Project 3: Movement Decoding for Brain Computer Interfaces

Akshay Rajhans
arajhans@ece.cmu.edu

Abstract –

In this project, we build a classifier using Support Vector Machine (SVM) for two datasets of the problem of decoding movement using brain-computer interfaces. The purpose of the SVM classifier is to classify experimental data into two categories – left movement and right movement.

The accuracy of the SVM classifier is tested using 6-fold cross-validation. In each fold, the value of lambda, a parameter deciding the relative importance between the classifier margin and the classifier (in)accuracy is chosen by doing 5-fold cross-validation. Mean and standard deviation of the test accuracy results obtained in the 6 folds is computed.

1. Overview

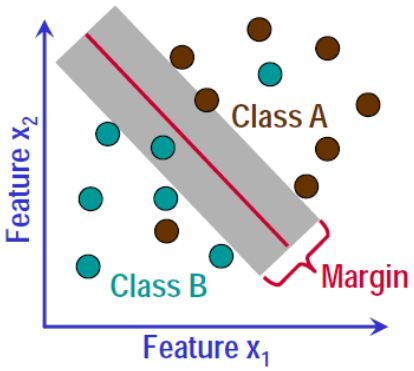
In this project we build a classifier using Support Vector Machine (SVM) for two different data sets of the experiment of movement detection using brain-computer interfaces. The purpose of the classifier is two classify the experimental data into two categories – left movement and right movement. The decision function of a general SVM classifier works by creating a linear decision function w'x+c whose value being of either sign decides which class a particular data point belongs to.

Decision Function:

$$f(X) = W^T X + C \quad \begin{cases} \geq 0 & (Class\ A) \\ < 0 & (Class\ B) \end{cases}$$

Features

The decision function is shown above. Pictorially, the classification problem is as shown below. Here, only the first two dimensions of the data are shown, but in both data sets we have M = 204 dimensions, therefore 204 weights.



Each experiment gives us a data point (a point in the M dimensional space), i.e. an M-dimensional vector. There are 120 such points of each class in either data sets.

The accuracy of the classifier is tested in by 6-fold cross-validation. $1/6^{th}$ (20x2) of the data points are saved for test set and the classifier is trained on the rest of the $5/6^{th}$ (here 100x2) points from either class.

The value of the parameter lambda can be varied to set the relative importance of the cost between the weights i.e. the classifier margin and the classifier (in)accuracy for the training set. Here different values of lambda that were tested were {0.01, 1, 100, 10000}. The best value of lambda is found by a second tier of cross-validation. Of the $5/6^{th}$ of data points available for training, $1/5^{th}$ (i.e. $1/6^{th}$ of the total points) are set aside for test set, while the remaining $4/5^{th}$ (i.e. $4/6^{th}$ of total points) are used for training set. This procedure is repeated 5 times while choosing different combinations of training and test sets. The best value of lambda is chosen as the one that results on an average the minimum number of misclassifications.

Once the lambda value is chosen, the classifier is trained again, this time on the entire training set of $5/6^{th}$ of points. Then this classifier is tested on the $1/6^{th}$ of the data points in the test set. The accuracy (% of correct classifications) are recorded and the procedure is repeated 6 times, choosing different combinations of training and test sets.

Finally, the mean and the standard deviation of the accuracies obtained in the 6 folds is calculated.

This entire process is implemented on the two data sets provided.

2. Mathematical Formulation

While training the SVM classifier, the objective is to maximize the classification margin, while minimizing the number of misclassifications. This goal can be captured using the constrained optimization problem

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

where y_i is the training label of the i^{th} point, +/-1, ξ_i is the error made in classifying the i^{th} sample point, N is the number of weights, i.e. the number of features in the data set (here 204).

The constrained optimization problem can be turned into an unconstrained optimization problem by moving the slack in the constraints into the cost function. The violation of the constraints is penalized heavily with minimum penalty while

satisfying the constraints. This can be captured using an indicator function, which has a discontinuity at the boundary of the constraints. We approximate the indicator function using the negative log barrier function with a parameter 't'.

The equivalent unconstrained optimization problem in each iteration after moving the log barrier function into the cost becomes

$$\min_{W,C,\xi} \quad \sum \xi_i + \lambda \cdot W^T W - \frac{1}{t}\sum_{i=1}^{N} \log(W^T X_i \cdot y_i + C \cdot y_i + \xi_i - 1) - \frac{1}{t}\sum_{i=1}^{N} \log(\xi_i) \qquad (**)$$
$$t \to \infty$$

With each iteration, t is increased by a factor of beta, and the log barrier function starts emulating the indicator function increasingly well. When we reach sufficient accuracy, (t = tmax), we stop the iteration.

This interior point iteration has been shown in the pseudocode `interior_point`. We start the interior point method iteration by making an initial (feasible) guess z0.

```
Pseudocode interior_point
w0 = zeros(M,1); c0 = 0;
for i=1:N
        xi(i) = 0.001 + max((1-y(i)*(w0'*x(:,i)+c)),0);
end
z0 = [w0; c0; xi0];
while t < tmax
        z* = solve_unconstrained(z0);
        z0 = z*;
        t = beta*t; (beta = 15)
end while
```

For each value of t, the unconstrained optimization problem (**) is solved using a Newton iteration, as shown in pseudocode `solve_unconstrained`. The optimal solution z* returned by the Newton step is used to update the initial value z0 for the next Newton step and the parameter t is updated. This is done until t becomes sufficiently large indicating that sufficient accuracy of the log barrier function is achieved.

```
Pseudocode solve_unconstrained
z = z0
sigma = inf;
while sigma/2 <= epsilon (epsilon = 1e-6)
        deltaZ = - hessZ\gradZ;
        sigma = gradZ'*(-deltaZ);
        s = line_search(z,deltaZ);
        z = z + s*deltaZ;
end
```

In each Newton step, the optimization vector z is moved one step closer to the optimum by doing z = z + s* Δz. The step size s is computed by doing line search. In words, the idea is to take as big a step as possible (up to 1*Δz) so long as the new z doesn't become infeasible. The actual implementation is shown in pseudocode `line_search`.

```
Pseudocode line_search
stop = false;
s = 1;
while ~ stop
        z1 = z0 + s*deltaZ;
        i.e. [w1; c1; xi1] = [w0; c0; xi0] + s*deltaZ;
        if (w1'*x(:,i)*y(i)+c1*y(i)+xi(i)-1>0) & (xi(i)>0) forall  i
                        stop = true;
        end
        if ~stop
                s = s/2;
        end
end
```

3. Experimental Results

This section shows the experimental results.

For the first training fold, the channel weights plots for the two data sets for trial 1 are shown in Fig. 1 and 2.

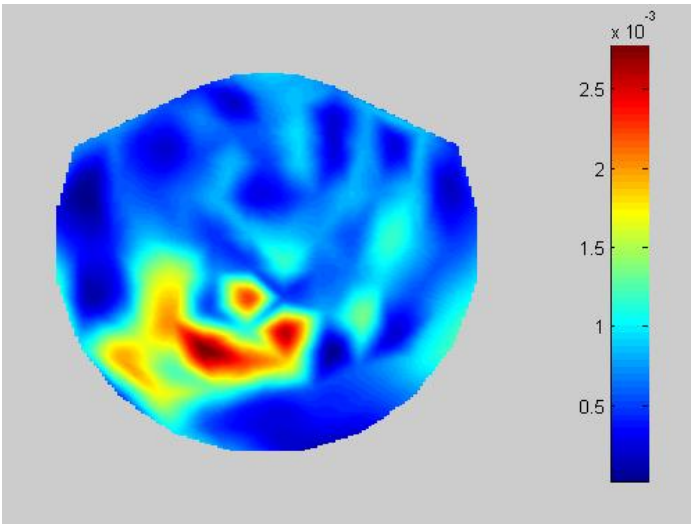


Fig. 1: Channel plot for the first training fold for the Img data set

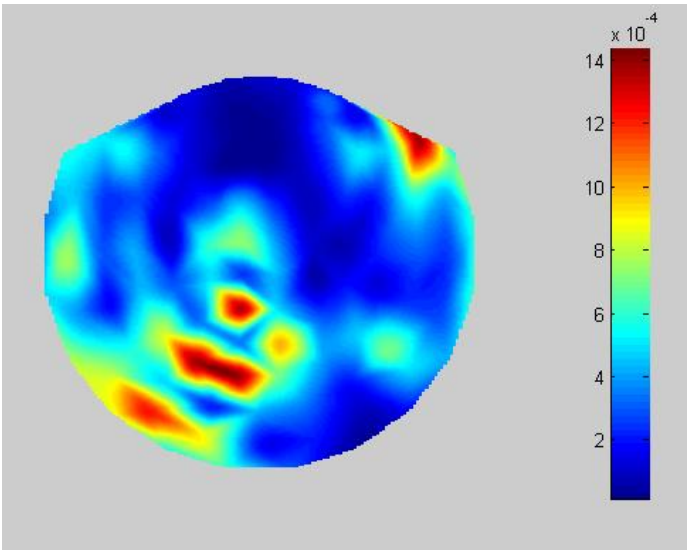


Fig. 2: Channel plot for the first training fold for the Overt data set

Fold #	Overt				Img			
	# Mis		Ac (%)		# Mis		Ac (%)	
	Trial 1	Trial 2	Trial 1	Trial 2	Trial 1	Trial 2	Trial 1	Trial 2
1	0	4	100	90	5	5	87.5	87.5
2	1	1	97.5	97.5	6	4	85	90
3	2	2	95	95	5	4	87.5	90
4	4	2	90	95	3	5	92.5	87.5
5	2	1	95	97.5	4	3	90	92.5
6	4	0	90	100	5	5	87.5	87.5

Table 1: Accuracy of the six runs over 2 trials of 2 datasets

The test accuracy of each fold for the two data sets is shown in the Table 1.

Data set	Mean (%)		Standard Deviation (%)	
	Trial 1	Trial 2	Trial 1	Trial 2
Overt	94.58	95.83	4.0052	3.4157
Img	88.33	89.17	2.582	2.0412

Table 2: Mean and standard deviation of the accuracies of the six folds for the two data sets over two trials

The mean accuracy and standard deviation of the accuracies of all folds for the two data sets is shown in Table 2.

	Overt				Img			
w	-0.0005	0.0004	-0.0004	-0.0004	-0.0119	0.0032	-0.0095	-0.0188
	-0.0001	0.0000	-0.0001	-0.0010	0.0051	0.0066	-0.0113	-0.0371
	-0.0005	-0.0001	-0.0004	0.0006	-0.0058	0.0124	-0.0058	0.0193
	-0.0002	-0.0002	-0.0002	-0.0002	0.0036	0.0072	-0.0021	-0.0048
	-0.0003	-0.0002	-0.0005	0.0002	0.0023	-0.0065	-0.0070	0.0042
	-0.0001	-0.0003	0.0001	0.0002	-0.0010	-0.0113	0.0009	0.0084
	-0.0005	-0.0001	-0.0005	-0.0001	-0.0077	0.0028	-0.0016	-0.0029
	-0.0001	-0.0002	0.0001	0.0001	0.0001	-0.0038	-0.0067	0.0013
	0.0001	-0.0000	-0.0002	-0.0002	-0.0014	0.0029	0.0035	-0.0053
	-0.0003	-0.0002	0.0002	-0.0001	0.0020	-0.0105	-0.0057	-0.0010
	-0.0000	-0.0000	-0.0006	-0.0004	0.0006	0.0008	0.0007	-0.0071
	-0.0000	-0.0001	0.0006	0.0003	0.0096	-0.0090	0.0059	-0.0022
	-0.0004	-0.0001	-0.0004	-0.0001	0.0035	-0.0019	-0.0008	-0.0023
	0.0003	0.0003	0.0003	0.0002	-0.0029	0.0011	0.0021	0.0030
	0.0000	0.0001	-0.0001	0.0002	-0.0062	0.0001	-0.0081	-0.0031
	0.0003	-0.0000	-0.0002	-0.0003	-0.0006	-0.0022	-0.0104	-0.0102
	0.0000	0.0001	-0.0003	0.0001	-0.0015	0.0027	-0.0023	-0.0005
	0.0001	-0.0002	-0.0001	0.0005	0.0022	0.0015	-0.0000	0.0181
	0.0002	0.0001	-0.0007	-0.0002	0.0058	0.0034	-0.0211	-0.0039
	-0.0003	0.0001	-0.0006	-0.0004	-0.0067	0.0025	-0.0111	-0.0148
	-0.0000	0.0004	0.0002	-0.0002	0.0000	0.0041	-0.0074	-0.0015
	-0.0004	0.0001	-0.0005	0.0000	-0.0068	-0.0002	-0.0116	0.0038
	0.0002	0.0005	-0.0006	-0.0001	0.0055	0.0035	-0.0018	0.0060
	-0.0003	0.0003	0.0002	-0.0000	-0.0064	-0.0004	0.0015	0.0017
	0.0000	0.0002	0.0006	-0.0000	-0.0010	0.0028	0.0165	-0.0016
	0.0002	-0.0003	0.0002	-0.0001	0.0073	0.0013	0.0012	0.0004
	-0.0003	-0.0001	0.0009	-0.0001	-0.0078	-0.0062	0.0209	-0.0024
	0.0001	-0.0002	-0.0003	-0.0005	0.0076	-0.0033	0.0002	-0.0089
	-0.0001	-0.0003	-0.0004	0.0003	-0.0023	-0.0062	-0.0028	-0.0048
	0.0000	-0.0000	-0.0002	-0.0001	-0.0084	-0.0018	-0.0038	-0.0037
	0.0000	-0.0003	0.0001	0.0004	0.0018	0.0015	-0.0012	-0.0018
	0.0003	0.0003	0.0001	0.0001	-0.0046	0.0004	0.0136	-0.0018
	-0.0001	0.0003	0.0002	0.0003	-0.0072	0.0000	0.0055	0.0069
	0.0000	-0.0000	0.0001	-0.0005	0.0032	-0.0057	0.0112	-0.0086
	-0.0002	-0.0001	0.0007	0.0005	-0.0099	-0.0056	0.0297	0.0153
	-0.0001	0.0001	-0.0002	0.0001	0.0033	0.0055	-0.0011	-0.0020
	-0.0002	0.0003	-0.0004	0.0001	-0.0019	0.0013	-0.0087	0.0023
	-0.0002	0.0006	0.0001	0.0002	0.0003	0.0046	0.0060	-0.0053
	-0.0000	0.0003	-0.0011	0.0004	0.0028	0.0025	-0.0333	0.0028
	-0.0001	-0.0000	-0.0000	-0.0002	-0.0014	0.0004	-0.0015	0.0031
	-0.0000	0.0004	0.0002	0.0001	0.0030	0.0005	0.0043	0.0012
	-0.0003	0.0005	0.0001	0.0000	-0.0023	-0.0035	0.0062	0.0029
	0.0002	-0.0000	0.0007	-0.0001	0.0003	0.0010	0.0177	-0.0012
	0.0000	-0.0001	-0.0004	-0.0001	0.0038	0.0116	-0.0027	-0.0038
	0.0001	0.0003	0.0005	-0.0003	-0.0019	0.0066	0.0036	0.0096
	-0.0003	-0.0002	-0.0003	-0.0005	0.0036	0.0045	-0.0054	0.0015
	0.0003	0.0001	-0.0003	-0.0001	-0.0004	0.0014	-0.0086	0.0020
	-0.0005	-0.0001	0.0003	-0.0004	-0.0048	-0.0092	0.0166	0.0031
	0.0001	-0.0003	-0.0001	0.0000	0.0084	-0.0041	-0.0064	0.0017
	-0.0002	-0.0009	-0.0007	0.0001	-0.0136	-0.0174	-0.0262	-0.0054
	-0.0002	0.0003	-0.0006	0.0001	0.0069	-0.0121	-0.0284	0.0017
c	-0.6083				-30.8974			

Table 3: The w and c of the first folds of the two data sets (trial 2 is shown here)

Table 3 shows the w and c values for the two data sets for the first fold. The numbers are from the second trial. The w vectors are to be read column-wise, i.e. the first within the first column that shows the w for the Overt data set, the first sub-column shows the w(1:51) values, the second w(52:102) and so on.

Fold #	Overt		Img	
	Trial 1	Trial 2	Trial 1	Trial 2
1	100	100	100	0.01
2	100	100	0.01	0.01
3	100	100	1	0.01
4	1	0.01	1	100
5	100	100	100	0.01
6	1	100	0.01	100

Table 4: The best lambda values that got selected in the two trials on the two data sets

Table 4 shows the different lambda values that got picked as a result of the inner 5-fold cross-validation in each of the 6 iterations for the two data sets.

Apart from this, training errors (# misclassifications) for each inner fold for each lambda value were also saved for each top level iteration for either data sets for two trials. Unfortunately, this is too much information to represent in this report. However, on an average, they seem to agree with the other average values of the overall accuracy results of ~95% for the Overt data set and ~89% accuracy of the Img data set.

4. Discussion

In this we try to interpret our experimental sections from Sec. 3.

4.1 Factors that may impact classification accuracy

One obvious factor that impacts classification accuracy is which data set is used. Looking at Tables 1 and 2, it is clear that the nature of the Overt data set is such that the classification is more accurate on that data set compared to Img.

Other than the data set, another factor that might impact the classification accuracy is the value of lambda chosen. The lambda value decides the relative importance between the classification margin (w) and classification accuracy (xi). Large values of lambda force w to smaller values. Small valued of lambda allow larger values of w. Since w is a factor in determining the classification margin, lambda is in turn responsible for the classification margin and hence classification accuracy.

While more trials would be needed to concretely say this, but it appears that lambda values on the larger side (1 or 100) tend to get chosen for the Overt data set, as compared to Img, which has more instances of 0.01 as the best lambda value. As conjectured in the earlier paragraph, smaller value of lambda tends to allow bigger w values, in turn worse classification margin. This probably explains the slightly lower accuracy (~89%) of the classifier for the Img case, compared to the higher (~95%) accuracy for the Overt case.

4.2 Limits or problems of your approach

Currently, the code is efficient and vectorized, but still serial. The cross validation operations are ideal for running in parallel either by manually spawning multiple computations on different machines in a cluster or automatically by using parallel processing toolbox in Matlab.

In the current implementation for 6x5 = 30 cross-validation training and testing for 4 values of lambda are done serially. They can be run parallelly.

4.3 Possible improvements that can be done

As said in the earlier subsection, the parallel computation, if implemented can in theory make the computation faster by ~ 120 (6x5x4) if 120 computers are available. In reality though, even if it is implemented on 5 machines, one inner cross-validation operation per computer, it will speed up the computation approximately 5 times.

It's probably okay for the code of a class-project to do the cross-validations serially. However, any serious attempt should implement parallel computation because it is so readily doable.

4.4 Anything unique you have done to improve/validate your program's accuracy/efficiency

4.4.1 Validation of accuracy

To validate that the program is running accurately, we used two techniques –

A) Qualitative Validation:
   Qualitative validation can be carried out by making sure that the visual representation of the w vectors looks similar to the eye. Indeed, for either of the data sets, the color/shape/pattern was repeatable across the 2 trials each consisting of 6 folds of cross-validation. and

B) Quantitative Validation:
   Quantitative validation was done by looking at the accuracy and the mean and the standard deviation across the two trials. From Table 2, we can see that the mean and standard deviation numbers are repeatable across the two trials.

4.4.2 Improvement of efficiency

The major step towards improving the efficiency was to vectorize the gradient and Hessian computations. Gradient and Hessian computations are required once in each Newton step, several of which are needed in every interior point method implementation. Then there is the nested cross validation that is done 6x5 = 30 times. Thus, the gradient and Hessian computations are critical to the efficiency of the code. With the dumbest implementation using for loops to do the summations, it would take approximately 5 minutes to compute the Hessian matrix once. The denominators in the gradient and Hessian matrices are same for all the terms and can be pre-computed. Further, the for loops can be replaced by matrix-vector multiplications. Wherever summation is needed, a vector'*vector type computations were used; while in cases where summation wasn't to be computed, diag(vector)*vector was used. Multiplications of terms in a vector and divisions of the denominator were done using the elementwise versions .* and ./ respectively. All these operations are much much faster than their corresponding for loop implementations. Hence the computation time for the Hessian came down from ~ 5 minutes to a fraction of a second.

With the current implementation, it takes approximately half an hour to do the entire series of computations on a single data set.

5. Conclusions

In this project, we build a support vector machine classifier in order to classify the data given by two experiments of movement detection using brain-computer interaction. The accuracy of the classification was tested using 6-fold cross-validation. The parameter lambda, that determines the relative importance of the classification margin (w) and classification (in)accuracy (xi) in the cost function, was decided from four possible values of {0.01, 1, 100, 10000} by doing 5-fold cross-validation in each of the 6 top-level folds.

Experimental results show that this is a promising approach with mean accuracy of classification being about 95% for one data set and about 89% for the other. The computation time was significantly good, however parallel implementation of cross-validation if and when added is likely to provide even more speed up.

References

[1] Project 3 presentation, Jinyin Zhang, Xin Li, ECE Department, Carnegie Mellon University

[2] Lect ure 13, Unconstrained Optimizaion – Gradient and Newton Methods, Xin Li, ECE Department, Carnegie Mellon University

[3] Lect ure 15, Constrained Optimizaion – Inequality Constraints, Nonlinear Constraints and Interior Point Method, Xin Li, ECE Department, Carnegie Mellon University

[4] Lect ure 20, Classification – Support Vector Machine and Regularization, Xin Li, ECE Department, Carnegie Mellon University