TECHIN 513 – Lab 4

(Stochastic) Gradient Descent, Linear Regression

Lab Instructions

1. Please approach the course instructor or grader for assistance if you need help.
2. We highly recommend working in groups of **2 members**. When collaborating with students in other groups, please do not share code but only discuss the relevant concepts and implementation methods.
3. Please document your code well by **using appropriate comments, variable names, spacing, indentation, docstrings**, etc. Please refer to TECHIN 509 for more information.
4. The starter code is not binding on you. Feel free to modify it as you wish. Everything is fine so long as you are getting the correct results.
5. Please upload the .ipynb file to canvas. Use Markdown cells appropriately to answer the questions. Each team only needs to submit one report.
6. Please enter the names of all the team members in your Jupyter notebook so that you do not lose credit for your work.

Lab Submission Requirements

In your notebook, briefly describe your results and discuss the problems you encountered, the solutions that you came up with and any choices you made. Also, answer the questions asked in each section. Make sure to include the code of all tasks in code cells.

Lab Objectives

The goal of this lab is to deepen your understanding of gradient descent algorithm, and apply linear regression on real-world data to solve problems.

Task 1: Multiple Linear Regression to Predict House Price (10 points)

In this task, we will perform multiple linear regression to predict house prices. Different from simple linear regression where feature and target variables are scalars, multiple linear regression fits the relationship between one target variable (scalar) and two or more features. We normally organize these features using a vector.

1. Load “Housing.csv” file.
2. Examine the data to ensure it does not contain null entries.
3. Examine price, area, bedrooms, bathrooms, stories, and parking to check if any outlier is present. Remove the outliers from the data. (Hint: We can remove the outliers using a Boolean mask. Refer to TECHIN 509 or other resources for more details.)
4. Use train_test_split() function from sklearn.model_selection to split the dataset into training (80%) and testing (20%). Use random seed 42 for splitting.
5. Convert entries in string format to numerical values, and rescale all entries to avoid one feature dominate the others. (Code for this step is given. The description is for completeness and your reference.)

6. Use area, bedrooms, bathrooms, stories, and parking as features. Fit a linear model using the training data to predict the price. You can use LinearRegression from sklearn.linear_model to fit the model.
7. Rescale the testing data. Evaluate the learned model by reporting MSE and R^2.
8. Plot the actual price versus the predicted price over the testing data. Discuss the prediction quality.
9. Report the fitted linear model in a Markdown cell.

Task 2: Locally Weighted Linear Regression (10 points)

In lecture, we discussed that linear regression is applicable when the data exhibits linear relationship. In this task, we will explore locally weighted linear regression which can be applied when data exhibits non-linear relationship.

Locally weighted linear regression extends linear regression by focusing on **local neighborhoods** of data samples. Unlike standard linear regression, which fits a single global model to the entire dataset, locally weighted linear regression **dynamically assigns weights** to training examples based on their proximity to a testing example, fitting a local model for each prediction.

For each query point x, locally weighted linear regression calculates a weight for every training example x^i based on its distance from x. A commonly used weighting function is the Gaussian kernel:

$$w^i = exp\left(-\frac{||x - x_i||}{2\tau^2}\right),$$

where $||x - x_i||$ represents the l-2 norm (see more details here: https://mathworld.wolfram.com/L2-Norm.html). Here, τ is the bandwidth parameter that controls how localized the model is:

- Smaller τ: More localized influence (focuses on closer neighbors).
- Larger τ: Wider influence (includes farther neighbors).

For each query point x, a linear regression model is fit to the training data using the calculated weights. The cost function becomes: $J(\theta) = \sum_{i=1}^{n} w^i\left(y^i - \theta^T x^i\right)^2$, where w^i are the weights. The local model predicts the output y for the testing example x.

Load the tips.csv data. Extract total_bill and size as our features. Our goal is to predict the tip amount (tip column in data). Complete the following steps:

1. Implement locally weighted linear regression to fit a model.

2. Compare the predicted value and the actual tip amount. Choose appropriate visualization for comparison.

Discussion

- Repeat Step 2 with different choices of τ. Discuss the sensitivity to the choice of τ.
- Although locally weighted linear regression can be applied when non-linearity is observed, can you name one disadvantage of locally weighted linear regression compared to linear regression? (Hint: Computation efficiency)

Bonus Task: Gradient Descent and Stochastic Gradient Descent (4 points)

In this task, we will implement gradient descent (GD) and stochastic gradient descent (SGD) algorithms:

1. Implement GD and SGD from scratch. Please use docstring and appropriate comments to explain your function.
2. Load "penguins.csv".
3. Use GD and SGD to fit a linear model, respectively. The linear models aim to predict how body mass (target variable) of penguins is correlated to flipper length (feature).
4. Plot how the loss functions evolve over iterations when using GD and SGD. Clearly label your plot. Evaluate the mean squared error achieved using GD and SGD over the dataset. Note: you may need to explore different choices of hyperparameters such as learning rate and number of iterations to ensure your functions find near-optimal parameters.

Hint: Your code may run into overflow error and return nan as a result. This is a typical error when the values of variables become too large and exceed the limit of numpy. The following strategies may address this error:

1. Examine your data to check the extreme values of features and target variables. If they are of significant different orders of magnitude, it is likely to cause overflow error. If you believe this fits your case, you may use StandardScalar from sklearn.preprocessing to standardize the features. Note that if you standardize the features during training, you should repeat this step during testing as well.
2. Alternatively, you should also check the values of parameters. If the values of parameters become unreasonably large, you can clip the values of parameters by manually setting upper and lower bounds.
3. Tune the learning rate. Scale down the learning rate and see if this can handle the error.
4. Try a few different initial values of parameters θ.

Discussion

1. Show the formula of your loss function in a Markdown cell. Discuss your observations in Step 5 based on the comparison between GD and SGD.
2. Discuss whether you encounter overflow error, and what strategy is adopted to handle the error.