Homework 3

1. Feature engineering for nearest neighbor predictors. Some common feature engineering transforms have no effect for some types of predictors. Here we examine some of these.

(a) Adding a constant feature. Is it very common to add a first feature that always has the value 1. This is done with the feature engineering transform \( T(x) = (1, x) \). How does this feature engineering mapping affect a \( k \)-nearest neighbor or soft nearest neighbor predictor? Justify your answer.

(b) Splitting into positive and negative parts. Consider the feature engineering transform \( T(x) = ((x)_+, (x)_-) \), where \((x)_+ = \max\{x, 0\}\) and \((x)_- = \min\{x, 0\}\), both of these acting elementwise, i.e., separately on each entry. How does this feature engineering mapping affect a \( k \)-nearest neighbor or soft nearest neighbor predictor? Justify your answer.

Hint. How are \( \|x - x^i\|_2 \) and \( \|T(x) - T(x^i)\|_2 \) related?

2. All-pairs interactions. The following problem will use \( U, \) and \( v \) found in all_pairs_data.json. The data has \( U \in \mathbb{R}^{n \times 3} \). Throughout this problem, use a 50-50 train/test split.

(a) Fit a linear least-squares model directly to the data matrix, with the first feature being a constant feature \( x_1 = 1 \). Since we’ve given you enough data and the data is approximately standardized, you do not have to worry about regularization or standardization.

The linear least-squares model is:

\[
\sum_{i=1}^{N} (\theta^T u^{(i)} - v^{(i)})^2
\]

If we define \( v \) as the vector of \( v^{(i)} \)'s and \( U \) as a matrix of \( u^{(i)} \)'s (every row of \( U \) is one observation \( u^{(i)} \)), we can rewrite this in matrix form:

\[
\|U \theta - v\|^2_2
\]

Report the train and test RMSE of this predictor.

Hint: In Julia, you can compute the least-squares solution as \( \text{theta} = U \backslash v \).

Hint: In this case, it is fine to split the data in half naively, using the first half as the train dataset and the second half as the test dataset (or vice-versa).

(b) Create an embedding which includes all of the interactions (products) between every pair of distinct variables, along with a constant feature and the variables themselves. For example, if \( u \in \mathbb{R}^2 \), then the embedding should be

\[
\phi(u) = (1, u_1, u_2, u_1 u_2)
\]
You’ll have to construct a new data matrix. This data matrix should include a column of ones and \(d\) columns for \(u_1, \ldots, u_d\). The remaining columns will contain the products 
\[
\begin{align*}
&u_1^2, \ u_1u_2, \ldots, u_3u_2, \ u_3^2.
\end{align*}
\]

Fit a new \(\theta\) to this data matrix. Report the train and test RMSE of this predictor. Compare it with the RMSEs you got in (a).

(c) Is it necessary to use all of the features? Print the \(\theta\) you found in part (b) and inspect it. Remember \(\theta\) represents the weights assigned to each feature. Are there any features that aren’t being used in the predictor? Remove them. (You’ll have to construct a smaller data matrix - see the hint.)

Report which indices \(\theta_i\) you removed. Remember that in Julia, vectors are 1-indexed, so \(\theta = [\theta_1, \theta_2, \ldots, \theta_{13}]^\top\).

Fit a new \(\theta\) with the unnecessary features removed and report the train and test RMSE of this predictor. Compare it with the RMSEs you got in (a) and (b).

\textit{Hint:} To remove column \(i\) from a matrix \(U\) in Julia, you can use array slicing:

\[
U = [U[:,1:i-1] \ U[:,i+1:end]]
\]

Be careful to slice the columns (second index) and not the rows. In the above line, we use : to index all the rows.

\textit{Hint:} It is perfectly fine to hard-code the array slicing: if you decide to remove index 9, you may hard-code the value 9.

\textit{Hint:} You should remove at least 3 features. You may remove more if you like.

3. \textit{Feature engineering.} In each of the parts (a)-(i), you are given a set of \(N = 1000\) raw data points \((u,v)\), where \(u \in \mathbb{R}^2\) and \(v \in \{\text{Red}, \text{Blue}\}\). We will be using the raw output feature map

\[
\psi(v) = \begin{cases} 
+1 & v = \text{Red} \\
-1 & v = \text{Blue} 
\end{cases}
\]

and we will use a linear classification predictor with the form

\[
\hat{y} = \begin{cases} 
+1 & \theta^\top x \geq 0 \\
-1 & \theta^\top x < 0 
\end{cases}
\]

where we will find \(\theta\) by training a machine learning algorithm over the data. For each part, introduce a feature mapping \(\phi\) for the input data, i.e., \(x = \phi(u)\), such that if we find the best possible \(\theta\), then the RMS error

\[
\left( \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - \hat{y}_i)^2 \right)^{1/2}
\]

is almost zero. For each part, give a brief justification of why you chose a specific \(\phi\).