# Homework 7: Feature Engineering

|  |  |
| --- | --- |
| Contact TA | William Rebelsky |
| Released date | Tuesday, 10/31 |
| Submission deadline | Midnight 11:59 PM, Thursday, Nov 9 |
| Total marks | 100 |

# Task Descriptions

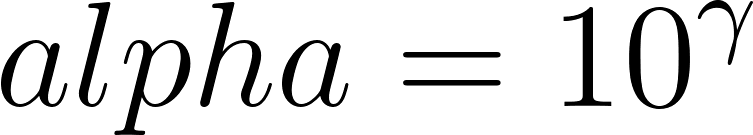
Complete the tasks by filling in the TODOs of the provided code template: [hw7.py](https://drive.google.com/file/d/19QKq9EsaVxYULNp2HB9iybm1NbUxQj_a/view?usp=sharing). Feel free to define additional functions if they make your code cleaner.

You could also reuse your code from previous homeworks and from [the textbook notes](https://github.com/jermwatt/machine_learning_refined). Particularly, the [textbook notes of Chapter 9](https://github.com/jermwatt/machine_learning_refined/tree/main/notes/9_Feature_engineer_select/chapter_9_library) have both the [PCA-sphering](https://github.com/jermwatt/machine_learning_refined/blob/main/notes/9_Feature_engineer_select/chapter_9_library/section_9_7_helpers.py#L149) and [standard normalization](https://github.com/jermwatt/machine_learning_refined/blob/main/notes/9_Feature_engineer_select/chapter_9_library/section_9_7_helpers.py#L125) source code. **Use with caution**.

You should use numpy functions instead of jax.numpy for the PCA and standard normalization. Here, normalization is a parameter-free data pre-processing step and should not be involved in the auto differentiation in Jax.grad

## Task 1：Comparing standard normalization to PCA-sphering on MNIST

Compare a run of **ten** gradient descent steps using the multi-class Softmax cost over **50,000** random digits from (1) **the MNIST dataset**, (2) **a standard normalized version**, and (3) **a PCA-sphered version** of the data. Note that the entire MNIST dataset consists of 70K data samples. You can use the first 50K data samples for model training.

For each run, find the largest fixed steplength of the form [](https://www.codecogs.com/eqnedit.php?latex=alpha%20%3D%2010%5E%7B%5Cgamma%7D#0) for [](https://www.codecogs.com/eqnedit.php?latex=%5Cgamma#0) an **integer** you find to produce descent. You can compare the final cost. The integer that produces the smallest final cost after 10 iterations should be picked.

Create a plot comparing the progress of each run in terms of the cost function and the accuracy. You should return 5 plots: 1 for each cost individually, 1 with all the costs together, 1 with all the accuracies. **Describe what you see.**

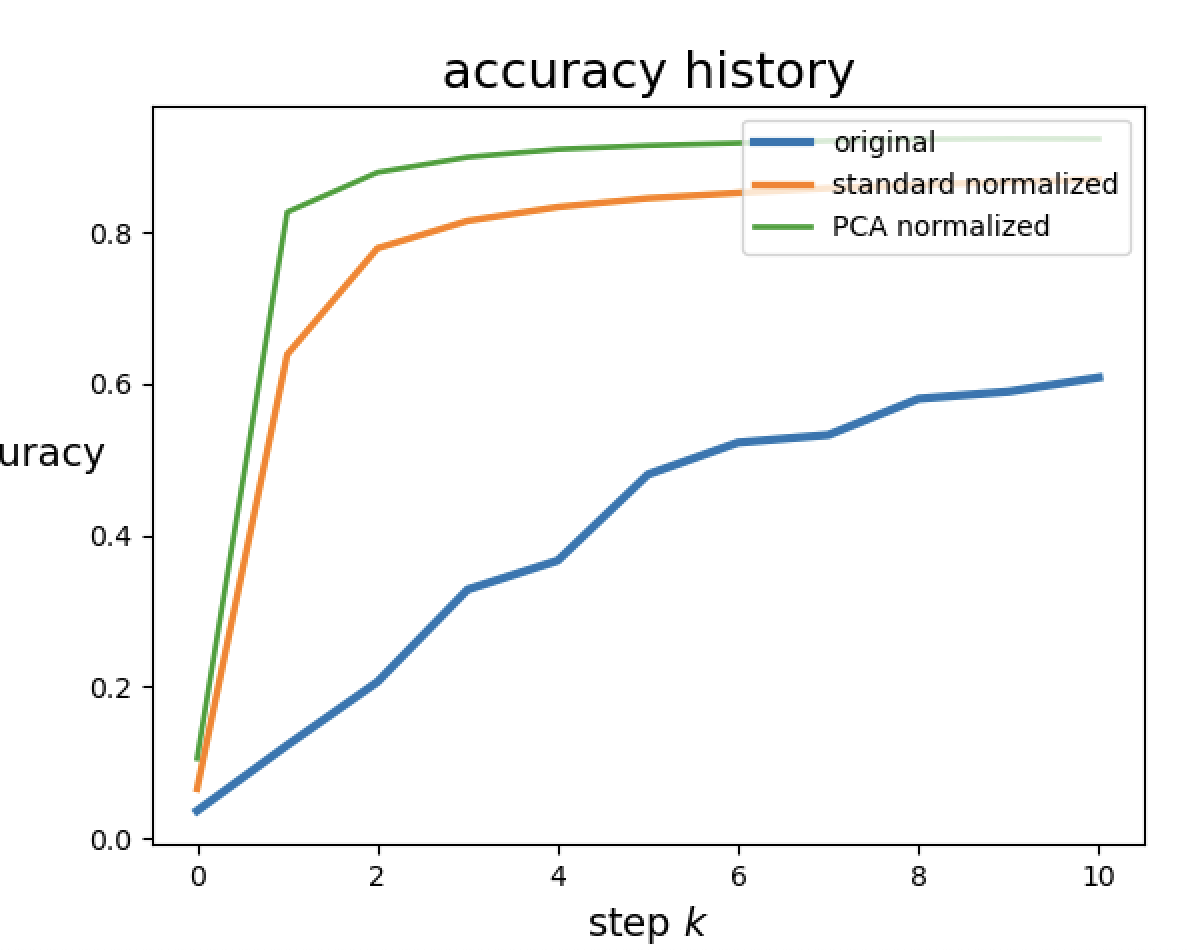
Additionally, make sure your initialization of each run is rather small, particularly the first run where you apply no normalization at all to the input as each raw input point of this dataset is large in magnitude. In the case of the raw data initializing at a point too far away from the origin can easily cause numerical overflow, producing NaN or INF values, ruining the rest of the corresponding local optimization run.

The starter code already provides you two functions to help with task1.

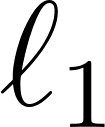
* model(x, w)
* multiclass\_softmax(w, x, y) : this is implemented with the logsumexp trick to avoid the overflow problem.

**Tips:**

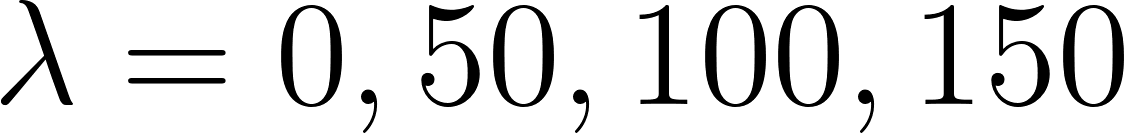
* Your cost should be dropping for the three cases (without normalization, standard normalized, and PCA normalized data).
* You should be expecting to see accuracy curves comparison like the following figure. Your curves might not look exactly the same due to randomness in initialization. Make sure each curve in your plots has the corresponding legend!
* You should use the np.random.seed() function to make sure your code is reproducible for graders.
* You don’t need to use l2 regularization in your cost function for this task.
* The **standard gradient descent with a fixed learning rate** should work on the dataset. You don’t need to diminish the learning rate or normalize the gradient. You don’t need mini-batch based training.



## Task 2: Exploring predictors of housing prices

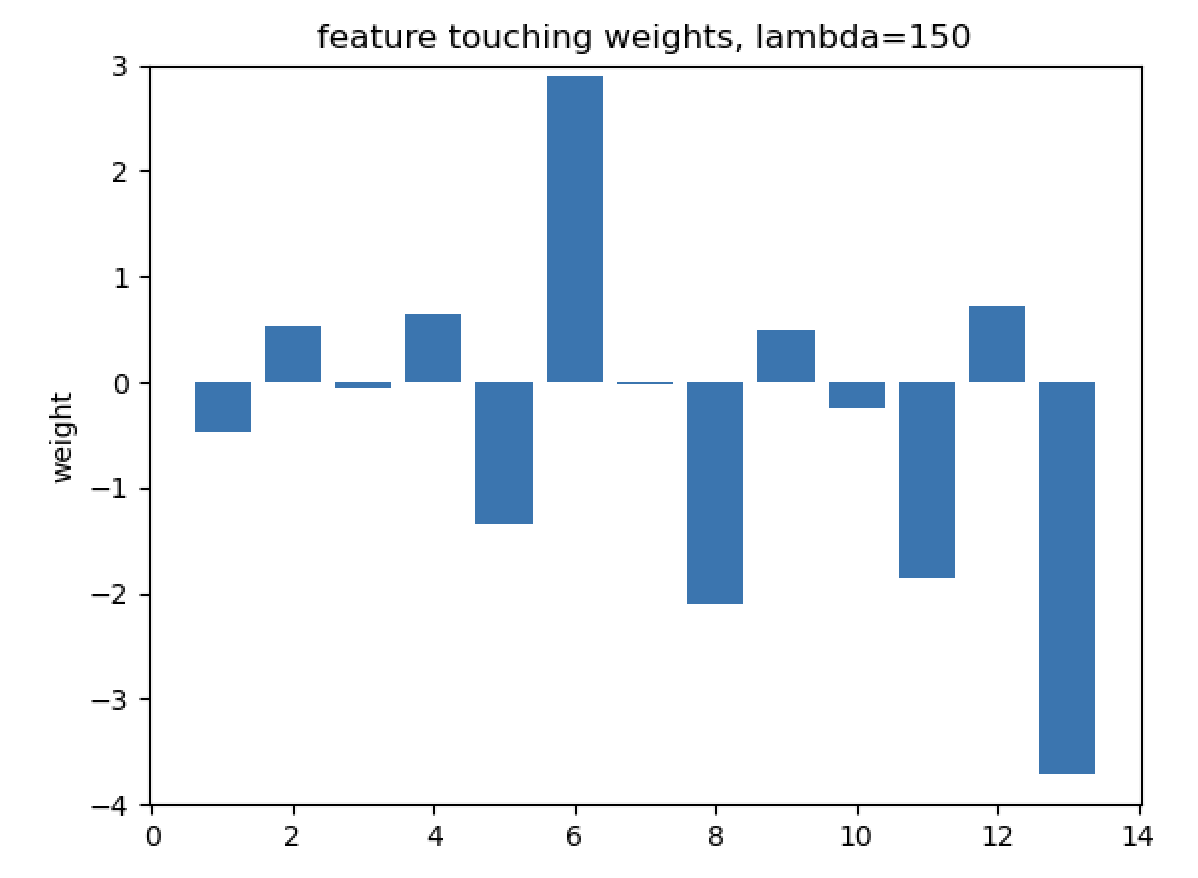
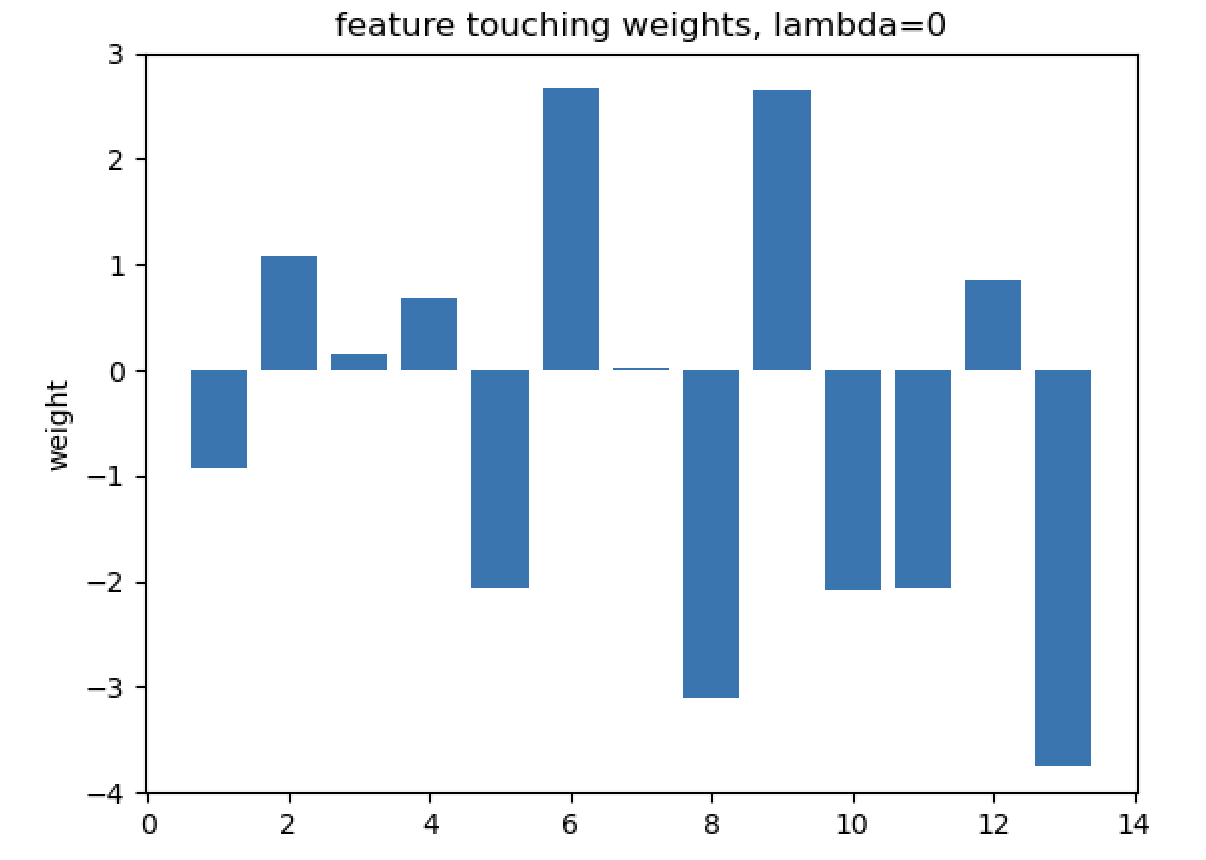
Implement the [](https://www.codecogs.com/eqnedit.php?latex=%20%5Cell_1%20#0) regularization procedure for linear regression employing the Least Squares cost. You can download the dataset [here](https://drive.google.com/file/d/19JiWJMBWN8B9zu-UWru-e4BlgzjjRufy/view?usp=sharing).

You should **standard** **normalize the data** first before feeding them into the linear regression model.

Examine [](https://www.codecogs.com/eqnedit.php?latex=%20%5Clambda%20%3D%200%2C%2050%2C%20100%2C%20150%20#0).

For each value of [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda#0), run gradient descent with a fixed number of steps and step length parameter to minimize the regularized cost. You will need to figure out these parameters to make sure your model can converge.

Plot the visualizations of feature importance as shown below for the four [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda#0) values. The figure below illustrates the results for only two values. The main conclusion of the below charts is that, ”*as lambda becomes larger, some features remain the important features as their weight values are not close to zero*.“



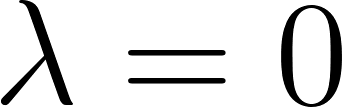
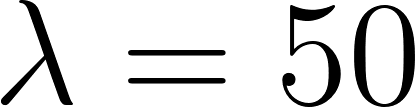
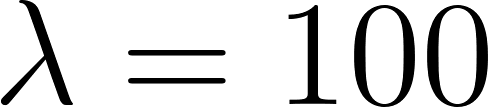
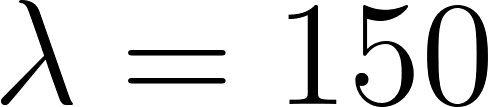
**You should produce histograms of feature touching weights for each value of** [****](https://www.codecogs.com/eqnedit.php?latex=%5Clambda#0) **and explain what you see based on your solutions. Note that you need not reproduce the exact same visualizations.**

**Tips:**

* You could use the same learning rate and max\_iters for different lambdas.
* You should use a standard normalizer. Use of PCA-normalizer will cause the trend to be less obvious.
* The above figure is plotted using plt.bar() function.
* You should set [ylim](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ylim.html) to be consistent across different lambdas. It will make it easier to compare the bar graphs across different lambdas. In the above figure, the line of code that set ylim is: plt.ylim([-4, 3])

## Deliverables

Make sure all plots have a clear legend

* Task 1:
  + You should return 5 plots: 1 for each cost individually, 1 with all the costs together, 1 with all the accuracies.
    - Plot of the cost functions over the 10 runs
    - Plot of the accuracies over the 10 runs
  + choice of gamma for:
    - Raw data without normalization
    - Standard normalized data
    - PCA-sphered data
  + An explanation of what you see
* Task 2:
  + Histogram of weights for:
    - [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%3D0#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%3D50#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%3D100#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%20%3D%20150#0)
  + Step length, number of steps, and final cost for the figures
  + Plot the cost history for each lambda to demonstrate that your model converged.
  + Explanation of what you see (and whether or not you can recreate the plots)

## 