Homework 7

1. **Fitting a multi-class classifier.** In `multi_class.json`, you will find a $300 \times 30$ matrix $U_{\text{train}}$ and a 300-vector $v_{\text{train}}$ consisting of raw training input and output data, and a $300 \times 30$ matrix $U_{\text{test}}$ and a 300-vector $v_{\text{test}}$ consisting of raw test input and output data, respectively. We will work with $x = \phi(u) = (1, u)$.

In `multi_logistic.jl` we have also provided you with a function

$$\text{multi_logistic}(X, Y, \text{reps}).$$

This function takes in input/output data $X$ and $Y$, and an array of representations of our embeddings $\text{reps}$. That is, the $k$th element of $\text{reps}$ is $\psi(v_k)$, for $k = 1, \ldots, K$, and $K$ is the number of labels.

The function outputs the parameterized predictor $g$ for the multi-class classifier. (Since $g$ is a predictor, its argument is simply $x$; to get a prediction $\hat{y}$ from a data input $x$, use $\hat{y} = g(x)$.) You must include the Flux and LinearAlgebra Julia packages in your code in order to utilize this function.

(a) Inspect $v_{\text{train}}$ and $v_{\text{test}}$. What is $\mathcal{V}$, the label set? What is $K$, the number of labels? Report the number of instances of each label in the training set. Do the same for the test set.

(b) Propose a simple embedding $\psi : \mathbb{R} \rightarrow \mathbb{R}^K$ for your raw output data $v$, so that $y = \psi(v)$.

(c) Apply your embedding from part (b) to both the training and test raw output data. Using `multi_logistic.jl`, fit a multi-class classifier to the training data. Use a nearest neighbor un-embedding to un-embed your data. Report the confusion matrices of your classifier on the training and test set. Also report the overall training error rate and the overall test error rate.

**Julia hints.** `unique(z)` returns the number of unique elements in an array $z$. For the nearest neighbor unembedding $\psi^\dagger(\hat{y})$, you may use

$$\text{psiinv}(\hat{y}) = \text{reps}[[\text{argmin}([\text{norm}(\hat{y}-\text{reps}[k]) \text{ for } k=1:\text{length(\text{reps})}])]],$$

where $\text{reps}$ is an array whose $k$th element is $\psi(v_k)$, for $k = 1, \ldots, K$. 
2. Predicting Covid-19 infection based on symptoms.

In this problem we would like to build a binary classifier for a highly relevant problem: given a list of symptoms, predict whether or not a patient has Covid-19.

In `covid_symptoms.json` you will find a matrix \( U \) with 20 binary features (answers to yes/no questions) and a vector \( v \) with the binary classification output: 1 if the patient was diagnosed with Covid-19 and -1 if they were not.

We have also provided a function in `classification_regression_fit.jl`.

\[
\text{classification_regression_fit}(X, Y, l, r, \lambda, \kappa; \text{numiters}=40)
\]

This function returns \( \theta^* \), the parameter vector for a linear classifier:

\[
\theta^* \leftarrow \minimize \ l(\hat{y}, y) + \lambda r(\theta)
\]

This is almost the same function you used a few weeks ago, with the addition of \( \kappa \): the parameter for your binary classification loss. It takes in input/output data \( X \) and \( Y \), a loss function \( l(\hat{y}, y) \), a regularizer \( r(\theta) \), \( \lambda \), \( \kappa \), and optionally, the number of epochs Flux should run. You will have to import `LinearAlgebra` and `Flux` to run the function.

(a) In binary classification we can use the Neyman-Pearson metric with parameter \( \kappa \):

\[
\kappa E_{fn} + E_{fp}
\]

\( \kappa \) expresses our relative dislike for mistaking a positive example for a negative one. In this context (predicting Covid-19 infection), which is more dangerous: a false positive or a false negative? Should \( \kappa \) be less than 1 or greater than 1? Provide a short justification explaining your answer.

(b) First, add a column of ones to \( U \) (as in previous homeworks). We will use a random 90/10 train/test split. Partition your data into \( U_{\text{train}}, v_{\text{train}}, U_{\text{test}} \) and \( v_{\text{test}} \).

We will use SVM. Define the hinge loss:

\[
\ell_{\text{hinge}}(\hat{y}, y) = \begin{cases} 
(1 + \hat{y})_+ & \text{if } y = -1 \\
\kappa(1 - \hat{y})_+ & \text{if } y = 1
\end{cases}
\]

and L2 regularizer (you can exclude the \( \theta_i \) corresponding to your column of ones, although it’s not required).

\[
r(\theta) = \|\theta\|_2^2
\]

You can set \( \lambda = 0.01 \) or experiment with different values; the focus of this problem is understanding how \( \kappa \) changes the false negative and false positive rate of our SVM classifier.
(c) For $\kappa$ between 0.01 and 10 (see the hint), use `classification_regression_fit` to compute the predictor $\theta$. Find the accuracy and number of false negatives and positives for the test set. Plot the false negatives and false positives on one graph (with $\kappa$ on the x axis).

Plot the accuracy on a separate graph (with $\kappa$ on the x axis).

(d) Evaluate your results from part (c) and choose the “best” classifier (for example, the one with highest accuracy or fewest false negatives). Report the accuracy and confusion matrix of the “best” classifier you chose.

*Hint:* Use `kappas = 10 .^ range(-2,1,length=10)` to compute your vector of $\kappa$ values.

*Hint:* `sign.(y_hat)` will convert a vector `y_hat` of floating-point values to a prediction of -1 or +1.

*Hint:* Using a logarithmic x axis for these graphs will help you see the trend. To plot $x$ and $y$ with a log-space $x$ axis, use `plot(x, y, xaxis=:log)`. 