Assignment 6
Question 1.1

Odds ratio

\[
\frac{p(y_i | w^T x_i)}{p(-y_i | w^T x_i)}
\]

Linear model

\[
\log \left( \frac{p(y_i | w^T x_i)}{p(-y_i | w^T x_i)} \right) = w^T x.
\]

Objective function

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} -\log(p(y_i | w^T x_i)).
\]
**Question 1.1**

Odds ratio

\[
\frac{p(y_i | w^T x_i)}{p(-y_i | w^T x_i)}
\]

Linear model

\[
\log \left( \frac{p(y_i | w^T x_i)}{p(-y_i | w^T x_i)} \right) = w^T x.
\]

Objective function

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} -\log(p(y_i | w^T x_i)).
\]

Starting from equation 1

**First step**

replace \( p(-y_i | w^T x_i) \) with \( p(y_i | w^T x_i) \) using the fact that,

\[
p(y_i | w^T x_i) + p(-y_i | w^T x_i) = 1
\]

**Second step**

Apply “exp” on both sides to get rid of the log

**Third step**

Solve for \( p(y_i | w^T x_i) \) and plug it into the objective function
Question 1.2 One-vs-all Logistic Regression

% Classification using one-vs-all least squares

% Compute sizes
[n,d] = size(X);
k = max(y);

W = zeros(d,k); % Each column is a classifier
for c = 1:k
    yc = ones(n,1); % Treat class 'c' as (+1)
    yc(y ~= c) = -1; % Treat other classes as (-1)
    W(:,c) = (X'*X)
             /(X'*yc);
end

model.W = W;
model.predict = @predict;
end

function [yhat] = predict(model,X)
W = model.W;
[~,yhat] = max(X*W,[],2);
end
Question 1.2 One-vs-all Logistic Regression

% Classification using one-vs-all least squares

% Compute sizes
[n,d] = size(X);
k = max(y);

W = zeros(d,k); % Each column is a classifier
for c = 1:k
    yc = ones(n,1); % Treat class 'c' as (+1)
    yc(y ~= c) = -1; % Treat other classes as (-1)
    W(:,c) = (X'*X)\(X'*yc);
end

model.W = W;
model.predict = @predict;
end

function [yhat] = predict(model,X)
    W = model.W;
    [~,yhat] = max(X'*W,[],2);
end

---

Matrix X
Matrix W
Matrix y

dimensions = ?
dimensions = ?
dimensions = ?

n samples, k classes, p features

use findMin with LogisticLoss instead (see assignment 4 for the LogisticLoss function)
Question 1.2 One-vs-all Logistic Regression

% Classification using one-vs-all least squares
% Compute sizes
[n,d] = size(X);
k = max(y);

W = zeros(d,k); % Each column is a classifier
for c = 1:k
    yc = ones(n,1); % Treat class 'c' as (+1)
    yc(y ~= c) = -1; % Treat other classes as (-1)
    W(:,c) = (X'*X)\(X'*yc);
end

model.W = W;
model.predict = @predict;

function [yhat] = predict(model,X)
W = model.W;
[y~,yhat] = max(X*W,[],2);
end

dimensions = n x p
dimensions = p x k
dimensions = n x k

n samples, k classes, p features

use findMin with LogisticLoss instead
(see assignment 4 for the LogisticLoss function)
**Question 1.3 Softmax Loss and derivative**

- The softmax probability function is given as,

\[
p(y_i|W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c'=1}^k \exp(w_{c'}^T x_i)}.
\]

- Assume \( y_i \in \{1, 2, 3\} \)

- Convert \( y \) to binary form; i.e.

\[
\bar{y}_i = \begin{cases} 
[1, 0, 0] & \text{if } y_i = 1 \\
[0, 1, 0] & \text{if } y_i = 2 \\
[0, 0, 1] & \text{if } y_i = 3 
\end{cases}
\]
The softmax probability function is given as,

$$p(y_i|W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c=1}^{k} \exp(w_c^T x_i)}.$$ 

- Assume $y_i \in \{1, 2, 3\}$
- Convert $y$ to binary form; i.e.

  $$\tilde{y}_i = \begin{cases} 
  [1, 0, 0] & \text{if } y_i = 1 \\
  [0, 1, 0] & \text{if } y_i = 2 \\
  [0, 0, 1] & \text{if } y_i = 3 
  \end{cases}$$

  where $y_{i1}, y_{i2}, y_{i3}$

- Therefore,

  $$P(y_i|W, x) = p(\tilde{y}_i|W, x)$$

  $$= \tilde{y}_{i1} \frac{\exp(x_i W_{1}^\top)}{\sum_{c=1}^{k} \exp(x_i W_{c}^\top)} + \tilde{y}_{i2} \frac{\exp(x_i W_{2}^\top)}{\sum_{c=1}^{k} \exp(x_i W_{c}^\top)} + \ldots + \tilde{y}_{iC} \frac{\exp(x_i W_{C}^\top)}{\sum_{c=1}^{k} \exp(x_i W_{c}^\top)}$$

- $C$ is the number of classes
- $W^j$ refers to column $j$ of $W$
- $y_{ic}$ is the binary predicted target value for class ‘$c$’ of sample ‘$i$’
Question 1.3 Softmax Loss and derivative

- The softmax probability function is now formulated as,

\[ P(y_i|W, x) = p(\tilde{y}_i|W, x) = \tilde{y}_{i1} \frac{\exp(x_i W^1)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \tilde{y}_{i2} \frac{\exp(x_i W^2)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \ldots + \tilde{y}_{iC} \frac{\exp(x_i W^C)}{\sum_{c=1}^{k} \exp(x_i W^c)} \]
Question 1.3 Softmax Loss and derivative

- The softmax probability function is now formulated as,

\[ P(y_i|W, x) = p(\hat{y}_i|W, x) = \frac{\exp(x_i W^1)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \frac{\exp(x_i W^2)}{\sum_{c=1}^{k} \exp(x_i W^c)} + ... + \frac{\exp(x_i W^C)}{\sum_{c=1}^{k} \exp(x_i W^c)} \]

Only one of these terms is non-zero for any training example ‘i’
The softmax probability function is now formulated as,

\[ P(y_i|W, x) = p(\tilde{y}_i|W, x) = \tilde{y}_{i1} \frac{\exp(x_i W^1)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \tilde{y}_{i2} \frac{\exp(x_i W^2)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \ldots + \tilde{y}_{ic} \frac{\exp(x_i W^c)}{\sum_{c=1}^{k} \exp(x_i W^c)} \]

The negative logarithm of the probability:

\[
\log(p(\tilde{y}_i|W, x)) = ? \quad \text{(apply log)}
\]

\[-\log(p(\tilde{y}_i|W, x)) = ? \quad \text{(multiply by -1)}\]

The derivative of the negative log probability with respect to \( W_j^c \) can be broken into two cases:

\[
\frac{\partial -\log(p(\tilde{y}_i|W, x))}{\partial W_j^c} = \begin{cases} 
? & \text{if } y_i = c; \text{i.e. } \tilde{y}_{ic} = 1 \\
? & \text{if } y_i \neq c; \text{i.e. } \tilde{y}_{ic} = 0
\end{cases}
\]
Question 1.3 Softmax Loss and derivative

- The softmax probability function is now formulated as,

\[
P(y_i|W, x) = p(\tilde{y}_i|W, x) = \tilde{y}_{i1} \frac{\exp(x_i W^1)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \tilde{y}_{i2} \frac{\exp(x_i W^2)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \ldots + \tilde{y}_{iC} \frac{\exp(x_i W^C)}{\sum_{c=1}^{k} \exp(x_i W^c)}
\]

- The negative logarithm of the probability:

\[
\log(p(\tilde{y}_i|W, x)) = ? \quad \text{(apply log)}
\]

\[
- \log(p(\tilde{y}_i|W, x)) = ? \quad \text{(multiply by -1)}
\]

- The derivative of the negative log probability with respect to \( W^c_j \) can be broken into two cases:

\[
\frac{\partial - \log(p(\tilde{y}_i|W, x))}{\partial W^c_j} = \begin{cases} 
? & \text{if } y_i = c; \text{ i.e. } \tilde{y}_{ic} = 1 \\
? & \text{if } y_i \neq c; \text{ i.e. } \tilde{y}_{ic} = 0
\end{cases}
\]

\( W^c_j \) is column \( c \) of row \( j \) of \( W \), which corresponds to the coefficient of feature \( j \) of class \( c \).
Question 1.3 Softmax Loss and derivative

- The softmax probability function is now formulated as,

\[ P(y_i|W, x) = p(\tilde{y}_i|W, x) = \tilde{y}_{i1} \frac{\exp(x_i W^1)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \tilde{y}_{i2} \frac{\exp(x_i W^2)}{\sum_{c=1}^{k} \exp(x_i W^c)} + \ldots + \tilde{y}_{ic} \frac{\exp(x_i W^c)}{\sum_{c=1}^{k} \exp(x_i W^c)} \]

- The negative logarithm of the probability:

\[ \log(p(\tilde{y}_i|W, x)) = ? \quad \text{(apply log)} \]
\[ -\log(p(\tilde{y}_i|W, x)) = ? \quad \text{(multiply by -1)} \]

- The derivative of the negative log probability with respect to \( W_j^c \)

\[
\frac{\partial -\log(p(\tilde{y}_i|W, x))}{\partial W_j^c} = \begin{cases} 
? & \text{if } y_i = c; \text{ i.e. } \tilde{y}_{ic} = 1 \\
? & \text{if } y_i \neq c; \text{ i.e. } \tilde{y}_{ic} = 0
\end{cases}
\]

Hint: Use the indicator function to distinguish between the two cases
Question 1.4 - Softmax Classifier

```matlab
function [model] = leastSquaresClassifier(X,y)
    % Classification using one-vs-all least squares

    % Compute sizes
    [n,d] = size(X);
    k = max(y);

    % Each column is a classifier
    W = zeros(d,k);
    for c = 1:k
        yc = ones(n,1); % Treat class 'c' as (+1)
        yc(y == c) = -1; % Treat other classes as (-1)
        W(:,c) = (X'*X)
                  \ (X'*yc);
    end

    model.W = W;
    model.predict = @predict;
end
```
Question 1.4 - Softmax Classifier

Use findMin instead with the softmax loss grad function
Question 1.4 - Softmax Classifier

Change the contents of the green box on the left using that of the green boxes on the right.
Question 1.4 - Softmax Classifier

```matlab
function [model] = leastSquaresClassifier(X,y)
% Classification using one-vs-all least squares
[n,d] = size(X);
k = max(y);
W = zeros(d,k); % Each column is a classifier
for c = 1:k
    yc = ones(n,1); % Treat class 'c' as (+1)
    yc(y == c) = -1; % Treat other classes as (-1)
    W(:,c) = (X' * X) \
               (X' * yc);
end
model.W = W;
model.predict = @predict;
end
```

```matlab
W = zeros(d,k); % Each column is a classifier
% findMin expects 1-dimensional parameter vector
% Therefore use W( :) to get W's 1-dimensional form
W( :) = findMin( @yourSoftmaxLossFunction, W( :), .... )

model.W = W;
model.predict = @predict;
end
```

```matlab
function [loss, grad] = yourSoftmaxLossFunction(w, X, y, k)
% reshape w's dimensions to "p x k"
% p is the number of features, k is the number of classes
W = reshape(w, [p k]);

% Compute loss
loss = the softmax loss function you derived for Q1.3

% Compute gradient
grad = the softmax gradient function you derived for Q1.3
% reshape grad's dimensions to "1 x (p * k)"
% i.e. convert the grad matrix to a 1-dimensional vector
grad = reshape(grad, [p*k 1]);
```

Change the contents of the green box on the left using that of the green boxes on the right.
Question 1.5 - Cost of Multinomial Logistic Regression

Time complexity for processing one example = ?

Time complexity for predicting one example = ?
```python
# Training
# Run for T iterations
for t = 1 to T
    # Loop over training examples
    for i = 1 to n
        for k = 1 to K
            softmax_value(i,k) = compute softmax for class k for training example i over the 'd' features
        end for
        for j = 1 to d
            for k = 1 to K
                softmax_gradient(j, k) = compute the gradient for the coefficient of feature j of class k using softmax_value
            end for
        end for
    end for
# Testing
# Loop over test examples
for i = 1 to n_test
    for k = 1 to K
        softmax_value(i,k) = compute softmax for class k for test example i over the 'd' features
    end for
end for
for i = 1 to n_test
    yhat(i) = argmax of softmax_value(i,k) over 'k'
end for
```
Question 2
random walk

```
load simpleGraph.mat % Loads adjacency A and labelList
n = length(A);
p = zeros(n,2);
r = 100;
for i = 2:n
    for j = 1:r
        % Run random walk
        yhat = runRandomWalk(A,labelList,i);
        if yhat == 1
            p(i,1) = p(i,1) + 1;
        elseif yhat == -1
            p(i,2) = p(i,2) + 1;
        end
    end
end
% Output final probabilities
probabilities = p/r
```

<table>
<thead>
<tr>
<th>matrix labelList</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>1 1 0 0 0 1</td>
</tr>
<tr>
<td>1 0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0 1 0</td>
</tr>
</tbody>
</table>
Question 2
random walk

% Run random walk
yhat = runRandomWalk(A, labelList, i);
if yhat == 1
    p(i,1) = p(i,1) + 1;
elseif yhat == -1
    p(i,2) = p(i,2) + 1;
end

matrix A
Adjacency matrix

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 1 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 1 \\
6 & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

matrix labelList

\[
\begin{array}{c}
3 \\
4 \\
5 \\
\end{array}
\[
\begin{array}{c}
1 \\
1 \\
-1 \\
\end{array}
\]
Question 2
random walk

adjacency matrix

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 1 \\
5 & 0 & 0 & 1 & 0 & 1 \\
6 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]
Question 2
random walk

% Run random walk
yhat = runRandomWalk(A,labelList,i);
if yhat == 1
p(i,1) = p(i,1) + 1;
elseif yhat == -1
p(i,2) = p(i,2) + 1;
end

matrix A
Adjacency matrix

matrix labelList

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Question 2
random walk

% Run random walk
yhat = runRandomWalk(A,labelList,i);
if yhat == 1
    p(i,1) = p(i,1) + 1;
elseif yhat == -1
    p(i,2) = p(i,2) + 1;
end

matrix A
Adjacency matrix

matrix labelList

1 2 3 4 5 6
0 1 0 0 0 1
1 0 1 0 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 0 1 0 1
1 0 0 0 1 0
Question 2
random walk

% Run random walk
yhat = runRandomWalk(A,labelList,i);
if yhat == 1
    p(i,1) = p(i,1) + 1;
elseif yhat == -1
    p(i,2) = p(i,2) + 1;
end
Question 2
random walk

\[ yhat = \begin{cases} 
-1 & \text{with prob } = \frac{1}{1+2} = \frac{1}{3} \\
1 & \text{with prob } = \frac{1}{3} \\
0 & \text{with prob } = \frac{1}{3} 
\end{cases} \]

\[ v = 5 \]

\[ \text{prob} = \frac{1}{3} \]

matrix A

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix}
\]

matrix labelList

\[
\begin{array}{c|c}
3 & 1 \\
4 & 1 \\
5 & -1 \\
\end{array}
\]

% Run random walk
yhat = runRandomWalk(A,labelList,i); if yhat == 1
\[ p(i,1) = p(i,1) + 1; \]
elseif yhat == -1
\[ p(i,2) = p(i,2) + 1; \]
end