Homework 3

1. Linear regression models with one-hot embeddings. Suppose \( u \) is a categorical that can take \( k \) values, \( i.e., U = \{1, \ldots, k\} \). The one-hot embedding of \( u \) into \( \mathbb{R}^k \) is defined as \( \phi(u) = e_u \), where \( e_j \) is the \( j \)th unit vector. We will add a first feature which is constant, \( i.e., x_1 = 1 \), so the embedding we use is \( x = \phi(u) = (1, e_u) \in \mathbb{R}^d \) with \( d = k + 1 \).

We have a data set with \( n \) observations, \( x^1, \ldots, x^n, y^1, \ldots, y^n \).

(a) Show that the data matrix \( X \in \mathbb{R}^{n \times d} \) (with rows \((x^1)^T, \ldots, (x^n)^T\)) always has linearly dependent columns. This means that we cannot use (basic) least squares to fit a regression model, when we use one-hot embedding of a categorical.  

Hint. If \( X \) has linearly dependent columns, there is a vector \( z \neq 0 \) such that \( Xz = 0 \) (\( z \) is in the nullspace of \( X \)). Try to construct this vector; you can use it in your answer.

(b) Now suppose that we add ridge regularization (also called quadratic regularization) on \( \theta_{2:k+1} \) to our fitting method (with \( \lambda > 0 \)). We do not regularize the model coefficient associated with the constant feature \( x_1 = 1 \). Show that the associated least squares problem has linearly independent columns.

(c) Show that the sum of the last \( k \) coefficients \( \theta_i \) (\( i.e., \), those associated with \( u \)) is zero, \( i.e., \sum_{i=2}^{k+1} \theta_i = 0 \).

Hint. Consider the least squares problem

\[
\text{minimize} \| Az - b \|,
\]

where \( z \) is the variable, and \( A \) and \( b \) are problem data, where \( A \) has linearly independent columns. The least squares solution is \( \hat{z} = (A^TA)^{-1}A^Tb \), and the optimal residual is \( \hat{r} = A\hat{z} - b \). The orthogonality principle states that for any \( z \in \mathbb{R}^n \), we have

\[
(Az) \perp \hat{r}.
\]

Remark. The simple ridge regression problem above, with \( u \) consisting of one categorical, can be solved analytically. (We are not asking you to do this yet.) But the conclusions of parts (a)–(c) hold when the raw input \( u \) contains other features in addition to the categorical.

Solution.

(a) If we have the vector \( z = (-1, 1_k) \), then for every \( i \):

\[
(Xz)_i = -x^i_1 + 1^T x^i_{2:k+1} = 1 - 1 = 0.
\]

Since exactly one entry of \( x^i_{2:k+1} \) is one while all the others are zero.
(b) To simplify notation, let \( X' = X_{1:n,2:k+1} \), which is just the data matrix \( X \) without the constant term column.

We want to choose \( \theta \) to minimize

\[
\| [\mathbf{1} \ X'] \theta - y \|_2^2 + \lambda \| \theta_{2:k+1} \|_2^2,
\]

which we can rewrite as

\[
\| [\mathbf{1} \ X'] \theta - y \|_2^2 + \lambda \| [0_k \ I] \theta \|_2^2.
\]

We can combine the two terms into one, as

\[
\left\| \begin{bmatrix} \mathbf{1} & X' \sqrt{\lambda I} \end{bmatrix} \theta - (y, 0_k) \right\|
\]

The regularized \( \theta \) then minimizes \( \| \tilde{X} \theta - \tilde{y} \|_2^2 \), where

\[
\tilde{X} = \begin{bmatrix} \mathbf{1} & X' \sqrt{\lambda I} \end{bmatrix}
\]

and \( \tilde{y} = (y, 0_k) \). Note that \( \tilde{X} \) has linearly independent columns.

(c) By the orthogonality principle, we have that \((\tilde{y} - \tilde{X}\theta)^T \tilde{X} z = 0 \) for any \( z \in \mathbb{R}^{k+1} \).

Picking \( z = (-1, 1_k) \), as in part (a), yields

\[
\tilde{X} z = \begin{bmatrix} \mathbf{1} & X' \sqrt{\lambda I} \end{bmatrix} \begin{bmatrix} -1 \\ 1_k \end{bmatrix} = \begin{bmatrix} -1 + X' \mathbf{1} \\ \sqrt{\lambda} \mathbf{1}_k \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{\lambda} \mathbf{1}_k \end{bmatrix}.
\]

Where \(-1 + X' \mathbf{1} = 0 \) from part (a).

Now

\[
\tilde{y} - \tilde{X} \theta = \begin{bmatrix} y \\ 0_k \end{bmatrix} - \begin{bmatrix} \mathbf{1} & X' \sqrt{\lambda I} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_{2:k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \theta_1 + X' \theta_{2:k+1} \\ \sqrt{\lambda} \theta_{2:k+1} \end{bmatrix},
\]

so

\[
0 = (\tilde{y} - \tilde{X} \theta)^T \tilde{A} z = \begin{bmatrix} y - \mathbf{1} \theta_1 - X' \theta_{2:k+1} \\ \sqrt{\lambda} \theta_{2:k+1} \end{bmatrix}^T \begin{bmatrix} 0 \\ \sqrt{\lambda} \mathbf{1}_k \end{bmatrix} = \lambda \mathbf{1}^T \theta_{2:k+1}.
\]

Since \( \lambda > 0 \), then \( \mathbf{1}^T \theta_{2:k+1} = 0 \).
2. **Wildfire predictor.** Our task is to create a predictor which identifies the parts of the forest at risk from wildfires. We would like to predict the burned area given a set of features presented in the table below.

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>position x</td>
<td>x-axis spatial coordinate within the park</td>
<td>1-9</td>
</tr>
<tr>
<td>position y</td>
<td>y-axis spatial coordinate within the park</td>
<td>1-9</td>
</tr>
<tr>
<td>month</td>
<td>month of the year</td>
<td>1-12</td>
</tr>
<tr>
<td>FFMC</td>
<td>Fine Fuel Moisture Code</td>
<td>18.7-96.2</td>
</tr>
<tr>
<td>temp</td>
<td>temperature in Celsius degrees</td>
<td>2.2-33.3</td>
</tr>
<tr>
<td>wind</td>
<td>wind speed in km/h</td>
<td>0.4-9.4</td>
</tr>
<tr>
<td>rain</td>
<td>outside rain in mm/m²</td>
<td>0.0 to 6.4</td>
</tr>
<tr>
<td>area</td>
<td>burned area of the forest in ha on a log-scale</td>
<td>0.00-6.99</td>
</tr>
</tbody>
</table>

The Fine Fuel Moisture Code (FFMC) is a numeric rating of the moisture content of litter and other cured fine fuels. This code is an indicator of the relative ease of ignition and the flammability of fine fuel.

In `wildfire_data.json`, you will find an $n \times d$ matrix $U$ of data, with rows $(u^i)^T$, with $u^i \in \mathbb{R}^7$, and $n$-vector $v$ with the burned area data. The columns are listed in order, so `position x` is first, followed by `position y`, followed by `month`, etc.

Randomly partition the data into a training set consisting of 80% of the data and a validation set consisting of the remaining 20% of the data. We will work with $y = \psi(v) = v$.

(a) Standardize the training set. Report the means and standard deviations of each feature column before and after the standardization. For the validation set, standardize using the mean and standard deviation from the corresponding feature columns in the training set. Report the means and standard deviations of each feature column in the validation set before and after this transformation.

*Hint:* Import `Statistics` in your Julia file or notebook. Then you can compute the mean of the columns of a matrix $U_{\text{raw}}$ as `mean(U_{\text{raw}}, \text{dims}=1)`. You can compute the standard deviation of the columns of a matrix $U_{\text{raw}}$ as `std(U_{\text{raw}}, \text{dims}=1)`.

*Hint:* To standardize a matrix $U_{\text{raw}}$, compute $U_{\text{std}} = (U_{\text{raw}} .- U_{\text{mean}}) ./ U_{\text{std}}$

(b) For both the training and test sets, encode the month features as 12 one-hot embedded features. Standardize these features. Create a data matrix containing this embedded month feature along with the other six features and an constant (intercept) feature (a vector of ones). (There should be a total of 19 features.) Print one row of your data matrix and verify it contains reasonable values.

(c) For 100 values of the regularization parameter $\lambda$ uniformly spaced on a log scale between $10^{-1}$ and $10^5$, fit a ridge regression model to the data. Plot the train and test RMS errors versus lambda using a log scale for the $\lambda$-axis.
Report the smallest RMS error you achieve and the corresponding value of $\lambda$ that achieves it.

**Hint:** To produce $n$ values uniformly spaced on a log scale between $10^a$ and $10^b$, use $10 .^\text{range}(a, \text{stop}=b, \text{length}=n)$.

**Hint:** Suppose you have a vector `losses` of RMSE values for different $\lambda$. `minimum(losses)` returns the minimum value. `argmin(losses)` returns the index of the minimum value..

(d) For both the training and test sets, transform the month features using sinusoidal embeddings: replace $u_3^i$ by $(\sin(2\pi u_3^i/12), \cos(2\pi u_3^i/12))$. Using this embedding instead of the one-hot embedding, repeat the experiment in part (b). (Including the constant feature, the transformed data should have 9 features.)

(e) Provide a (short!) justification for the sinusoidal embedding. This can be one or two sentences.

**Solution.**

(a) After standardization, the mean of the data should be close to 0. The standard deviation should be close to 1.

(b) The single row of data should start with a 1. There should be a section corresponding to the one-hot embedding with eleven 0’s and one 1.

(c) For part (c) we got RMSE values between 1.2 and 2.0. These can vary as the dataset is shuffled randomly; repeatedly running the solution code produced different results. A few samples are shown here.

![Plot for part (c)](image1)

![Plot for part (c)](image2)

![Plot for part (c)](image3)

**Figure 1** Plot for part (c)
(d) For part (d) the results are very similar. The goal is to see how we can embed the data more efficiently (one column instead of 12) and get comparable results.

![RMS error (part d)](image)

**Figure 2** Plot for part (d)

(e) Possible explanations for the sinusoidal embedding is that the months are cyclic: after the 12th month, we have the 1st month. Embedding the data in a sinusoid reflects this property. Additionally, it results in fewer features than the one-hot embedding from part (c) so the data matrix is smaller.

Sample code:

```julia
include("readclassjson.jl")

using Random
using LinearAlgebra
using Statistics
using Plots

#---------------------------

lambds = 10 .^ range(-1, stop=5, length=100)

function RMSE(y_hat, y)
    return norm(y-y_hat)/sqrt(length(y_hat))
end

#---------------------------

data = readclassjson("wildfire_data.json")

U = data["U"]
```
v = data["v"]

# randomly split into 80/20
n = size(v)[1]
#idx = collect(1:n)
idx = shuffle(1:n)
n_train = Int(round(.8*n))

U_train = U[idx[1:n_train],:]
U_test = U[idx[n_train:end],:]
print("Size U train/test ", size(U_train), size(U_test))

v_train = v[idx[1:n_train],:]
v_test = v[idx[n_train:end],:]

U_months_train = U_train[:,3]
U_months_test = U_test[:,3]
# Done partitioning data.

# (a) Standardize the training set.
U_mean = mean(U_train, dims=1)
U_std = std(U_train, dims=1)
println("Before standardization:")
println(" Train mean: $(U_mean)")
println(" Train std : $(U_std)")
println(" Test mean : $(mean(U_test, dims=1))")
println(" Test std : $(std(U_test, dims=1))")

U_train = (U_train .- U_mean) ./ U_std
U_test = (U_test .- U_mean) ./ U_std
println("After standardization:")
println(" Train mean: $(mean(U_train, dims=1))")
println(" Train std : $(std(U_train, dims=1))")
println(" Test mean : $(mean(U_test, dims=1))")
println(" Test std : $(std(U_test, dims=1))")

# Encode as one-hot embedded and create a new data matrix
U_onehot_train = zeros((n_train, 12))
for i=1:n_train
U_onehot_train[i,Int(U_months_train[i])] = 1.
end

n_test = length(U_months_test)
U_onehot_test = zeros((n_test, 12))
for i=1:n_test
U_onehot_test[i,Int(U_months_test[i])] = 1.
end

U_train_b = [ones(n_train) U_train[:,1:2] U_train[:,4:end] U_onehot_train]
U_test_b = [ones(n_test) U_test[:,1:2] U_test[:,4:end] U_onehot_test]

# Print one row to inspect for weirdness
println("Part (b): One row ", U_train_b[1,:])

# Part (c): One-hot embedding
lambds = 10 .^ range(-1, stop=5, length=100)
rms_train_error = zeros(length(lambds))
rms_test_error = zeros(length(lambds))

for i=1:length(lambds)
    # Solve for the regularized problem
    theta_b = inv(U_train_b'*U_train_b + lambds[i]*I)* U_train_b'*v_train
    rms_train_error[i] = RMSE(U_train_b*theta_b, v_train)
    rms_test_error[i] = RMSE(U_test_b*theta_b, v_test)
end

plot(lambds, rms_train_error, label="RMS Train error", xscale=:log10)
title!("RMS error (part c)")
p = plot!(lambds, rms_test_error, label="RMS Test error")
savefig(p, "part_c.png")

# Part (d): Sinusoidal embedding
U_sin_train = sin.(2.0 * pi * U_months_train ./ 12.0)
U_sin_test = sin.(2.0 * pi * U_months_test ./ 12.0)
U_train_d = [ones(n_train) U_train[:,1:2] U_train[:,4:end] U_sin_train]
U_test_d = [ones(n_test) U_test[:,1:2] U_test[:,4:end] U_sin_test]

# Now we just repeat part (c). The plots should be really similar.
# This shows we can embed the data more space-efficiently and get the same results.
rms_train_error_d = zeros(length(lambds))
rms_test_error_d = zeros(length(lambds))

for i=1:length(lambds)
    # Solve for the regularized problem
theta_d = inv(U_train_d'*U_train_d + lambds[i]*I)* U_train_d'*v_train
rms_train_error_d[i] = RMSE(U_train_d*theta_d, v_train)
rms_test_error_d[i] = RMSE(U_test_d*theta_d, v_test)
end

plot(lambds, rms_train_error_d, label="RMS Train error", xscale=:log10)
title!("RMS error (part d)")
p = plot!(lambds, rms_test_error_d, label="RMS Test error")
savefig(p, "part_d.png")

3. Constant predictors. In this problem we will investigate the constant predictor. This is a regression model with $\phi(u) = 1$, meaning we don’t use the data at all. (We still have labels $y^{(i)}$.)

We’ll use empirical risk minimization to fit $\theta$ to our data $(y^{(1)}, \ldots, y^{(n)})$. While it may seem silly to have a problem with no data, in lecture we saw that constant predictors yield statistical functions such as the mean or median of the data $y^{(i)}$ depending on what loss function is used.

(a) Generate 500 random data points centered around 0.0 for use as your $y$. Each $y^{(i)}$ will be a scalar value. (You can do this with randn(100).) First, we use the mean-squared loss:

$$ \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \| \theta - y^{(i)} \|_2^2 $$

Your goal is to minimize the mean-squared loss

$$ \theta = \text{minimize} \frac{1}{n} \sum_{i=1}^{n} \| \theta - y^{(i)} \|_2^2 $$

Show that the optimal $\theta$ is the mean of $y$

$$ \theta = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} $$

*Hint:* Recall the least-squares solution for $\|X\theta - y\|_2^2$ is $\theta = (X^T X)^{-1} X^T y$. Is $\theta$ a scalar or vector? What is $X$ in this case?

(b) Compute the optimal $\theta$. Report this $\theta$ and the associated loss $\mathcal{L}(\theta)$. What does $\mathcal{L}(\theta)$ represent?

**Solution.**

(a) This is a special case of least-squares with $X = 1_n$: the ones vector. So:

$$ \frac{1}{n} (X^T X)^{-1} X^T y = \left(1_n^T 1_n\right)^{-1} 1_n^T y = \frac{1}{n} 1_n^T y = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} $$
This is the average of the $v^{(i)}$'s. $\theta$ represents the average of the data.

(b) If the student used `randn(500)` to generate the data, $\theta$ should be close to 0 and $\mathcal{L}(\theta)$ should be close to 1. With this optimal $\theta$, $\mathcal{L}(\theta)$ represents the variance of the data.

Sample code:

```python
y = randn(500)
theta = sum(y)/500.0
loss = 1.0/500.0 * sum((theta - y).^2)
```