NYU | TANDON SCHOOL OF ENGINEERING

New York University Tandon School of Engineering
Computer Science and Engineering

CS-UY 4563: Written Homework 2.
Due Tuesday, February 18th, 2020, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list the names of any collaborators at the top of your solution set, or write "No Collaborators" if you worked alone.

Problem 1: Practice Framing a Multiple Regression Problem (10pts)

An online retailer like Amazon wants to determine which products to promote based on reviews. They only want to promote products that are likely to sell. For each product, they have past sales as well as reviews. The reviews have both a numeric score (from 1 to 5) and text.

(a) To formulate this as a machine learning problem, suggest a target variable that the online retailer could use.

One target variable could be the number of sales per day of the product. Or better, the likelihood a customer buys the product after a visit to the product page (places like Amazon collect that info.). You might also take price into account – it probably makes sense recommend products which make more money per sale.

(b) For the predictors of the target variable, a data scientist suggests to combine each review's numeric score with the frequency of occurrence of words that convey judgement like "bad", "good", "broke", or "bargain". The data scientist would like to use these features in a linear model.

Describe one logical way to build feature vectors for each product. Think about normalization here!

Note: Make sure you describe features for each product, not for individual reviews!

Our first column could simply include the average rating for the product. There are other answers that would be fine here too: for example, you could have 5 separate columns which separately encode the percentage of review scores which are $1, 2, 3, 4, 5$ for the product.

For the word data, one reasonable option is to create a feature column for each of the selected words which convey judgement. I.e. we would have a column for "bad", "good", "bargain" etc. In each column, in the row for a particular product, we could include the number of times the word appears in all reviews for that product. It's probably a good idea to normalize for the total number of reviews or total length of reviews for the product (e.g. use word frequency divided by total number of reviews.) Otherwise, very heavily reviewed products might have more "good" words than less reviewed products, even if the heavily reviewed product is rated more poorly overall.

(c) Suppose that some reviews have a numeric score from 1 to 5 and others have a score from 1 to 10. How would change your features?

We would likely want to adjust things so all reviews are on the same scale – an easy way to do this is to multiply the numerical score for any review using a $1 - 5$ rating by 2. Then, when we average scores to form our first column, we would simply include the average rating for the product on the $1 - 10$ scale.

(d) Now suppose the reviews have either: a score from 1 to 5; a rating that is simply good or bad; or no numeric rating at all. How would you change your features?

Again, there are a lot of possible answers here. One option is to to translate "good" and "bad" to a numerical score on our $1 - 10$ scale (e.g. good $= 8$, bad $= 3$) and include in the average as above. Another option is to do some sort of 1-hot encoding. For example, we could have 12 columns for every possible score $1, \ldots, 10, \text{good}, \text{bad}$. In each of those columns we put the proportion of the reviews for each product that was given that score.

Problem 2: Thinking About Data Transformations (10pts)

You are trying to fit a multiple linear regression model for a given data set. You have already transformed your data by appending a column of all ones, which resulted in a final data matrix:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & & \vdots & \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{bmatrix}$$

However, your model does not seem to be working well. It obtains poor loss in both training and test.

(a) A friend suggests that you should try mean centering your data columns. In other words, for each i, compute the column mean $\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{i,j}$ and subtract $\bar{x}_i$ from every entry in column i. Note that we won't mean center the first column, as doing so would set the 1s to 0s. Using Python broadcasting you might mean center by running:

```
T = X[:,1:]
X[:,1:] = T - np.mean(T, axis=0)
```

Do you expect your friend's suggestion to improve the performance of the linear model. Will it help in all cases? Some cases? No cases?

Note: it would be fine to give a less formal answer, as long as it's clear you get the point.

Mean centering is unlikely to help in any case, at least for any of the standard loss function we have considered. In particular, it won't make a difference for any loss of the form $L(X\beta - y)$, where X is the data matrix, y is the vector of n targets, and β is a parameter vector. Below we explain why:

Let $\bar{X}$ denote our matrix with mean centered columns. Denote the i^{th} column of $\bar{X}$ by $\bar{X}_i$. Clearly $\bar{X}_i = X_i - c_i\vec{1}$ for some constant c_i (in this case $c_i = \bar{x}_i$). Here $\vec{1}$ is the all ones vector.

Now, given any β, we claim that there always exists some $\bar{\beta}$ such that $X\beta = \bar{X}\bar{\beta}$. Specifically set $\bar{\beta}_i = \beta_i$ for all $i > 1$ and set $\bar{\beta}_1 = \beta_1 + \sum_{i=2}^{n} c_i$. At the same time, given any $\bar{\beta}$, there is always some β such that $X\beta = \bar{X}\bar{\beta}$. Specifically set $\beta_i = \bar{\beta}_i$ for all $i > 1$ and set $\beta_1 = \bar{\beta}_1 - \sum_{i=2}^{n} c_i\bar{\beta}_i$.

We conclude that for any loss of the form stated above,

$$\min_{\beta} L(X, y, \beta) = \min_{\beta} L(\bar{X}, y, \beta).$$

So scaling certainly won't help improve training loss. It's possible it leads to some small difference in test loss, but its hard to say either way: we don't expect the test loss to be small if training loss isn't.

(b) Another friend suggests normalizing your data columns to have unit standard deviation. In other words for each i, compute the column standard deviation $\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_{i,j} - \bar{x}_i)^2}$ and *divide* every column by σ_i. Using Python broadcasting you might run:

```
T = X[:,1:]
X[:,1:] = T/np.std(T, axis=0)
```

Do you expect your friends suggestion to improve the performance of the linear model. Will it help in all cases? Some cases? No cases?

The argument is essentially the same as above. Let $\bar{X}$ be our data set with each column multiplied by c_i. For any β there is some $\bar{\beta}$ such that $X\beta = \bar{X}\bar{\beta}$: set $\bar{\beta}_i = \frac{1}{c_i}\beta_i$ for all $i > 1$. Similarly, for any $\bar{\beta}$ there is some β with $X\beta = \bar{X}\bar{\beta}$: set $\beta_i = c_i\bar{\beta}_i$ for all $i > 1$. Again we conclude that:

$$\min_{\beta} L(X, y, \beta) = \min_{\beta} L(\bar{X}, y, \beta).$$

so the transformation cannot improve training loss.

(c) Would your answers to either of the two questions above change if you were fitting the model with ℓ_2 regularization? In other words, instead of minimizing the squared loss $L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$ alone, you were minimizing $L(\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_2^2$.

Again, no need to be super formal here. You just need to identify that regularization changes things.

Under ℓ_2 regularization, it's possible that both mean centering or scaling by standard deviation could improve performance. It's also possible they could make things worse, or do nothing at all. For example, suppose we have a column i in X which has very small standard deviation. Scaling that column by the inverse of the standard deviation will effectively allow our regularized solution to use a larger coefficient for β_i. If that column is a good predictor for y, this change could substantial improve test loss.

Note: If you struggled with this problem, the easiest way to arrive at these answers was to test things out in Python. You could start with the multiple linear regression lab, explicitly mean-center or scale columns, and see that the train loss does not change.

Problem 3: Practice With Gradients (10pts)

For $\mathbf{X} \in \mathbb{R}^{n\times d}$ and target vector $\mathbf{y} \in \mathbb{R}^n$, consider fitting a linear model of the form:

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \mathbf{X}\boldsymbol{\beta}$$

under the so-called ℓ_p loss: $L_p(\boldsymbol{\beta}) = \|\mathbf{y} - f_{\boldsymbol{\beta}}(\mathbf{x})\|_p^p$. Here $\|\cdot\|_p^p$ denotes the ℓ_p norm raised to the p power. I.e. for any even integer $p = 2, 4, 6, \ldots$

$$\|\mathbf{z}\|_p^p = \sum_{i=1}^{n} z_i^p$$

(a) Derive an expression for $\nabla g(\mathbf{z})$ where $g(\mathbf{z}) = \|\mathbf{z}\|_p^p$.

We have that $\partial g/\partial z_i = pz_i^{p-1}$. So if we stack all of the partial derivatives into a column vector we have: $\nabla g(\mathbf{z}) = p\mathbf{z}^{p-1}$ where $\mathbf{z}^{p-1}$ denotes raising every entry in the vector $\mathbf{z}$ to the $p-1$ power.

(b) Derive an expression for $\nabla L_p(\boldsymbol{\beta})$. **Hint:** Use chain rule.

Let $h(\mathbf{z}) = \|\mathbf{y} - \mathbf{z}\|_p^p$ for some fixed vector $\mathbf{y}$. Using the same argument as in part (a), we have that $\nabla h(\mathbf{z}) = -p\left(\mathbf{y} - \mathbf{z}\right)^{p-1}$. Then we can apply the multivariate chain rule from class, and in particular the special case where the input of a function is multiplied by a matrix.

Specifically, we have that $L_p(\boldsymbol{\beta}) = h(\mathbf{X}\boldsymbol{\beta})$. So $\nabla L_p(\boldsymbol{\beta}) = \mathbf{X}^T\nabla h(\mathbf{X}\boldsymbol{\beta}) = -p\mathbf{X}^T\left(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right)^{p-1}$ where again the power denotes an entrywise operation.

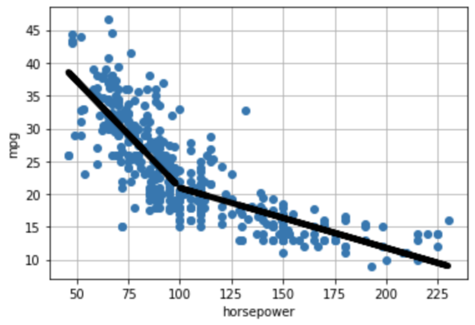
Problem 4: Piecewise Linear Regression via Feature Transformations (15pts)

Your goal is to fit a *piecewise* linear model to a single variate dataset of the form $(x_1, y_1), \ldots, (x_n, y_n)$ where all values are scalars. We will only use two pieces. In other words, for some known value λ,

$$f(x_i) = \begin{cases} a_1 + s_1 x_i & \text{for } x_i < \lambda \\ a_2 + s_2 x_i & \text{for } x_i \geq \lambda \end{cases}$$

with the additional **constraint** that $a_1 + s_1\lambda = a_2 + s_2\lambda$. This constraint ensures that our two linear models actually "meet" at $x = \lambda$, which means we get a continuous prediction function.

For example, when $\lambda = 100$, a piecewise linear fit for our MPG data might look like:



(a) Show that this model is equivalent to the following **unconstrained** model:

$$f(x_i) = \begin{cases} a_1 + s_1 x_i & \text{for } x_i < \lambda \\ a_1 + s_1\lambda - s_2\lambda + s_2 x_i & \text{for } x_i \geq \lambda \end{cases}$$

From our constraint, we always have that $a_2 = a_1 + s_1\lambda - s_2\lambda$. So we can eliminate the a_2 parameter: it is completely determined by the other model parameters. Replacing a_2 in $f(x_i)$ with $a_1 + s_1\lambda - s_2\lambda$ gives the alternative formulation.

(b) Show how to fit an optimal f under the squared loss using an algorithm for multiple linear regression. In particular, your approach should:

- Transform the input data to form a data matrix $\mathbf{X}$ with multiple columns.
- Use a multiple regression algorithm to find the $\boldsymbol{\beta}$ which minimizes $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.
- Extract from the optimal $\boldsymbol{\beta}$ optimal values for a_1, s_1, s_2.

You need to describe 1) a correct data transformation and 2) a correct mapping from $\boldsymbol{\beta}$ to a_1, s_1, s_2. **Note that in our model λ is known. It is not a model parameter which needs to be optimized.**

Construct $\mathbf{X}$ as follows:

- First column equal to $\vec{1}$.
- Second column equal to x_i for any position i such that $x_i < \lambda$. Otherwise, equal to λ for any position i such that $x_i \geq \lambda$.
- Third column equal to 0 for any position i such that $x_i < \lambda$. Otherwise, equal to $x_i - \lambda$ for any position i such that $x_i \geq \lambda$.

Now consider $\mathbf{X}\boldsymbol{\beta}$. With some thought we see that the i^{th} entry of this vector is equal to $f(x_i)$ with parameters $a_1 = \boldsymbol{\beta}_1$, $s_1 = \boldsymbol{\beta}_2$, and $s_2 = \boldsymbol{\beta}_3$.

(c) Implement your algorithm in Python and apply it to the dataset from `demo_auto_mpg.ipynb`. Produce a piecewise linear fit for MPG as a function of Horsepower using the value $\lambda = 100$. Plot the result. You can attach a Jupyter notebook to your submission, or simply include the printed code and plot.

See my solution in `part_4_soln.ipynb`.

(d) (**5pts bonus**) Modify your approach to handle the case when λ is unknown. Again obtain a fit for MPG vs. horsepower. What value of λ gives the optimal fit? Include any modified code and a plot of your result.

Easiest way to do this is by brute forcing over a grid of possible values of λ.