EL9123 Introduction to Machine Learning, Spring 2018
Midterm Exam (with Solutions)

Prof. Yao Wang

No electronics devices are allowed. No peak into your neighbors. Write neatly and do your best.

1. (15 pt) Suppose that you want to predict the grade that a student will get in an upcoming exam, y, from two factors: x_1 is the number of hours s/he studies, x_2 is the average grade in the homework s/he gets so far. Assume that the normalized grade $\tilde{y}$ is linearly related to normalized features $\tilde{x}_1$ and $\tilde{x}_2$, i.e., $\tilde{y} = w_1\tilde{x}_1 + w_2\tilde{x}_2$, where normalized variables have zero mean and unit variance. You are given N samples consisting of $x_{n,1}, x_{n,2}, y_n, n = 1, 2, \dots, N$.

   (a) (5pt) How do you normalize the raw training data to obtain $\tilde{y}_n, \tilde{x}_{n,1}, \tilde{x}_{n,2}$? Denote the mean and standard deviation of y by μ_y and σ_y, and similarly, the mean and standard deviation of $x_p, p = 1, 2$ by $\mu_{x,p}, \sigma_{x,p}$. Express the mean and STD of these variables in terms of $x_{n,1}, x_{n,2}, y_n, n = 1, 2, \dots, N$.

   First we should compute the mean and STD of each variable. For example, for y, $\mu_y = \frac{1}{N}\sum_{n=1}^{N} y_n, \sigma_y^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - \mu_y)^2$. Note that more precisely, the variance should be computed using a scale factor of $\frac{1}{N-1}$. You will get full mark either way. Then we can normalize each variable. For example, for y, we use $\tilde{y}_n = (y_n - \mu_y)/\sigma_y$. We can apply similar approach to x_1 and x_2.

   (b) (5pt) Suppose you want to find the regression coefficients w_1 and w_2 to minimize the residual sum of squares (RSS) for the normalized training samples. Derive the linear equation that the coefficients must satisfy in the form of $C\mathbf{w} = \mathbf{d}$. Express clearly C and $\mathbf{d}$ in terms of the normalized training data.

   The predicted values for the training samples can be written in matrix form as $\hat{\mathbf{y}} = A\mathbf{w}$, and the RSS can be expressed as

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \tilde{\mathbf{y}}\|^2 = \|A\mathbf{w} - \tilde{\mathbf{y}}\|^2$$

   where

$$\tilde{\mathbf{y}} = \begin{bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \cdots \\ \tilde{y}_N \end{bmatrix}, \quad A = \begin{bmatrix} \tilde{x}_{1,1} & \tilde{x}_{1,2} \\ \tilde{x}_{2,1} & \tilde{x}_{2,2} \\ \cdots & \cdots \\ \tilde{x}_{N,1} & \tilde{x}_{N,2} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

   To minimize $L(\mathbf{w})$, we set the derivative to zero, yielding

$$\frac{\partial J}{\partial \mathbf{w}} = 2A^T(A\mathbf{w} - \tilde{\mathbf{y}}) = 0, \quad \text{or} \quad A^T A\mathbf{w} = A^T\tilde{\mathbf{y}}$$

with

$$C = A^T A = \begin{bmatrix} \sum_n \tilde{x}_{n,1}^2 & \sum_n \tilde{x}_{n,1}\tilde{x}_{n,2} \\ \sum_n \tilde{x}_{n,1}\tilde{x}_{n,2} & \sum_n \tilde{x}_{n,2}^2 \end{bmatrix}$$

and

$$\mathbf{d} = A^T \mathbf{y} = \begin{bmatrix} \sum_n \tilde{x}_{n,1}\tilde{y}_n \\ \sum_n \tilde{x}_{n,2}\tilde{y}_n \end{bmatrix}$$

(c) (5pt) Given a student's features x_1 and x_2 (unnormalized), how do you predict his/her exam grade?

We will use apply the weights determined above to the nomalized features to obtain the predicted value for the normalized y: $\hat{\tilde{y}} = w_1(x_1 - \mu_{x,1})/\sigma_{x_1} + w_2(x_2 - \mu_{x,2})/\sigma_{x_2}$
Then we renormalize to obtain $\hat{y} = \sigma_y \hat{\tilde{y}} + \mu_y$

2. (20 pt) For the linear regression problem described above, write a python program which i) derives the mean and standard deviation of each variable from the training data; ii) performs training data normalization; ii) derives the optimal regression coefficients for the normalized training data; iii) performs prediction for the testing data; iv) determines the average of RSS for all testing samples. Your program can assume that X, a 1000×2 array, stores the feature data, and y, a 1000×1 array, stores the target data. You use first 500 samples as training data, and remaining 500 samples as testing data. Note that you are not allowed to use the function `preprocessing( )`, nor the linear regression model in sklearn. You should write the necessary code to implement each step. However, you could use the function `x=numpy.linalg.lstsq(A,b)` for step ii).

   See Appendix for solution

3. (10 pt) Consider the linear regression problem that predicts the zero mean target y from normalized features $\mathbf{x} = [x_1, x_2, ..., x_P]$ using $\hat{y} = \sum_{p=1}^{P} w_p x_p = \mathbf{w}^T \mathbf{x}$. Suppose that you want to constrain your regression coefficients to sum to 1, while minimizing the RSS for N training samples with feature vectors $\mathbf{x}_n = [x_{n,1}, x_{n,2}, \ldots, x_{n,P}]$ and corresponding target values y_n, $n = 1, 2, \ldots, N$. Formulate the problem as a constrained optimization problem, and derive the solution (or equations that the solution should satisfy). Define all the matrices and vectors clearly in your derivation. Hint: the constraint can be written as $\mathbf{1}^T \mathbf{w} = 1$, where $\mathbf{1}$ is a column vector consisting of 1's in all elements. Note that this may be the only hard problem for you in this exam. If you don't know how to proceed, you should move onto other problems first and get back to this if you have time.

   The problem can be formulated as

$$\begin{array}{rc} \text{Minimize} & J(\mathbf{w}) = \frac{1}{2}\|A\mathbf{w} - \mathbf{y}\|^2 \\ \text{Subject to} & \mathbf{1}^T \mathbf{w} - 1 = 0 \end{array}$$

   Using the Lagrangian multiplier method, we can set up the Lagrangian cost function

$$L(\mathbf{w}, \lambda) = \frac{1}{2}\|A\mathbf{w} - \mathbf{y}\|^2 + \lambda(\mathbf{1}^T \mathbf{w} - 1)$$

   We will examine two possibilities. First we check whether the solution obtained with minimizing the original loss function, which can be expressed as $\mathbf{w} = (A^T A)^{-1} A^T \mathbf{y}$, satisfies

$\mathbf{1}^T\mathbf{w} = 1$. If yes, this is the correct solution. If not, we will minimize L with respect to both $\mathbf{w}$ and λ. This yields

$$\frac{\partial J}{\partial \mathbf{w}} = A^T(A\mathbf{w} - \mathbf{y}) + \lambda \mathbf{1} = \mathbf{0}$$

and

$$\mathbf{1}^T\mathbf{w} - 1 = 0$$

The above two sets of equations can be written as a linear equation of $\mathbf{w}$ and λ, as

$$\begin{bmatrix} A^TA & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \lambda \end{bmatrix} = \begin{bmatrix} A^T\mathbf{y} \\ 1 \end{bmatrix}$$

Solving the above equations will yield the desired solutuon for $\mathbf{w}$.

4. (10 pt) In LASSO regression, we minimize the following loss function

$$L(\mathbf{w}, \lambda) = \sum_n (y_n - \hat{y}_n)^2 + \lambda \sum_p |w_p|,$$

where $\hat{y}_n = \sum_{p=1}^P w_p x_{n,p} = \mathbf{w}^T\mathbf{x}_n$, where $\mathbf{x}_n$ are normalized to have zero mean and unit variance.

(a) (2pt) What is the impact of λ? When λ is large, do you tend to have more or fewer non-zero coefficients?

λ controls the amount of regularization we incorporate. With a larger λ, there will be fewer non-zero coefficients.

(b) (4pt) Suggest a way to optimize the regularization parameter λ.

We can use cross validation to select λ from a large set of candidate values for λ. For each candidate λ_k in a chosen range, we can do a cross validation to obtain the average value for the loss function for different test subsets. Then we pick the λ_k that yields the least loss. If the sample size is large, we could just divide the entire set in half, and use half for training and half for testing. If the sample size is small, we need to do a five or ten fold cross validation.

(c) (2 pt) Under what situation, adding the LASSO regularization becomes important? Hint: Think in terms of the number of features and feature correlations.

When the number of features is large relative to the sample size and when some features are highly correlated with each other, we should use LASSO regularization to pick a subset of features that can yield the most accurate prediction for the test set.

(d) (2 pt) Why is it important to do data normalization with LASSO regression?

The correct coefficient values depend on the variance of the features. If we do not normalize the data, by equally minimizing the absolute value of each coefficient, we are likely penalizing features which have a small dynamic range. By making each feature to have zero mean and unit variance, we can safely use the coefficient magnitude to measure the importance of each feature.

5. (15 pt) With logistic regression for binary classification, we assume the probability that a sample with feature vector $\mathbf{x}$ belong to class y according to the model $P(y = 1|\mathbf{x}) = \frac{1}{1+e^{-z(\mathbf{x})}}$ and $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x})$, with $z(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

(a) (4pt) Suppose we assign a sample to class 1 only if $P(y = 1|\mathbf{x}) >= T$, where T is a preset threshold value, $0 < T < 1$. What is the corresponding linear classifier?

By setting $\frac{1}{1+e^{-z}} = T$, we can obtain the z value reaching this probability: $z(T) = \ln \frac{T}{1-T}$. The corresponding classifier uses a linear separating plane described by $\mathbf{w}^T\mathbf{x} - z(T) = 0$. The classifier output is described by

$$y(\mathbf{x}) = \begin{cases} 1 & \text{If } \mathbf{w}^T\mathbf{x} - z(T) \geq 0 \\ 0 & \text{Otherwise} \end{cases}$$

(b) (3pt) If you are equally concerned about the classification error for either class 1 (y=1) or class 0 (y=0), how would you choose T?

I would choose $T = 1/2$.

(c) (3pt) Suppose the problem is to detect the presence of cancer (y=1 for cancer, and y=0 for non-cancer) based on some medical test results described by $\mathbf{x}$. You are more concerned with missing any "true" cancer. How would you select T?

I would choose T to be smaller than 1/2. If I dont want to miss any possible cancer case, I would choose T close to 0. But this may lead to many false positive, causing unnecessary worry on the patient's side and incurring unnecessary further tests.

(d) (5pt) Suppose you have two features x_1 and x_2, and from the scatter plot of the training data, you know that the two classes are not linearly separable in terms of these features. Furthermore, the scatter plot suggests that class 1 may be defined by a curved region defined by $x_2 \geq a_0 + a_1x_1 + a_2x_1^2$. How would you use logistic regression to solve the problem?

I would use a set of transformed features, $1, x_1, x_1^2, x_2$.

6. (10pt) In Logistic regression, we minimize the negative log likelihood of class labels to determine the weights $\mathbf{w}$. Suppose that instead, you simply minimize the RSS on the estimated probability for class 1, where you use the class label y_n as the ground truth probability. That is,

$$L(\mathbf{w}) = \sum_n \left(y_n - \frac{1}{1 + e^{-z(\mathbf{x}_n)}} \right)^2, \text{ with } z(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

.

(a) (5pt) Derive the gradient of the loss function with respect to $\mathbf{w}$.

The loss function can be expressed as

$$L(\mathbf{w}) = \sum_n f_n(z_n), \text{ with } f_n(z_n) = \left(y_n - \frac{1}{1 + e^{-z_n}} \right)^2, z_n = \mathbf{w}^T\mathbf{x}_n$$

The gradient can be expressed as

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_n \frac{\partial f_n}{\partial z_n} \frac{\partial z_n}{\partial \mathbf{w}}$$

with

$$\frac{\partial f_n}{\partial z_n} = 4 \left(y_n - \frac{1}{1 + e^{-z_n}} \right) \frac{e^{-z_n}}{(1 + e^{-z_n})^2}, \quad \frac{\partial z_n}{\partial \mathbf{w}} = \mathbf{x}_n.$$

In matrix notation, the gradient can be expressed as

$$\frac{\partial L}{\partial \mathbf{w}} = A^T \frac{\partial f}{\partial \mathbf{z}}, \quad \text{with} \quad A^T = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N], \frac{\partial f}{\partial \mathbf{z}} = \left[\frac{\partial f_1}{\partial z_1}, \frac{\partial f_2}{\partial z_2}, \ldots, \frac{\partial f_N}{\partial z_N}\right]^T.$$

(b) (5pt) Describe a gradient descent algorithm to update $\mathbf{w}$.

Given data described in matrix A and $\mathbf{y}$ and a learning rate α, starting with an initial $\mathbf{w}_0$, we can use the following iterations

```
w=  w0

for iter=1 to itermax
        evaluate z at all samples using  z= A w
        evaluate gradient of f at all z, call it gfz,
        evaluate the gradient using grad=A^T  gfz
        update the weight using w = w− alpha∗ grad
```

See appendix for a more complete solution using Python. However, if you wrote something like above, you will get full marks.

7. (10 pt) Consider the data set with 6 samples each with a single feature x_i and binary class label $y_i = \pm 1$.

x_i	2	3	4	5	6	7
y_i	-1	-1	-1	1	1	1

(a) (2 pt) Is this dataset linearly separable?

Yes.

(b) (4pt) Determine the maximum margin classifier. Express the separating plane as $ax+b = 0$, specify a and b. What is the margin of this classifier?

The maximum margin classifier would be equivalent to compare the feature value x with a threshold=4.5, assign 1 if $x >= 4.5$, and assign 0 otherwise. The equivalent separating plane is $x - 4.5 = 0$, with $a = 1$ and $b = -4.5$. The margin for the training data is 0.5.

(c) (4pt) The maximum margin classifier can be considered a Support Vector Classifier. What are the corresponding support vectors?

The supporting vectors are those that are on the margin on each side of the separating plane. In this case, they include $x_i = 4$ and $x_i = 5$.

8. (10 pt) Now, suppose that your training data becomes the following due to noise:

x_i	2	3	4	5	6	7
y_i	-1	-1	1	-1	1	1

(a) (2pt) Is this dataset linearly separable? Can you still find a maximum margin classifier?

This dataset is not linearly separable and we cannot find a maximum margin classifier.

(b) (4pt) A Support Vector Classifier is designed and it makes two errors for the training data. Where is the separating line? What are the corresponding support vectors? Which samples are on the margin? Which samples are on the wrong side of the separating plane? What is the margin?

There are two possible solutions. Solution one is a classifier that has a separating line at 4.5 and margin of 1.5. The support vectors includes samples $x_n = 3, 4, 5, 6$. Samples $x_n = 3$ and $x_n = 6$ are on the two sides of the margin. Samples $x_n = 4$ and $x_n = 5$ are on the wrong side of the separating line but within the margin.

Solution two is a classifier that also has a separating line at 4.5, but has a margin of 2.5. The support vectors includes all samples. Samples $x_n = 2$ and $x_n = 7$ are on the two sides of the margin. Samples $x_n = 3$ and $X_n = 6$ are within the margin. Samples $x_n = 4$ and $x_n = 5$ are on the wrong side of the separating line but within the margin.

If you give either of the above two solutions, your answer is correct. The following answers are incorrect. See Appendix for the solution given by sklearn.svm.SVC under different regularization parameter C.

Note that separating line at 3.5 (or 5.5) with margin 0.5 is NOT a correct answer, because it makes only 1 error. Also it will have a sample ($x_n = 5$) that is on the wrong side beyond the margin.

Separating line at 2.5 (or 6.5) with margin 0.5 is also NOT a correct answer, because there must be a correct sample on each side of the margin. Sample $x = 2$ is on the left side, but sample $x = 3$ is on the wrong side, sample $x = 4$ has a margin of 1.5.

(c) (4pt) Suppose that you found that the separating line for the SVC can be expressed as $\frac{2}{3}x - 3 = 0$. What are the slack variables (i.e. hinge losses) corresponding to all the training samples? Note you should make use of your knowledge of what samples may have non-zero slack variables to simplify your calculation. If you do not know answers to the previous subproblem, then compute directly for all samples, based on which you should be able to identify the support vectors.

From the given line equation, the separating line is located at $x_c = 9/2 = 4.5$, and the margin is $1/(2/3) = 1.5$. Therefore, this corresponds to Solution 1 above. Only those that are inside the margin or on the wrong side of the separating line would have a non-zero loss. The slack variable can be determined by $\epsilon_n = \max(0, 1 - y_n z_n)$, where $z_n = \frac{2}{3}x_n - 3$. Therefore, we only need to calculate the slack for $x_n = 4$ and for $x_n = 5$. For $x_n = 4, z_n = \frac{2}{3}4 - 3 = -1/3, y_n = 1, \epsilon = 4/3$. For $x_n = 5, z_n = \frac{2}{3}5 - 3 = 1/3, y_n = -1, \epsilon_n = 4/3$. It can also be easily confirmed that for all other samples, $\epsilon_n = 0$.

midtermP2_P6_P8_solution

April 1, 2018

0.1 Solution for Prob. 2.

```
In [1]: # For now use some random X and y (not required for exam)
        import numpy as np
        X = np.random.rand(1000,2)
        y = np.random.rand(1000,1)

In [2]: nsamp, natt = X.shape

        # Divide data into training and testing data
        ns_train = nsamp//2
        ns_test = nsamp - ns_train
        X_tr = X[:ns_train,:]     # Gets the first ns_train rows of X_tilde
        y_tr = y[:ns_train]       # Gets the correspoinding rows of y_tilde
        X_ts = X[ns_train:,:]
        y_ts = y[ns_train:]

        # Compute the mean and standard deviation per features
        Xm = np.mean(X_tr,axis=0) # averaging over column
        ym = np.mean(y_tr)
        Sx = np.sqrt(np.mean((X_tr - Xm[None,:])**2,axis=0))
        sy = np.sqrt(np.mean((y_tr-ym)**2))

        # Normalize the training data
        X_tr_tilde = (X_tr - Xm[None,:])/Sx
        y_tr_tilde = (y_tr-ym)/sy

        # Derive optimal regression coefficients for normalized training data
        w_hat = np.linalg.lstsq(X_tr_tilde,y_tr_tilde)

        #For testing normalization, use the same mean and std used for training
        X_ts_tilde = (X_ts - Xm[None,:])/Sx
        y_ts_tilde = (y_ts-ym)/sy

        # Perform prediction for the testing data
        y_ts_tilde_pred = X_ts_tilde.dot(w_hat[0]);

        # Obtain prediction for unnormalized data
```

```
y_ts_pred = y_ts_tilde_pred*sy + ym

# Average RSS for all testing data
RSS_test = np.mean((y_ts_pred-y_ts)**2)
```

0.2 Solution for Prob. 6(b)

```
In [3]: # Q-6
        """
        # Given A,y, alpha, niter, winit
        w = winit

        # Loop over iterations
        for it in range(niter):
            z = A.dot(w)
            py = 1/(1+np.exp(-z))
            gfz = -2*(y-py).dot(np.exp(-z)).dot(py**2) #gradient of f wrt z for all z
            grad = A.T.dot(fgz) #final gradient
            w=w-alpha*grad

Out[3]: '\n# Given A,y, alpha, niter, winit\nw = winit\n\n# Loop over iterations\nfor it in ra
```

0.3 Solution for Prob. 8 solved by sklearn svm.SVC

```
In [14]: # SVM
         import numpy as np
         from sklearn import svm
         Clist = np.array([0.1,1,10,100])
         nC = Clist.shape[0]
         Xtr = np.array([[2, ], [3, ], [4, ], [5, ], [6, ], [7, ]])
         ytr = np.array([-1,-1,1,-1,1,1])
         nsamp = ytr.shape[0]

         print("X=", Xtr[:,0])
         print("y=", ytr)
         print("C values = ", Clist)
         margin = np.zeros((nsamp))
         y_hat =  np.zeros((nsamp))

         for i in range(nC):
             C = Clist[i]
             svc = svm.SVC(C=C, kernel='linear')

             svc.fit(Xtr,ytr)
             z= svc.decision_function(Xtr)
             # slack=1-ytr*z
             y_hat = svc.predict(Xtr)
             SVs= svc.support_vectors_
```

```
            # SVweights=svc.dual_coef_
            weights=svc.coef_
            intercept=svc.intercept_

            print(i,C)
            print(y_hat)
            # print(slack)
            print(SVs)
            # print(SVweights)
            print(weights)
            print(intercept)
```

```
X= [2 3 4 5 6 7]
y= [-1 -1  1 -1  1  1]
C values =  [   0.1    1.    10.   100. ]
0 0.1
[-1 -1 -1  1  1  1]
[[ 2.]
 [ 3.]
 [ 5.]
 [ 4.]
 [ 6.]
 [ 7.]]
[[ 0.4]]
[-1.8]
1 1.0
[-1 -1 -1  1  1  1]
[[ 3.]
 [ 5.]
 [ 4.]
 [ 6.]]
[[ 0.66666667]]
[-3.]
2 10.0
[-1 -1 -1  1  1  1]
[[ 3.]
 [ 5.]
 [ 4.]
 [ 6.]]
[[ 0.66666667]]
[-3.]
3 100.0
[-1 -1 -1  1  1  1]
[[ 3.]
 [ 5.]
 [ 4.]
 [ 6.]]
```

```
[[ 0.66666667]]
[-3.]
```