Lab06 – Cost Functions and Parameter Space: Solving models the hard way

CS3400 Machine Learning

Learning Outcomes

- Become familiar with the use and meaning of both error and cost functions
- Develop the skill of using cost functions to characterize model's performance for a given dataset.
- Develop the skill for coding mathematical models and representing them as a cost function.
- Explore cost function space and see how model parameters effect model error.

Overview

The previous lab had you experiment with mathematical models and converting them into code. We approached modelling using a 5-step framework to help organize our thoughts when considering a model. These steps included:

1. Data Familiarity
2. Model Familiarity
3. Model Implementation
4. Model Output (and Visualization)
5. Error Between Model and Data

This lab will focus on converting this approach to a standardized form – the cost function – and further exploring the implications and approaches needed to solve for good model parameters. While in the previous lab you were required to make 'blind' guesses at good model parameters, the experiments that you will run in this lab will have you use a more directed approach - a *grid search*. The grid search will have you evaluate a set of model parameters, plot the model error in the *parameter space*, and find which model parameters gives minimal modelling error.

Instructions

You have been provided with a stubbed .py script (cost_functions.py), a set of unit tests for your cost functions (test_ cost_functions.py), and two data sets (gaussdist.csv and advertising.csv). The file cost_functions_stub.py contains a partial implementation of the two cost functions that you will be writing for this lab. Your first job will be to convert the models you created in Lab05 into our newly defined *cost function API*. With the cost functions working, you will run 2

experiments to explore the cost function/parameter space and solve for the parameters that produce the best predictions from your model.

Cost Function API

Take a moment to look through the cost_functions.py file that you were provided. The methods required share many similarities with the approach that you took last week. Take special note of the following methods and their association to our Lab05 approach.

- __init__(self, features, y_true) – *Data Familiarity*
- predict(self, features, params) – *Model Implementation / Model Output*
- _mse(self, y_true, pred_y) – *Error Between Model and Data*
- cost(self, params) – *Error Between Model and Data*

From these methods you may notice several things.

- Underscores are often used in python to indicate that the given method is intended to be private. The predict function is often left private but for this lab you will use the predict method directly in your notebook.
- MSE may be new to you – or maybe you found it as another error metric last week. MSE refers to Mean Squared Error. It is a common error metric that you will be required to implement for this lab.
- The cost method and the MSE method may look very similar to you, but you should appreciate the parameters that are passed to each method. Our cost functions are "tied" to a dataset when instantiated, this leads to the features and response variables being held constant over many evaluations. The cost method should call _mse so that an end user can supply different parameter values to explore the parameter space, and the cost function will return the error calculated using the new parameters and the stored supervised data.

Additionally, we are having you organize the cost function in this way is to play well with future labs. You will see that cost functions are incredibly powerful when used in conjunction with other methods to help us solve the model parameters.

Jupyter Notebook – Running experiments

Create a Jupyter notebook named <lastname>_lab06. The notebook should have a title, your name, and an introduction. You will be running the following experiments in the notebook.

Each of the following steps should be performed in individual cells in your jupyter notebook. If you wish to write a .py script to perform some of these operations and import it into your notebook, that is fine, but the individual segments should be in their own cells.

Experiment 1: Coarse Grid Search - Gaussian Distribution

Notebook Setup

1. Load the gaussdist.csv into your notebook.
2. Identify the columns associated with the response variable and the feature variable. Hint Look at the expected output for a gaussian – it is not monotonic. The relationship between independent and dependent values is 1-to-1.
3. The features and response variables should be stored in separate numpy arrays.
4. Instantiate your cost function class for *GaussianCostFunction* using the provided data.
5. Plot the feature (x-axis) versus the response (y-axis) of the gaussdist.csv dataset. Make sure to label your plot.
6. Use the parameter set (μ=1 and σ=0.75) and the _predict method to generate your model predictions and plot them on the same plot as the original data.
7. Use the cost method to solve for the error and add this error to the plot's title.

Grid Search - Coarse

Blindly guessing at good parameters is no way to go through life. Now that you have a function that accepts a set of parameters and returns the error, let's explore the error space by evaluating an evenly spaced set of model parameters – a grid search.

1. Create two new vectors that hold a set of values for each parameter (μ and σ).
2. The set of values should be evenly spaced so that you are sampling uniform regions in the "parameter space". For this experiment, range your μ and σ vectors from 0 to 10 and 0.5 to 2 respectively. Sample at 100 divisions for μ and 50 divisions for σ. Hint: take a look at numpy.linspace.
3. Plot your resulting 2-dimensional array of error values using a heatmap. *Make sure to label your axis and legends!* For a place to begin with heatmaps, look at seaborn.heatmap. Notice here that the color values for the heatmap represent *error*.
4. Using the results for the heatmap, pick a combination of parameters that produce a low error.
5. Use the selected parameters to calculate your model predictions and plot this output against the given data on a separate plot. Make sure to label the plot with the model error from using your choice of parameters.

Experiment 2: Refined Grid Search – Gaussian Distribution

Grid Search – Refinement Pass

An issue with grid search is that it can be dependent on picking the correct line space, or *resolution*. You may not always get it on the first try.

1. Repeat the steps in Grid Search – Coarse using the same number of grid samples, but this time use:

- $6 \geq \mu \geq 5$
- $1.75 \geq \sigma \geq 1$

2. Plot your resulting 2-dimensional array of error values using a heatmap. Through visual inspection of the heatmap, pick a combination of parameters that produce a low error.
3. Use the selected parameters to calculate your model predictions and plot this output against the given data on a separate plot. Make sure to label the plot with the model error from using your choice of parameters.
4. Using the argmin function in numpy, search through the 2-dimensional array of error values to find the minimum. Hint: What do the indices of this error matrix correspond to?

Experiment 3: "Blind" Grid Search – Multivariate Linear Model

Once we progress into higher dimensional spaces, we have to learn to rely more on the math behind our models than visualizations in feature/parameter spaces – hence "blind". The methods that you have used for the 1-independnent / 1-dependent variable case for experiment 1 and 2 can still be applied to higher dimensional feature spaces. For this experiment, you will perform a grid search on the multivariate linear model but I would *discourage* you from plotting all of the dimensions.

- For this experiment you will use the cost function you wrote for the multivariate *LinearCostFunction*.
- Instead of plotting features versus response, plot model prediction versus response using a parameter set of $[\beta_0, \beta_1, \beta_2, \beta_3]$ as $[0.1, 0.1, 0.1, 0.1]$. On this same figure, plot the *line of identity*. The *line of identity* begins at the origin and has a slope = 1.
- Instantiate the cost function that you created this week for the linear model using the advertsiement.csv dataset.
- Preform a grid search over all four coefficients ($\beta_0, \beta_1, \beta_2, \beta_3$). Remember that you can have both positive and negative value coefficients in a linear model. I would also suggest using no more than 50 divisions in each dimension.
- Use the same argmin approach from experiment 2 to find the combination of model parameters that results in the lowest possible error.
- Create another plot of the model prediction versus response using your found set of parameters and label the plot with the resulting error. Make sure to also plot the *line of identity*.

Questions:

After you run all the experiments create a markdown cell to answer questions in. Copy and paste each question into the cell prior to answering it. The following questions will be graded:

a) By looking at the provided cost_functions.py, use 1-2 sentences to describe in detail the purpose of each of the methods. To guide this description, discuss the method input, method output, and what function each method serves for the cost function.
b) For the heatmaps that you generated for this lab, what do they describe? What do the "valleys" and "peaks" of this heat map represent?
c) For experiment 2, you increased the number of samples within the specified range.
    a. Describe how the heatmap representation changed due to this increase in sampling.
    b. What benefit did this higher sampling rate have for finding the set of parameters with the minimum error?
    c. Was this sampling rate high enough? Defend your answer!
d) The Gaussian distribution model is limited to two dimensions while the multivariate linear model implemented for this lab is 4 dimensional.
    a. Describe a limitation of the grid search method as you add additional dimensions. Hint: Think about the time complexity required for the grid search as you add additional dimensions.
    b. With time complexity in mind, can you derive a rule (mathematical expression) to estimate how many grid points are needed to evaluate all combination of parameters based on the number of dimensions.
    c. With this rule, compare 2-dimensional models with 4-dimensional models. 10-dimensional? 100-dimensional?
e) In experiment 3 you plotted the line of identity in the figure that compared the given response variable to the model prediction.
    a. What does this line represent and how is it useful?
    b. What does it mean for a value to lie above the line? Below the line?
    c. How would predictions that perfectly replicate the given data appear in this plot?
f) What are the weaknesses of grid search? Why wouldn't we want to use it?

I will be will looking for the following:

- That all of the tests pass
- That you used Numpy functions as much as possible and avoided writing loops where possible
- You avoided unnecessary calculations and memory copies where reasonable
- An introduction (including your own statement of the problem/lab purpose) and a written summary of your results at the top of the notebook in Markdown. Make sure to put your name at the top of the notebook.
- Plots of your data look reasonable. Plots have labels for each axis.
- Reasonable answers to the questions.

Submission Instructions

Save the notebook as a pdf file named <lastname>_lab06.pdf. Put your pdf file and cost_functions.py file into a zip file. Upload the zip file to Canvas.

Rubric

Presentation	**10%**
Followed submission and formatting instructions	5%
Plots: axes are properly labeled, used correct axes for variables, points were colored as required, lines were coloring as required, used a legend, chose appropriate axes limits to make plot readable and do not cause misleading interpretations, etc.	5%
Cost Function Coding	**25%**
Implementation passes unit tests	20%
Used Numpy functions wherever possible (no Python loops)	5%
Experiment #1	**10%**
Heatmap and plots	5%
Appropriate choice for parameters (did not maximize error)	5%
Experiment #2	**10%**
Heatmap and plots	5%
Appropriate choice for parameters (using argmin)	5%
Experiment #3	**10%**
Appropriate plots of response versus prediction (including line of identity)	5%
Appropriate choice for parameters and grid space (using argmin)	5%
Reflection Questions	**30%**
Problem 1	5%
Problem 2	5%
Problem 3	5%
Problem 4	5%
Problem 5	5%
Problem 6	5%
Exceeds Expectations	**5%**