i^2} = \frac{1}{t} \cdot (\frac{1}{T_i^2} + \frac{1}{\zeta_i^2}) \right) \\ &(i = 1, 2, \cdots, N) \quad (j, k = 1, 2, \cdots, M) \end{aligned} \tag{11}$$

Therefore, we can construct the $(M + N + 1) \times (M + N + 1)$ Hessian matrix:

$$H = \begin{bmatrix} H_{WW} & H_{WC} & H_{W\zeta} \\ H_{CW} & H_{CC} & H_{C\zeta} \\ H_{\zeta W} & H_{\zeta C} & H_{\zeta\zeta} \end{bmatrix} \tag{12}$$

5) **Backtracking Line Search**

In order to choose the step size $s : (Z + s \cdot \Delta Z) \in dom(Z)$ in Newton method, we can apply backtracking line search in the following steps:

(a) Initialize $s = 1$.

(b) Set $s = 0.5 \cdot s$ and update the solution:

$$[W^{(k)} \quad C^{(k)} \quad \zeta^{(k)}] = Z + s \cdot \Delta Z \tag{13}$$

(c) Repeat step (b) until satisfying:

$$\begin{cases} W^{(k)T} X_i \cdot y_i + C^{(k)} \cdot y_i + \zeta_i^{(k)} - 1 > 0 \\ \zeta_i^{(k)} > 0 \qquad (i = 1, 2, \cdots, N) \end{cases} \tag{14}$$

6) **Two-level Cross Validation**

(a) 1st-level six-fold cross validation

For the 120×2 trails data set, we can divide them into six folds as 100×2 trails training data and 20×2 trails testing data per fold. And we can test the accuracy by the six-fold cross validation and calculate the mean accuracy and standard deviation:

$$\begin{aligned} \bar{Ac} &= \sum_{i=1}^{6} Ac(i)/6 \\ stdAc &= \sqrt{\frac{1}{6}\sum_{i=1}^{6} [Ac(i) - \bar{Ac}]^2} \end{aligned} \tag{15}$$

(b) 2nd-level five-fold cross validation

Inside the 100×2 trails training data, we can divide them into five folds as 80×2 trails training data and 20×2 trails testing data per fold. So we can utilize five-fold cross validation to determine the optimal λ of SVM:

$$\lambda \in \{0.01, 1, 100, 10000\} \tag{16}$$

3 Experimental Results

1) For the first training fold, the values of |W| with top 5 largest magnitudes and C is shown as Table 1:

Table 1: |W| and C in the first training fold

|W|	feaSubEOvert	feaSubEImg
1	0.0011	0.0309
2	8.2125 $\times 10^{-4}$	0.0235
3	7.1841 $\times 10^{-4}$	0.0192
4	6.3459 $\times 10^{-4}$	0.0155
5	5.8047 $\times 10^{-4}$	0.0147
C	-1.1329	-30.0114

And we can visualize the values of W as Figure 1 and Figure 2:

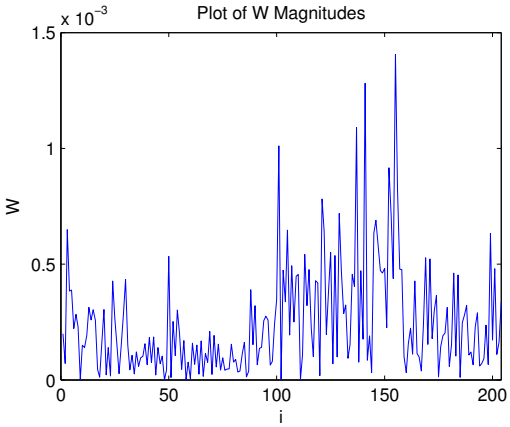

Figure 1: Plot of W for feaSubEOvert.mat

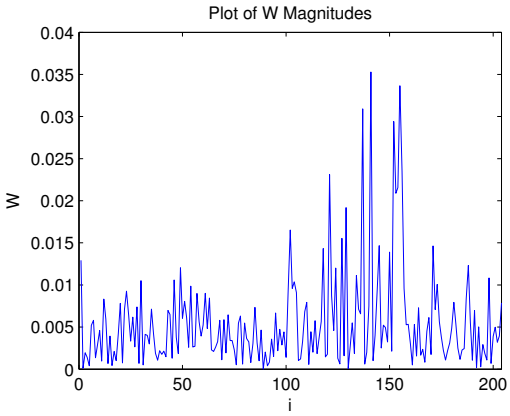

Figure 2: Plot of W for feaSubEImg.mat

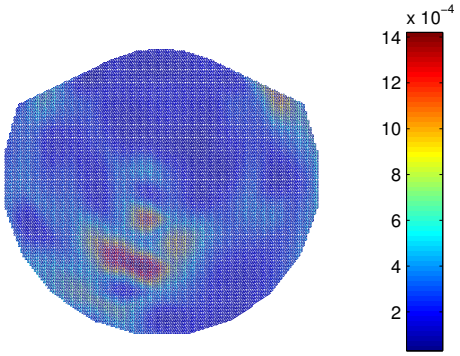

Figure 3: Channel Weights Plot of feaSubEOvert.mat

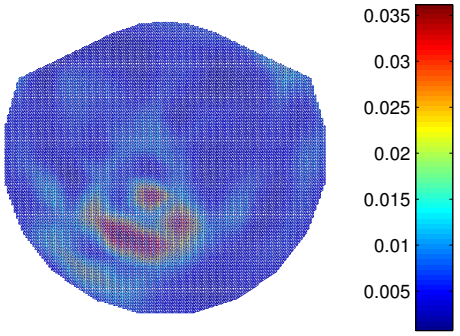

Figure 4: Channel Weights Plot of feaSubEImg.mat

Table 2: Test Accuracy of Each Fold

	feaSubEOvert	feaSubEImg
Fold 1	0.9000	0.8250
Fold 2	1.0000	0.8000
Fold 3	0.9750	0.8000
Fold 4	1.0000	0.9750
Fold 5	0.9000	0.9000
Fold 6	0.9250	0.9250

2) For the first training fold, the channel weights plot is shown as Figure 3 and Figure 4:

3) The test accuracy of each fold is shown as Table 2:

4) The mean accuracy and standard deviation of all folds is shown as Table 3:

Table 3: Mean Accuracy and Standard Deviation

	feaSubEOvert	feaSubEImg
meanAc	0.9500	0.8708
stdAc	0.0474	0.0732

5) The optimal λ of each fold is shown as Table 4:

Table 4: Optimal λ of Each Fold

	feaSubEOvert	feaSubEImg
Fold 1	100	0.01
Fold 2	10000	10000
Fold 3	100	1
Fold 4	1	1
Fold 5	100	0.01
Fold 6	100	100

4 Discussion

1) Factors that may impact classification accuracy

   (a) The better λ can achieve better SVM classification accuracy. The larger λ is, the more errors the SVM will allow in the training data. The smaller λ is, the easier over-fitting will occur due to the fewer errors that the SVM will allow. With better optimal λ, the unconstrained nonlinear optimization problem (3) can converge more accurately.

   (b) The initial parameters can impact SVM accuracy. It's important to choose the suitable intial parameters such as β, T_{max} and tolerance, which can change convergence threshold, reduce iterations and improve the accuracy dramatically.

   (c) The features of input data is also important can also affect classification accuracy. The more distinct features, the easier the SVM can classify the test data, which can achieve better accuracy.

2) Limits or problems with my approach

As Table 3, my programm achieve relatively higher mean accuracy and better standard deviation for `feaSubEOvert.mat` than `feaSubImg.mat`. I think maybe `feaSubEOvert.mat` has more distinct and confused features for my SVM to classify than `feaSubImg.mat`. So my algorithm need to improve the Newton Method to classify the more complex and confused features.

3) Possible improvements that can be made

(a) We only determine the optimal λ from $\lambda \in \{0.01, 1, 100, 10000\}$ for Newton Method as Table 4, which limits the accuracy in some way. If we can set the smaller step size to generate more selection of λ instead of the given 4 λ, we can obtain the more accurate optimal λ for Newton Method, which can achieve better balance between avoiding over-fitting and reducing errors.

(b) We can implement the PCA(Principal Component Analysis) to avoid the useless features and reduce the dimensions of features as well as improve efficiency, which may be helpful to increase the accuracy of `feaSubImg.mat` significantly.

(c) With more input data, the classifier can analyze more features to classify the test dat by avoiding over-fitting better and reducing the standard deviation.

4) Anything unique I have done to improve/validate program's accuracy/efficiency

(a) Improve Efficiency

I optimize the `costFcn.m` code by utilizing lots of `bsxfun` function in matrix format computation for all the Gradient matrix and Hessian matrix rather than for loop computation.

The `bsxfun` function can Apply element-wise operation to two arrays with implicit expansion enabled, which accelerates the execution almost 10 times and achieves amazing performance. The comparison between matrix format computation and for loop computation of average execution time is shown as Table 5:

Table 5: Execution Time Comparison

Computation	feaSubEOvert	feaSubEImg
Matrix	290s	293s
For Loop	3100s	3200s

(b) Reduce Computation

In the `solveOptProb_NM.m` function, I initialize the [F, G, H], ΔZ and error by implementing the `costFcn.m` once before the backtracking line search loop, which can reduce the computation. Because the initialized error is much smaller than simply setting $err = 1$, it can reduce the iteration times dramatically. And we need invoke `solveOptProb_NM.m` in both training and testing phase, which can help improve the performance.

(c) Improve Accuracy

In the begining, I tried the given $\lambda \in \{0.01, 1, 100, 10000\}$ to obtain the optimal λ, whose results as Table 4 can be improved as discussed above. Therefore I utilize the increased step size to expand λ vector size for five-cross cross validation, which achieves the much better accuracy. I take `feaSubEOvert.mat` as the example as Table 6.

Table 6: Accuracy and Optimal λ

	Accuracy	optimal λ
Fold 1	0.9000	156.25
Fold 2	1.0000	156.25
Fold 3	0.9750	0.05
Fold 4	1.000	31.25
Fold 5	0.9500	31.25
Fold 6	0.9500	3.9063×10^3
meanAc	0.9625	
stdAc	0.0379	

Compared the Table 6 with Table 2 and Table 3, the mean accuracy of `feaSubEOvert.mat` increases and the standard deviation decreases further, which means that the more choice of λ can help the optimization convergen more accurately.

(d) Validate Accuracy

After obtaining the optimal λ, I implement the interior point method again to calculate the test accuracy of each fold in the six-fold cross validation, which validates the accuracy of my classification.

(e) Visualize Weights

In order to analyze the relationship between W and λ better to choose the optimal λ, I visualize the absolute value of W as Figure 1 and Figure 2. I observe that the |W| of `feaSubEOvert.mat` is smaller than `feaSubEImg.mat` because the optimal λ of `feaSubEOvert.mat` is larger than `feaSubEImg.mat`, which enable the SVM to allow more errors in training data, which may cause the mean accuracy of `feaSubEImg.mat` is better than `feaSubEImg.mat`.

References

[1] Shai Shalev-Shwartz; Yoram Singer; Nathan Srebro , *"Pegasos: Primal Estimated sub-GrAdient SOlver for SVM"* ICML.

[2] John C. Platt, *"Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines"*. NIPS.