University of Southern California, Computer Science

Machine Learning: HomeWork 3

Rohit Kondekar
740-581-9473

October 15, 2014

1 Question 1

1.1 Kernelized Perceptron

Solution a:
When we kernelize a perceptron, rather than having weights for each feature, we have weights for each data point. Consider a data point (x_n, y_n). Assuming initially, the weight vector is initialized to zero. For iteration 1, the value for data point (x_n, y_n) would be simply $y_n\phi_n$. For iteration 2, the value for data point (x_n, y_n) would be $y_n\phi_n + y_n\phi_n$. This can be represented as $\sum_i \alpha_i \phi(x_n)$. So bascically, for a perticular data point n

$$\alpha_n \leftarrow \alpha_n + y_n$$

This can be proved by induction. Base Case: initially zero, so aplha 0. Inductive case: assume true for ith update on data point n. then for i+1th update, new weight vector is given by:

$$w^{i+1} \leftarrow \sum_i \alpha_i \phi(x) + y_n \phi(x)$$
$$w^{i+1} \leftarrow (\sum_i \alpha_i + I(y_n \neq w^i \phi(x)))\phi(x)$$

So α is updated whenever a data point is misclassified.

Solution b:
As we can see that **w** can be written in terms of α and $\phi(x)$ as $\sum_i \alpha_i \phi(x)$ for any given data point x. Though this depends upon $\phi(x)$ the prediction is given by $sign(w^t\phi(x))$

$$\begin{aligned} y^* &= sign(w^t\phi(x)) \\ &= sign(\sum_i \alpha_i \phi(x_i)^T \phi(x)) \end{aligned}$$

So we can see that it depends only on the inner product of nonlinear transformed features.

Solution c:
As shown in part 1, this can be shown using induction: Base Case: initially zero, so alpha 0. Inductive case: assume true for ith update on data point n. then for i+1th update, new weight vector is given by:

$$w^{i+1} \leftarrow \sum_i \alpha_i \phi(x) + y_n \phi(x)$$

$$w^{i+1} \leftarrow (\sum_i \alpha_i + I(y_n \neq w^i \phi(x)))\phi(x)$$

So here we have to maintain α_i for each data point i in the training set. The α_i would be updated at each iteration as: $\alpha_i \leftarrow \alpha_i + y_i$.

Pseudo Code:

```
$\alpha \leftarrow 0$
for iteration = 1..maxItr do
  for all $(x_n, y_n) \epsilon D$ do
    tmp ← $\sum_i \alpha_i \phi(x_i).\phi(x_n)$
    if($y_n.tmp$ ⩽ 0)
      $\alpha_n$ ← $\alpha_n + y_n$
    endif
  endfor
end for
```

2 Question 2

2.1 Support Vector Machine without Bias Term

a) The primal form of this SVM without bias term can be given by:

$$\begin{aligned} min_w \quad & C\sum_n \xi_n + \frac{1}{2}||w||_2^2 \\ st. \quad & 1 - \xi_n - y_n w^T \phi(x_n) \leqslant 0 \quad \forall n \\ & -\xi_n \leqslant 0 \quad \forall n \end{aligned}$$

b) Lagrangian of primal:

$$\begin{aligned} L(w, \{\xi_n, \alpha_n, \lambda_n\}) &= C\sum_n \xi_n + \frac{1}{2}||w||_2^2 - \sum_n \lambda_n \xi_n + \sum_n \alpha_n [1 - \xi_n - y_n w^T \phi(x_n)] \\ st. \quad & \alpha_n \geqslant 0 \\ & \lambda_n \geqslant 0 \end{aligned}$$

c) Taking derivative wrt primal variables:

$$\begin{aligned} \frac{dL}{dw} &= w - \sum_n \alpha_n y_n \phi(x_n) \\ \frac{dL}{d\xi_n} &= C - \lambda_n - \alpha_n \end{aligned}$$

which gives out value for w and contraint:

$$\begin{aligned} w &= \sum_n \alpha_n y_n \phi(x_n) \\ C - \lambda_n - \alpha_n &= 0 \end{aligned}$$

As $\lambda_n \geqslant 0$ we can right the constraint as:

$$C - \alpha_n \geqslant 0$$

d) Substituting value of w into lagrange equation:

$$\begin{aligned}L(w,\{\xi_n,\alpha_n,\lambda_n\}) &= \sum_n (C-\lambda_n-\alpha_n)\xi_n + \frac{1}{2}||\sum_n \alpha_n y_n \phi(x_n)||_2^2 + \sum_n \alpha_n - \sum_{nm} \alpha_n\alpha_m y_n y_m \phi_n^T \phi_m \\ &= \sum_n \alpha_n - \frac{1}{2}\sum_{nm} \alpha_n\alpha_m y_n y_m \phi_n^T \phi_m\end{aligned}$$

e) Dual Equation is given by:

$$max_\alpha \quad \sum_n \alpha_n - \frac{1}{2}\sum_{nm} \alpha_n\alpha_m y_n y_m \phi_n^T \phi_m$$
$$st. 0 \leqslant \alpha_n \leqslant C \quad \forall n$$

f) The main difference between this dual form and the dual form of original SVM is number of constraints. In this dual we donot have the contraint $\sum_n \alpha_n y_n = 0$. Though the equation is same as we get in original SVM, but it is a different optimization problem altogether because of the constraint.

g) The dual is concave function because $\sum_{nm} \alpha_n\alpha_m y_n y_m \phi_n^T \phi_m$ is bascically a L2-norm which is convex and negative of convex is concave, the other part is linear, so the whole equation turns out to be concave, and we are maximizing the equation.

3 Question 3

3.1 Regularization

Here, Log Likelihood of data is given by L(D) and we aim to maximize the likelihood, i.e. find the optimal parameters, given the data. This is also valid if we take log of likelihood, as log is monotonically increasing function, so we could maximize log-likelihood LL(D).

In other terms, we can define cross-entropy as negative of log-likelihood, so rather than maximizing LL(D) we minimize the cross-entropy, so our optimization function would be: **min -LL(D)**.
We would add L2 Norm based regularization, to crossentropy to prevent overfitting. So my optimization objective function would look like:

$$\textbf{min} \text{ -LL(D)} + \frac{\lambda}{2}||w||_2^2 \quad .$$

We need to minimize this objective function.

3.2 Logistic Regression

In Multinomial logistic regression, we may have more than 2 output classes. So probability of class given the data is given by:

$$p(C_k|x) = \frac{e^{w_k^T x}}{\sum_{k'} e^{w_{k'}^T x}}$$

a)

$$p(y=0|x) = \frac{e^{w_0^T x}}{e^{w_0^T x} + e^{w_1^T x}}$$
$$p(y=1|x) = \frac{e^{w_1^T x}}{e^{w_0^T x} + e^{w_1^T x}}$$

b) If we add a constant vector b to both weights, the posterior probability won't change as, take for eg.:

$$\begin{aligned}
p(y=0|x) &= \frac{e^{(w_0^T+b)x}}{e^{(w_0^T+b)x}+e^{(w_1^T+b)x}} \\
&= \frac{e^{w_0^Tx} * e^{b^Tx}}{e^{w_0^Tx} * e^{b^Tx}+e^{w_1^Tx} * e^{b^Tx}} \\
&= \frac{e^{w_0^Tx}}{e^{w_0^Tx}+e^{w_1^Tx}}
\end{aligned}$$

So, we can see that posterior probability remains unchanged.

c) As we know crossentropy for multinomial logistic regression (softmax function) is convex, adding a strictly convex function to it produces a strictly convex function. As we know L2 norm is a strictly convex function, the resulting regularized multinomial logistic regression is strictly convex (i.e. adding a convex function and a strictly convex function produces a strictly convex function).

Strictly convex function has only 1 global minimum, so only single unique solution.

4 Question 4

**** Please note: I am using interior-point-convex Algorithm for QuadProg**

4.1 Programming

4.2 Programming

4.3 Cross validation for linear SVM

**** Please note: I am using interior-point-convex Algorithm for QuadProg**

(a) Cross-Valiation Accuracy and average training time

C	4^{-6}	4^{-5}	4^{-4}	4^{-3}	4^{-2}	4^{-1}	4^{0}	4^{1}	4^{2}
Accuracy	0.5180	0.7860	0.8030	0.8090	0.8010	0.7930	0.7920	0.7950	0.7940
Time	0.4965	0.3394	0.3530	0.4500	0.5729	0.5440	0.6576	0.7106	0.9309

(b) Based on results: C = 0.015625 Max Accuracy = 80.9% Execution Time = 0.44997

(c) Test Accuracy using C = 0.015625 is 84.644%

4.4 Use linear SVM in LIBSVM

C	4^{-6}	4^{-5}	4^{-4}	4^{-3}	4^{-2}	4^{-1}	4^{0}	4^{1}	4^{2}
Accuracy	51.7000	78.8000	80.5000	79.7000	79.6000	79.4000	79.5000	79.3000	79.2000
Time	0.8629	0.6352	0.5311	0.4327	0.4615	0.5563	1.2457	2.9942	9.3822

(a) Cross validation accuracy is almost same (though not exactly same) as that of 4.3. Here the results are:

C = 0.0039062 Max Accuracy = 80.5% Execution Time = 0.53109.

Though the value of C for this is achieved is different.

(b) In my case, as I am using **'interior-point-convex'** Algorithm for QuadProg, my implementation is working faster than LibSVM. For higher values of C, quadprog implementation of Primal Form is many times faster than libsvm implementation.

4.5 Use kernel SVM in LIBSVM

(a) Polynomial Kernel Result

Polynomial Kernel: C = 0.015625 degree = 1 Accuracy = 51.7 Execution time = 0.65389
Polynomial Kernel: C = 0.015625 degree = 2 Accuracy = 51.7 Execution time = 0.6363
Polynomial Kernel: C = 0.015625 degree = 3 Accuracy = 51.7 Execution time = 0.6684
Polynomial Kernel: C = 0.0625 degree = 1 Accuracy = 78.7 Execution time = 0.60371
Polynomial Kernel: C = 0.0625 degree = 2 Accuracy = 51.7 Execution time = 0.65745
Polynomial Kernel: C = 0.0625 degree = 3 Accuracy = 51.7 Execution time = 0.68099
Polynomial Kernel: C = 0.25 degree = 1 Accuracy = 80.4 Execution time = 0.49168
Polynomial Kernel: C = 0.25 degree = 2 Accuracy = 65.4 Execution time = 0.70528
Polynomial Kernel: C = 0.25 degree = 3 Accuracy = 62.1 Execution time = 0.68553
Polynomial Kernel: C = 1 degree = 1 Accuracy = 79.5 Execution time = 0.4204
Polynomial Kernel: C = 1 degree = 2 Accuracy = 72.8 Execution time = 0.63856
Polynomial Kernel: C = 1 degree = 3 Accuracy = 79.2 Execution time = 0.72679
Polynomial Kernel: C = 4 degree = 1 Accuracy = 79.2 Execution time = 0.45648
Polynomial Kernel: C = 4 degree = 2 Accuracy = 70.9 Execution time = 0.75969
Polynomial Kernel: C = 4 degree = 3 Accuracy = 79 Execution time = 0.7609
Polynomial Kernel: C = 16 degree = 1 Accuracy = 79.5 Execution time = 0.58979
Polynomial Kernel: C = 16 degree = 2 Accuracy = 70.9 Execution time = 0.74482
Polynomial Kernel: C = 16 degree = 3 Accuracy = 79.2 Execution time = 0.76233
Polynomial Kernel: C = 64 degree = 1 Accuracy = 79.4 Execution time = 1.2031
Polynomial Kernel: C = 64 degree = 2 Accuracy = 70.9 Execution time = 0.75741
Polynomial Kernel: C = 64 degree = 3 Accuracy = 79.2 Execution time = 0.74191
Polynomial Kernel: C = 256 degree = 1 Accuracy = 79.3 Execution time = 3.1916
Polynomial Kernel: C = 256 degree = 2 Accuracy = 70.9 Execution time = 0.72502
Polynomial Kernel: C = 256 degree = 3 Accuracy = 79.2 Execution time = 0.73048
Polynomial Kernel: C = 1024 degree = 1 Accuracy = 79.2 Execution time = 10.9298
Polynomial Kernel: C = 1024 degree = 2 Accuracy = 70.9 Execution time = 0.71904
Polynomial Kernel: C = 1024 degree = 3 Accuracy = 79.2 Execution time = 0.72722
Polynomial Kernel: C = 4096 degree = 1 Accuracy = 79.3 Execution time = 47.0281
Polynomial Kernel: C = 4096 degree = 2 Accuracy = 70.9 Execution time = 0.78687
Polynomial Kernel: C = 4096 degree = 3 Accuracy = 79.2 Execution time = 0.78495
Polynomial Kernel: C = 16384 degree = 1 Accuracy = 78.8 Execution time = 85.184
Polynomial Kernel: C = 16384 degree = 2 Accuracy = 70.9 Execution time = 0.75199
Polynomial Kernel: C = 16384 degree = 3 Accuracy = 79.2 Execution time = 0.71291

Total Time Taken by Polynomial kernel for crossvalidation: = 166.6174

Best Result : Polynomial Kernel: C = 0.25 Degree = 1 Max Accuracy = 80.4% Execution Time = 0.49168

Test Result: Test Accuracy using C = 0.25 and degree = 1 is 85.5172%

(b) RBF Kernel Result

```
RBF Kernel: C = 0.015625 gamma = 6.1035e-05 Accuracy = 51.7 Execution time = 0.69822
RBF Kernel: C = 0.015625 gamma = 0.00024414 Accuracy = 51.7 Execution time = 0.66623
RBF Kernel: C = 0.015625 gamma = 0.00097656 Accuracy = 51.7 Execution time = 0.65934
RBF Kernel: C = 0.015625 gamma = 0.0039062 Accuracy = 51.7 Execution time = 0.65713
RBF Kernel: C = 0.015625 gamma = 0.015625 Accuracy = 51.7 Execution time = 0.67437
RBF Kernel: C = 0.015625 gamma = 0.0625 Accuracy = 51.7 Execution time = 0.73856
RBF Kernel: C = 0.0625 gamma = 6.1035e-05 Accuracy = 51.7 Execution time = 0.6865
RBF Kernel: C = 0.0625 gamma = 0.00024414 Accuracy = 51.7 Execution time = 0.66038
RBF Kernel: C = 0.0625 gamma = 0.00097656 Accuracy = 51.7 Execution time = 0.66735
RBF Kernel: C = 0.0625 gamma = 0.0039062 Accuracy = 54.5 Execution time = 0.66552
RBF Kernel: C = 0.0625 gamma = 0.015625 Accuracy = 61.2 Execution time = 0.6695
RBF Kernel: C = 0.0625 gamma = 0.0625 Accuracy = 51.7 Execution time = 0.73324
RBF Kernel: C = 0.25 gamma = 6.1035e-05 Accuracy = 51.7 Execution time = 0.66192
RBF Kernel: C = 0.25 gamma = 0.00024414 Accuracy = 51.7 Execution time = 0.66073
RBF Kernel: C = 0.25 gamma = 0.00097656 Accuracy = 73.1 Execution time = 0.66616
RBF Kernel: C = 0.25 gamma = 0.0039062 Accuracy = 80.3 Execution time = 0.58304
RBF Kernel: C = 0.25 gamma = 0.015625 Accuracy = 83.5 Execution time = 0.59991
RBF Kernel: C = 0.25 gamma = 0.0625 Accuracy = 51.7 Execution time = 0.69207
RBF Kernel: C = 1 gamma = 6.1035e-05 Accuracy = 51.7 Execution time = 0.66344
RBF Kernel: C = 1 gamma = 0.00024414 Accuracy = 76.1 Execution time = 0.69227
RBF Kernel: C = 1 gamma = 0.00097656 Accuracy = 79.9 Execution time = 0.56755
RBF Kernel: C = 1 gamma = 0.0039062 Accuracy = 81.7 Execution time = 0.49651
RBF Kernel: C = 1 gamma = 0.015625 Accuracy = 87.2 Execution time = 0.50772
RBF Kernel: C = 1 gamma = 0.0625 Accuracy = 76.6 Execution time = 0.77374
RBF Kernel: C = 4 gamma = 6.1035e-05 Accuracy = 77 Execution time = 0.67293
RBF Kernel: C = 4 gamma = 0.00024414 Accuracy = 79.7 Execution time = 0.56112
RBF Kernel: C = 4 gamma = 0.00097656 Accuracy = 80 Execution time = 0.44823
RBF Kernel: C = 4 gamma = 0.0039062 Accuracy = 84.7 Execution time = 0.42382
RBF Kernel: C = 4 gamma = 0.015625 Accuracy = 88.6 Execution time = 0.67545
RBF Kernel: C = 4 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.7235
RBF Kernel: C = 16 gamma = 6.1035e-05 Accuracy = 79.5 Execution time = 0.58849
RBF Kernel: C = 16 gamma = 0.00024414 Accuracy = 79.7 Execution time = 0.44636
RBF Kernel: C = 16 gamma = 0.00097656 Accuracy = 81.1 Execution time = 0.3905
RBF Kernel: C = 16 gamma = 0.0039062 Accuracy = 88 Execution time = 0.46126
RBF Kernel: C = 16 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.66046
RBF Kernel: C = 16 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.71855
RBF Kernel: C = 64 gamma = 6.1035e-05 Accuracy = 79.8 Execution time = 0.45771
RBF Kernel: C = 64 gamma = 0.00024414 Accuracy = 79.6 Execution time = 0.41542
RBF Kernel: C = 64 gamma = 0.00097656 Accuracy = 85.7 Execution time = 0.44712
RBF Kernel: C = 64 gamma = 0.0039062 Accuracy = 87.2 Execution time = 0.65152
RBF Kernel: C = 64 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.67559
RBF Kernel: C = 64 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.73949
RBF Kernel: C = 256 gamma = 6.1035e-05 Accuracy = 79 Execution time = 0.41609
RBF Kernel: C = 256 gamma = 0.00024414 Accuracy = 81.7 Execution time = 0.45685
RBF Kernel: C = 256 gamma = 0.00097656 Accuracy = 88.1 Execution time = 0.69482
RBF Kernel: C = 256 gamma = 0.0039062 Accuracy = 87.2 Execution time = 0.65124
RBF Kernel: C = 256 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.67262
RBF Kernel: C = 256 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.749
RBF Kernel: C = 1024 gamma = 6.1035e-05 Accuracy = 79.4 Execution time = 0.45277
RBF Kernel: C = 1024 gamma = 0.00024414 Accuracy = 85.4 Execution time = 0.71517
RBF Kernel: C = 1024 gamma = 0.00097656 Accuracy = 87.4 Execution time = 0.9142
RBF Kernel: C = 1024 gamma = 0.0039062 Accuracy = 87.2 Execution time = 0.65705
RBF Kernel: C = 1024 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.68584
RBF Kernel: C = 1024 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.71606
RBF Kernel: C = 4096 gamma = 6.1035e-05 Accuracy = 81.8 Execution time = 0.68642
RBF Kernel: C = 4096 gamma = 0.00024414 Accuracy = 87.8 Execution time = 1.5396
RBF Kernel: C = 4096 gamma = 0.00097656 Accuracy = 87.4 Execution time = 0.89465
RBF Kernel: C = 4096 gamma = 0.0039062 Accuracy = 87.2 Execution time = 0.64724
RBF Kernel: C = 4096 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.67712
RBF Kernel: C = 4096 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.73267
RBF Kernel: C = 16384 gamma = 6.1035e-05 Accuracy = 85.3 Execution time = 1.5239
RBF Kernel: C = 16384 gamma = 0.00024414 Accuracy = 87.4 Execution time = 2.3055
RBF Kernel: C = 16384 gamma = 0.00097656 Accuracy = 87.4 Execution time = 1.0646
RBF Kernel: C = 16384 gamma = 0.0039062 Accuracy = 87.2 Execution time = 0.66495
RBF Kernel: C = 16384 gamma = 0.015625 Accuracy = 88.2 Execution time = 0.71672
RBF Kernel: C = 16384 gamma = 0.0625 Accuracy = 77.9 Execution time = 0.73164
```

Total Time Taken by Polynomial kernel for crossvalidation: = 45.8656

Best Result : BF Kernel: C = 4 gamma = 0.015625 Max Accuracy = 88.6% Execution Time = 0.67545

Test Result: RBF Kernel: Test Accuracy using C = 4 and Gamma = 0.015625 is 90.4368%

As RBF kernel is giving better results in much lower execution time, we would prefer using RBF kernel to work on this problem. RBF Kernel: using C = 4 and Gamma = 0.015625