ECE-GY 6143: Introduction to Machine Learning
Midterm Solutions, Fall 2020

Prof. Sundeep Rangan

1. (8 points) *Least squares with a transformation.* Consider the following model for a scalar target $\hat{y}$ from features $\mathbf{x} = (x_1, \ldots, x_d)$,

$$\hat{y} = \sum_{j=1}^{d} \alpha_j \exp(-\beta_j x_j).$$

   (a) (2 points) Is the model linear in the parameters α if β is **fixed**?

   (b) (2 points) Is the model linear in parameters (α, β) **together**?

   (c) (4 points) Write a python function `transform`,

```
def transform(X, beta):
    ...
    return Z
```

   The function should take data matrix `X` and `beta` and return a matrix `Z` such that `yhat = Z.dot(theta)`. For full credit avoid for loops.

No explanations are need for parts (a) or (b). No partial credit will be awarded for these parts. There will be part credit for part (c).

Solution:

(a) Yes. It is linear in the parameters α.

(b) No. There is an exponential in β.

(c) One solution is:

```
def transform(X, beta):
    Z = np.exp(-X*beta[None,:])
    return Z
```

2. (8 points) *Least squares optimization.* Given a model,

$$\widehat{y}_i = \theta x_{i1} + (1-\theta)x_{i2},$$

find the θ to minimize the least squares loss,

$$J(\theta) = \sum_{i=1}^{N} (y_i - \widehat{y}_i)^2.$$

Solution:

Take the derivative,

$$\frac{\partial J}{\partial \theta} = 2\sum_{i=1}^{N} (\widehat{y}_i - y_i)\frac{\partial \widehat{y}_i}{\partial \theta} = 2\sum_{i=1}^{N} (\widehat{y}_i - y_i)(x_{i1} - x_{i2}).$$

Set the derivative to zero and substitute $\widehat{y}_i = \theta x_{i1} + (1-\theta)x_{i2}$,

$$\begin{aligned}
\frac{\partial J}{\partial \theta} = 0 &\iff \sum_{i=1}^{N} (\theta x_{i1} + (1-\theta)x_{i2} - y_i)(x_{i1} - x_{i2}) = 0 \\
&\iff \theta \sum_i (x_{i1} - x_{i2})^2 = \sum_i y_i (x_{i1} - x_{i2}) \\
&\iff \theta = \left(\sum_i (x_{i1} - x_{i2})^2 \right)^{-1} \sum_i y_i (x_{i1} - x_{i2}).
\end{aligned}$$

3. (9 points) *K-fold model validation.* A researcher performs K-fold validation for a regression problem and gets the results below.

Model order d	1	2	3	4	5	6	7	8	9	10
R^2 train mean	0.21	0.49	0.69	0.81	0.85	0.86	0.87	0.88	0.89	0.9
R^2 train SE	0.1	0.1	0.1	0.1	0.1	0.09	0.08	0.07	0.06	0.05
R^2 test mean	0.16	0.44	0.64	0.76	0.8	0.78	0.76	0.74	0.72	0.7
R^2 test SE	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

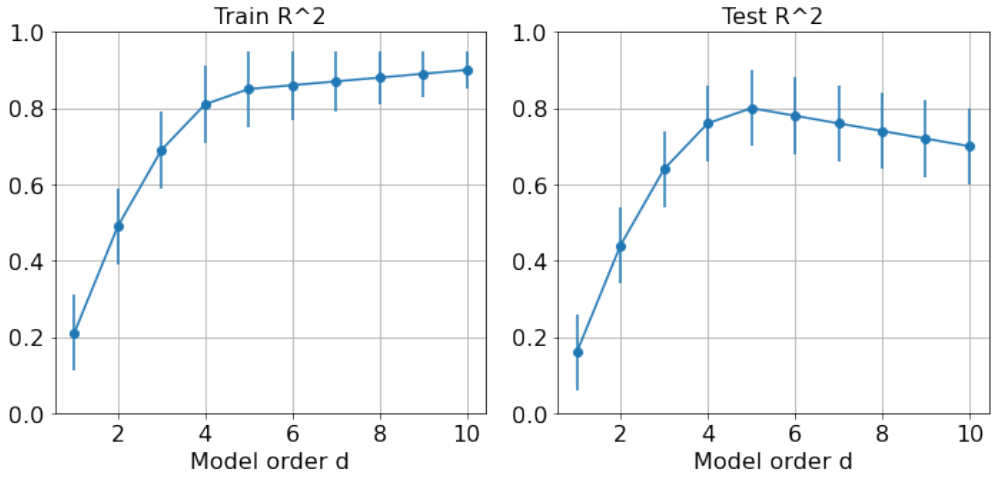


Figure 1: Training and test R^2 values as a function of the model order d

Answer the following. No explanations needed. No partial credit.

(a) (3 points) What is the optimal model order with the normal rule?

(b) (3 points) What is the optimal model order with the one SE rule?

(c) (3 points) At the model order of $d = 8$, the training R^2 is higher than the test R^2 since at $d = 8$ (SELECT ONE):

A. The training model is better than the test model.

B. There is significant over-fitting.

C. There is high bias error.

D. There is significant under-modeling.

Solution:

(a) Optimal order is $d = 5$ since it has the highest mean test R^2.

(b) Optimal order is $d = 4$: At $d = 5$, the mean test $R^2 = 0.8$ and the SE is 0.1. So, the R^2 target is 0.7. This simplest model meeting this target is $d = 4$.

(c) B. The training model begins to out-perform the test model since it over-fits the noise. If there is under-fitting, it will appear in both the training and test.

4. (15 points) *Model selection.* A researcher wants to perform some binary classification task using data from two cities:

   - `X1,y1`: Data from city 1.
   - `X2,y2`: Data from city 2.

   The outputs `y1` and `y2` are binary labels. For the questions below, assume you have the following methods:

```
# Splits data into training and test
Xtr,Xts,ytr,yts = train_test_split(X,y,test_size)

regr = LogisticRegression()   # Constructs a logistic regression object
regr.fit(Z,y)                 # Fits a model with features Z and outputs y
yhat = regr.predict(Z)        # Predicts outputs given features Z

C = np.vstack((A,B)) # stack two matrices on top of each other
c = np.hstack((a,b)) # join two vectors into one long vector
```

   Write a few lines of python for each of the following tasks:

   (a) (5 points) Fit two **separate models** for each city, and measure the test accuracy in each city separately. Use a train-test split with `test_size=0.3` for both cities.

   (b) (6 points) Fit one **combined model** combining the data in both cities and measure the test accuracy on the combined data. Use a train-test split with `test_size=0.3` for both cities.

   (c) (4 points) Select whether using a single model or separate models is better.

   There is no single correct answer. Make any reasonable assumptions.

   Solution:

   (a) For this case, we loop over the two models:

```
acc_sep = []
for X,y in [[X1,y1], [X2,y2]]:
    Xtr,Xts,ytr,yts = train_test_split(X,y,test_size=0.3)

    # Fit the model
    regr = LogisticRegression()
    regr.fit(Xtr,ytr)

    # Measure the test accuracy
    yhat = regr.predict(Xts)
    acc_sep.append( np.mean(yhat == yts))
```

   (b) For this case, we combine the data sets first:

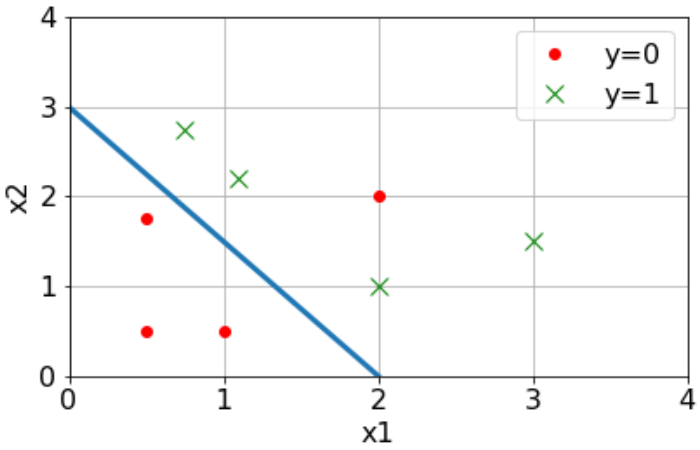
```
# Combine the data
```

```
X = np.vstack((X1,X2))
y = np.hstack((y1,y2))

# Split the merged data
Xtr,Xts,ytr,yts = train_test_split(X,y,test_size=0.3)

# Fit the model
regr = LogisticRegression()
regr.fit(Xtr,ytr)

# Measure the test accuracy
yhat = regr.predict(Xts)
acc_com =  np.mean(yhat == yts)
```

(c) We can compare the accuracy of the combined model with average of the two separate models:

```
acc_sep_avg = (acc_sep[0] + acc_sep[1])/2
if (acc_com >= acc_sep_avg):
    print('Select combined model')
else:
    print('Select separate models')
```

5. (15 points) *Linear Classification.* A data set has eight data points, $(\mathbf{x}_i, y_i)$ where each data point has two features $\mathbf{x}_i = (x_{i1}, x_{i2})$ a binary label $y_i = \{0, 1\}$. The points are shown below.



(a) (7 points) Write the equations for a binary linear classifier that produces $\widehat{y}$ from $\mathbf{x}$. Find parameters for the classifier such that boundary line of the classifier matches the solid line on the graph. The rule should only make one error.

(b) (8 points) Write the equations for $P(y = 1|\mathbf{x})$ for a logistic classifier. Find parameters for the classifier such that:

- The set of points $\mathbf{x}$ with $P(y = 1|\mathbf{x}) = 0.5$ matches the solid line in the figure
- $P(y = 1|\mathbf{x}) = 0.8$ at $\mathbf{x} = (2, 2)$.

Your answer will have logarithms in them. You do not need to evaluate the logarithms or simplify any expressions. Just provide enough steps that the parameters can be computed.

Solution:

(a) A linear classifier will be of the form,

$$\widehat{y} = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0, \end{cases} \quad z = w_0 + w_1 x_1 + w_2 x_2. \tag{1}$$

To find the parameters $\mathbf{w}$, we want the boundary line $z = 0$ when $x_2 = 3 - 1.5x_1$. Since we want to select $\widehat{y} = 1$ when you are above this line, which corresponds to $x_2 > 3 - 1.5x_1$. This matches the classifier (1) with $z = 1.5x_1 + x_2 - 3$ or equivalently,

$$\mathbf{w} = [-3, 1.5, 1].$$

(b) The logistic model is,

$$P(y=1|\mathbf{x}) = \frac{1}{1+e^{-z}}, \quad z = w_0 + w_1 x_1 + w_2 x_2.$$

Since we want to have the same boundary as in part (a), it must be a scaled version of the parameters,

$$\mathbf{w} = \alpha[-3, 1.5, 1],$$

for some α. For a probability of 0.8, we must have,

$$P(y=1|\mathbf{x}) = 0.8 = \frac{1}{1+e^{-z}} \Leftrightarrow z = -\ln\left(\frac{1}{0.8} - 1\right).$$

For this to occur at $\mathbf{x} = (2, 2)$, we need,

$$\begin{aligned} z = w_0 + w_1 x_1 + w_2 x_2 &= \alpha(-3 + 1.5(2) + 1(2)) \\ \Longleftrightarrow \alpha &= -\frac{1}{-3 + 1.5(2) + 1(2)} \ln\left(\frac{1}{0.8} - 1\right). \end{aligned}$$

If you write this, you will get full credit since there is no need to simplify. But, if you are interested, the expression evaluates to, $\alpha \approx 0.69$ and

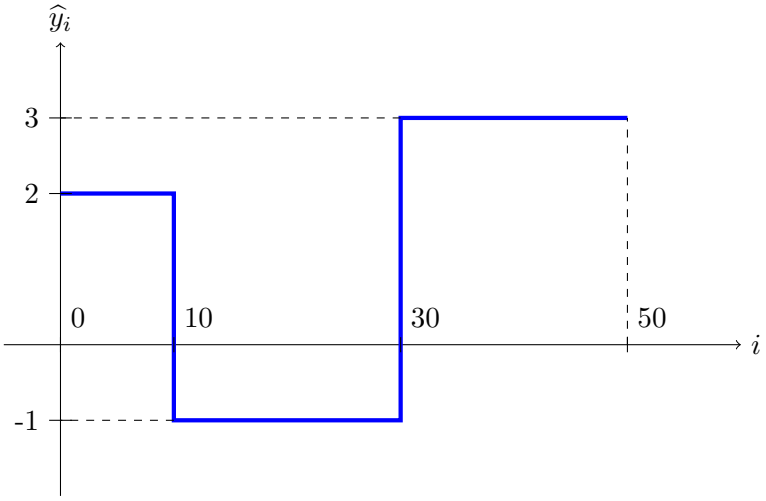
$$\mathbf{w} = \alpha[-3, 1.5, 1] \approx [-2.1, 1.0, 0.69].$$

6. (15 points) *LASSO.* Consider a model $\widehat{\mathbf{y}} = \mathbf{A}\mathbf{w}$ where $\mathbf{A}$ is the $T \times T$ matrix,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}.$$

and $\mathbf{w}$ are some unknown coefficients.

(a) (9 points) Find $\mathbf{w}$ for $\widehat{\mathbf{y}} = (\widehat{y}_0, \widehat{y}_1, \ldots, \widehat{y}_{T-1})$ with $T = 50$ shown in the figure below. At the two discontinuities assume $\widehat{y}_{10} = 2$ and $\widehat{y}_{30} = -1$.



(b) (6 points) For a given $\mathbf{y}$, suggest a regularized cost function to find $\mathbf{w}$ such that $\widehat{\mathbf{y}} = \mathbf{A}\mathbf{w}$ is close to $\mathbf{y}$ but $\widehat{y}_i$ is mostly constant and changes values for only a few i.

Solution:

(a) We see that

$$\widehat{y}_i = \sum_{j=0}^{i} w_j,$$

so $w_0 = \widehat{y}_0$ and $w_i = \widehat{y}_i - \widehat{y}_{i-1}$ for all $i > 0$. Therefore, we can take,

$$\begin{aligned} w_0 &= \widehat{y}_0 = 2 \\ w_{11} &= \widehat{y}_{11} - \widehat{y}_{10} = (-1) - (2) = -3 \\ w_{31} &= \widehat{y}_{31} - \widehat{y}_{30} = (3) - (-1) = 4 \end{aligned}$$

The other $w_j = 0$. Thus, we see that w is *sparse*. If you got confused with the w_{11} and w_{10} or w_{31} and w_{30}, you will still get full credits.

(b) We see that when we make $\mathbf{w}$ sparse, then $\widehat{y}_i$ changes only a few values i. Thus, to

impose sparsity, we can use a L1 regularization,

$$\|\mathbf{y} - \mathbf{A}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_1.$$

7. (15 points) *Computing gradients.* Consider the model,

$$\widehat{y}_i = \log\left[1 + \exp\left(\sum_{j=1}^{d} X_{ij}\beta_j\right)\right].$$

The model is trained to minimize the RSS

$$J = \sum_{i=1}^{n}(y_i - \widehat{y}_i)^2.$$

(a) (7 points) Compute the gradient components, $\partial J/\partial \beta_j$.

(b) (8 points) Write a python function to that returns function J and the gradients,

```
def Jeval(...):
    ...
    return J, Jgrad
```

You must determine the arguments for the function. For full credit, avoid for loops.

Solution:

(a) Let $z_i = \sum_{j=1}^{d} X_{ij}\beta_j$ so that we can write, $\widehat{y}_i = \log(1 + e^{z_i})$. Hence,

$$\frac{\partial \widehat{y}_i}{\partial z_i} = \frac{e^{z_i}}{1 + e^{z_i}} = \frac{1}{1 + e^{-z_i}}.$$

Then, by chain rule,

$$\frac{\partial J}{\partial \beta_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial \widehat{y}_i}\frac{\partial \widehat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \beta_j} = \sum_{i=1}^{N} 2(\widehat{y}_i - y_i)\frac{1}{1 + e^{-z_i}}X_{ij}.$$

(b) One possible solution is as follows:

```
def Jeval(beta,X,y):
    # Compute z and yhat
    z = X.dot(beta)
    yhat = np.log( 1 + np.exp(z))

    # Compute loss
    J = np.sum((y-yhat)**2)

    # Compute dJ_dz[i]
    dJ_dz = 2*(yhat-y)/(1+np.exp(-z))

    # Compute the gradient
    Jgrad = X.T.dot(dJ_dz)
```

```
        return J, Jgrad
```

8. (15 points) *Optimization line search.*

   (a) (7 points) Suppose for $\mathbf{w}_0 = (1, 2, 3)$ we have $f(\mathbf{w}_0) = 4$ and $\nabla f(\mathbf{w}_0) = (5, 6, 7)$. Using a first order approximation, what is $f(\mathbf{w}_1)$ where $\mathbf{w}_1$ is the gradient step,

$$\mathbf{w}_1 = \mathbf{w}_0 - \alpha \nabla f(\mathbf{w}_0), \quad \alpha = (10)^{-4}.$$

   Your answers will have some additions and subtractions. You do not need to evaluate them or simplify your expressions.

   (b) (8 points) Now suppose you are given a python function `feval` that returns $f(\mathbf{w})$ and its gradient $\nabla f(\mathbf{w})$:

```
f, fgrad = feval(w)
```

   For a given $\mathbf{w}_0$, we want to find the α that minimizes the following:

$$\widehat{\alpha} = \arg\min_{\alpha} f(\mathbf{w}_0 - \alpha \nabla f(\mathbf{w}_0)),$$

   Complete the following function to perform the optimization:

```
def linesearch(feval, w0, alpha_min, alpha_max, nalpha):
    ...
    return alpha_opt
```

   The function should try `nalpha` points linearly spaced from `alpha_min` to `alpha_max`.

Solution:

(a) The first order approximation is,

$$\begin{aligned} f(\mathbf{w}_1) &= f(\mathbf{w}_0 - \alpha \nabla f(\mathbf{w}_0)) \\ &\approx f(\mathbf{w}_0) - \alpha \nabla f(\mathbf{w}_0)^{\mathsf{T}} \nabla f(\mathbf{w}_0) \\ &= f(\mathbf{w}_0) - \alpha \|\nabla f(\mathbf{w}_0)\|^2 \\ &= 4 - (10)^{-4}(5^2 + 6^2 + 7^2). \end{aligned}$$

   You do not need to simplify the function further.

(b) One possible solution is as follows:

```
def linesearch(feval, w0, alpha_min, alpha_max, nalpha):
    alpha = np.linspace(alpha_min, alpha_max, nalpha)

    f0, fgrad0 = feval(w0)
    f1 = np.zeros(nalpha)
    for i in range(nalpha):
        w1 = w0 - alpha[i]*fgrad0
        f1[i], fgrad1 = feval(w1)

    iopt = np.argmin(f1)
    alpha_opt = alpha[iopt]
```

```
        return alpha_opt
```