# ECE-GY 6143: Introduction to Machine Learning
# Midterm , Spring 2019

Prof. Sundeep Rangan

1. (13 points) *Least squares with a transformation.* Consider the following model for a scalar target $\hat{y}$ from features $\mathbf{x} = (x_1, x_2)$,

$$\hat{y} = f(x, \theta) = \begin{cases} \theta_0 + \theta_1(x_1 + \theta_2 x_2) & \text{if } x_1 < x_2 \\ \theta_0 + \theta_1(1 + \theta_2)x_1 & \text{if } x_1 \geq x_2 \end{cases}$$

where $\theta = (\theta_0, \theta_1, \theta_2)$ are parameters.

(a) (9 points) Write this as a linear model. That is, find basis functions $\phi_i(\mathbf{x})$ and parameters $\beta_i$ such that

$$\hat{y} = \sum_{i=1}^{p} \beta_i \phi_i(\mathbf{x}).$$

Use a minimum number $p$ of basis functions. Write the linear model parameters $\beta_i$ in terms of $\theta_j$.

(b) (4 points) Given a linear parameter estimate $\boldsymbol{\beta}$, how do you find $\boldsymbol{\theta}$ in the original model?

**Solution**

(a) Define,

$$\phi_1(\mathbf{x}) = 1, \quad \phi_2(\mathbf{x}) = x_1, \quad \phi_3(\mathbf{x}) = \begin{cases} x_2 & \text{if } x_1 < x_2 \\ x_1 & \text{if } x_1 \geq x_2. \end{cases}$$

Then,

$$f(x, \theta) = \theta_0 \phi_1(\mathbf{x}) + \theta_1 \phi_2(\mathbf{x}) + \theta_1 \theta_2 \phi_3(\mathbf{x}).$$

Define,

$$\beta_1 = \theta_0, \quad \beta_2 = \theta_1, \quad \beta_3 = \theta_1 \theta_2.$$

(b) We invert the equations,

$$\theta_0 = \beta_1, \quad \theta_1 = \beta_2, \quad \theta_2 = \beta_3/\theta_1 = \beta_3/\beta_2.$$

2. (15 points) *Computing the bias.* Suppose that a true function is

$$y = f_0(x) := \min\{2x, 6\}.$$

We get data $(x_i, y_i)$, $i = 1, \ldots, n$ with $y_i = f_0(x_i)$, and fit a model,

$$\widehat{y} = f(\mathbf{x}, \hat{\theta}) = \widehat{\theta} x, \quad \widehat{\theta} = \frac{1}{n} \sum_{i=1}^{n} \frac{y_i}{x_i}.$$

For each set of the following sets of training data points $\mathbf{x} = (x_1, \ldots, x_n)$ and test data point $x_{\text{ts}}$ find the bias:

$$\text{Bias}(x_{\text{ts}}) = f(x_{\text{ts}}, \hat{\theta}) - f_0(x_{\text{ts}}).$$

(a) (5 points) Training data is $\mathbf{x} = \{1, 2\}$ and test data point is $x_{\text{ts}} = 2$.

(b) (5 points) Training data is $\mathbf{x} = \{1, 2\}$ and test data point is $x_{\text{ts}} = 4$.

(c) (5 points) Training data is $\mathbf{x} = \{1, 4\}$ and test data point is $x_{\text{ts}} = 4$.

**Solution**

For the three possible training points $x_i = 1, 2, 4$, we first compute $y_i$ and $y_i/x_i$ as shown in the table.

| $x_i$ | $y_i = \min\{2x, 6\}$ | $y_i/x_i$ |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 4 | 2 |
| 4 | 6 | 1.5 |

(a) The parameter estimate $\widehat{\theta}$ is the average of $y_i/x_i$ for the points in the training data. So, for the training data, $\mathbf{x} = \{1, 2\}$,

$$\widehat{\theta} = \frac{1}{2}(2 + 2) = 2,$$

and the bias at $x_{\text{ts}} = 2$ is

$$\begin{aligned} \text{Bias}(x_{\text{ts}}) &= f(x_{\text{ts}}, \hat{\theta}) - f_0(x_{\text{ts}}) \\ &= \widehat{\theta} x_{\text{ts}} - \max\{2x_{\text{ts}}, 6\} = 2(2) - 2(2) = 0. \end{aligned}$$

(b) The training data is the same, so again we get $\widehat{\theta} = 2$. The bias at $x_{\text{ts}} = 4$ is

$$\begin{aligned} \text{Bias}(x_{\text{ts}}) &= f(x_{\text{ts}}, \hat{\theta}) - f_0(x_{\text{ts}}) \\ &= \widehat{\theta} x_{\text{ts}} - \max\{2x_{\text{ts}}, 6\} = 4(2) - 6 = 2. \end{aligned}$$

(c) For the training data, $\mathbf{x} = \{1, 4\}$,

$$\widehat{\theta} = \frac{1}{2}(2 + 1.5) = 1.75.$$

The bias at $x_{\text{ts}} = 2$ is

$$\begin{aligned} \text{Bias}(x_{\text{ts}}) &= f(x_{\text{ts}}, \hat{\theta}) - f_0(x_{\text{ts}}) \\ &= \widehat{\theta} x_{\text{ts}} - \max\{2x_{\text{ts}}, 6\} = 4(1.75) - 6 = 1. \end{aligned}$$

3. (15 points) *Model selection.* You are given python arrays `X` and `y` where `X` is an `n x p` data matrix and `y` is a `n` dimensional vector of continuous targets. For each model order `d=0,1,...,p`, you consider a linear regression model using only the first `d` features:

```
yhat[i] = b + X[i,0]*w[0] + ... + X[i,d-1]*w[d-1],
```

where `b` and `w[0],...,w[d-1]` are the parameters. Write python code to:

- Split the data into training and test with approximately 50% of the samples in each.
- Use simple cross-validation to find the optimal order `d`.

You do not need to use $K$-fold validation or shuffle the data. You can use the normal rule, not the one SE rule. You may use the functions:

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

**Solution.** One solution is as follows:

```
# Split the data into training and test
n = X.shape[0]
ntr = n // 2
Xtr = X[:ntr]
Xts = X[ntr:]
ytr = y[:ntr]
yts = y[ntr:]

# Loop over model orders
p = X.shape[1]
rss = np.zeros(p)
for d in range(p):
    # Fit linear model with d features
    regr = LinearRegression()
    regr.fit(Xtr[:,:d], ytr)

    # Predict on test
    yhat = regr.predict(Xts[:,:d])
    rss[d] = np.mean((yhat-yts)**2)

# Find optimal model order
dopt = np.argmin(rss)
```

4. (14 points) *Linear Classification.* You are given the following five training data points, $(x_i, y_i)$, with binary class labels $y_i \in \{0, 1\}$:

| $x_i$ | 0 | 1.5 | 1.6 | 3 | 4 |
|---|---|---|---|---|---|
| $y_i$ | 0 | 1 | 1 | 0 | 0 |

Consider a classifier of the form,

$$\widehat{y}_i = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i < 0, \end{cases} \tag{1}$$

where $z_i$ is some function of $x_i$.

(a) (6 points) Find $\beta_0, \beta_1$ such that with $z_i = \beta_0 + \beta_1 x_i$, the classifier (1) makes a minimum number of errors. Indicate which training data points, if any, are misclassified.

(b) (8 points) Find $\beta_0, \beta_1, \beta_2$ such that when $z_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$, the classifier (1) makes a minimum number of errors. Indicate which training data points, if any, are misclassified.

**Solution**

(a) In this case, we are limited to a linear classifier. One possible linear classifier takes,

$$\widehat{y} = 1 \text{ when } x < 2. \tag{2}$$

This makes only one error on the training data – the point $(x_i, y_i) = (0, 0)$. To implement the classifier (3), take

$$z_i = 2 - x_i,$$

which corresponds to $(\beta_0, \beta_1) = (2, -1)$.

(b) To make no errors, we would like

$$\widehat{y} = 1 \text{ when } x \in [1, 2]. \tag{3}$$

So, we would like $z > 0$ in the region $x \in (1, 2)$. This can be done with the quadratic,

$$z = -(x-1)(x-2) = -x^2 + 3x - 2.$$

So, we take $(\beta_0, \beta_1, \beta_2) = (-2, 3, -1)$.

5. (14 points) *Regularized least squares.* We are given data $(x_i, y_i)$, $i = 0, \ldots, T-1$ and wish to fit a model of the form,

$$y_i \approx \widehat{y}_i = \sum_{j=0}^{d-1} w_j x_{i-j},$$

for some parameters $w_j$. You can assume $x_j = 0$ for $j < 0$. You decide to use regularized least squares,

$$\widehat{\mathbf{w}} := \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{w}\|^2 + \lambda\phi(\mathbf{w}).$$

(a) (5 points) What are the components $A_{ij}$ of the matrix $\mathbf{A}$?

(b) (5 points) Suggest a function $\phi(\mathbf{w})$ if it is known that we should have $w_j = 0$ for most $j$.

(c) (4 points) Suggest a function $\phi(\mathbf{w})$ if it is known that we should have $w_j \geq 0$ and $w_{j+1} \leq w_j$ for most indices $j$.

For parts (b) and (c), there is no single correct answer.

**Solution**

(a) Components are $A_{ij} = x_{i-j}$ with

$$\mathbf{A} = \begin{bmatrix} x_0 & 0 & \cdots & 0 \\ x_1 & x_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_{T-1} & x_{T-2} & \cdots & x_{T-d} \end{bmatrix}.$$

To get full credit, you needed to state the dimensions of $\mathbf{A} \in \mathbb{R}^{T\times d}$.

(b) Take $\phi(\mathbf{w}) = \|\mathbf{w}\|_1$.

(c) There are a few ways you can do this. For example, you can take a function such as

$$\phi(\mathbf{w}) := \sum_{j=0}^{d-1} \max\{0, -w_j\} + \sum_{j=0}^{d-2} \max\{0, w_{j+1} - w_j\}.$$

The first term penalizes negative values of $w_j$. The second term penalizes positive values $w_{j+1} - w_j$.

6. (16 points )*Computing gradients.* Consider the function,

$$J = \sum_{i=1}^{N} \ln(1 + e^{z_i}) - z_i y_i, \quad z_i = \sum_{j=1}^{p} \frac{a_j}{1 + (x_i - b_j)^2}.$$

(a) (8 points) Compute the gradient components, $\partial J/\partial a_j$ and $\partial J/\partial b_j$.

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

```
def Jeval(...):
    ...
    return J, Jgrad_a, Jgrad_b
```

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

**Solution**

(a) Let $L_i = \ln(1 + e^{z_i}) - z_i y_i$ so that $L = \sum_i L_i$ and

$$\frac{\partial L_i}{\partial z_i} = \frac{e^{z_i}}{1 + e^{z_i}} - y_i = \frac{1}{1 + e^{-z_i}} - y_i.$$

Then,

$$\frac{\partial L}{\partial a_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial a_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial z_i}\frac{\partial z_i}{\partial a_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial z_i}\frac{1}{1 + (x_i - b_j)^2}.$$

Similarly,

$$\frac{\partial L}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial z_i}\frac{\partial z_i}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial z_i}\frac{2a_j(x_i - b_j)}{(1 + (x_i - b_j)^2)^2}.$$

(b) One possible solution is as follows:

```
def Jeval(a,b,x,y):
    # Compute difference D[i,j] = x[i] - b[j]
    D = x[:,None] - b[None,:]

    # Compute P[i,j] = 1/(1+D[i,j]**2)
    #         z[i] = sum_j P[i,j]*a[j]
    P = 1/(1+D**2)
    z = np.sum(a[None,:]*P,axis=1)

    # Compute loss
    J = np.sum(np.log(1+np.exp(z)) - z*y)

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

    # Compute gradients
    Jgrad_a = np.sum(dJ_dz[None,:]*P,axis=0)
    Jgrad_b = np.sum(2*dJ_dz[None,:]*a[None,:]*D*(P**2),axis=0)

    return J, Jgrad_a, Jgrad_b
```

7. (13 points) *Gradient descent with a stopping condition.* You are given a python function that returns a loss function $f(\mathbf{a}, \mathbf{b})$ and the gradients with respect to vector parameters $\mathbf{a}$ and $\mathbf{b}$:

```
f, fgrad_a, fgrad_b = feval(a,b)
```

Write a few lines of python code that runs gradient descent for some initial conditions `ainit` and `binit` with a fixed step size `step`. The code is suppose to stop when either of the following conditions occurs:

- `max_iter=500` iterations have been done, or
- $\sum_j (\partial f/\partial a_j)^2 \leq (10)^{-8}$ and $\sum_k (\partial f/\partial b_k)^2 \leq (10)^{-8}$.

In python, the format for a `while` loop is:

```
while not done:
    # do something
```

**Solution.** One solution is as follows:

```
a = ainit
b = binit
tol = 1e-8
step = ...
max_iter = 500
it = 0
done = False
```

```python
    while not done:
        f, fgrad_a, fgrad_b = feval(a,b)
        a = a - step*fgrad_a
        b = b - step*fgrad_b

        # Check stopping condition
        it += 1
        norma = np.sum(fgrad_a**2)
        normb = np.sum(fgrad_b**2)
        done = (it >= max_iter) or ((norma < tol) and (normb < tol))
```