Machine Learning & Data Mining
Set 1

Caltech CS/CNS/EE 155
January 4^{th}, 2018

Policies

- Due 9 PM, January 12^{th}, via Moodle.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- If you have trouble with this homework, it may be an indication that you should drop the class.
- You should submit all code used in the homework. We ask that you use Python 3 and sklearn version 0.18 for your code, and that you comment your code such that the TAs can follow along and run it without any issues.

Submission Instructions

Please submit your assignment as a .zip archive with filename `LastnameFirstname.zip` (replacing `Lastname` with your last name and `Firstname` with your first name), containing a PDF of your assignment writeup **in the main directory** and your code files **in a directory named `src/`**. Failure to do so will result in a **2 point deduction.**

1 Basics [16 Points]

Relevant materials: lecture 1

Answer each of the following problems with 1-2 short sentences.

Problem A [2 points]: What is a hypothesis set?

Solution A: *A hypothesis set is a set of functions (i.e. a function space) from which the hypothesis function $f(x)$ is chosen to best approximate the target function.*

Problem B [2 points]: What is the hypothesis set of a linear model?

Solution B: *The hypothesis set of a linear model is all functions of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$; in other words, this is the set of linear functions of $\mathbf{x}$ with all possible values of the weights and biases.*

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting occurs when a model has a much lower in-sample error than out-of-sample error. This usually happens when the model is fitted too closely to the in-sample error and loses robustness.*

Problem D [2 points]: What are two ways to prevent overfitting?

Solution D: *Any of the following answers are acceptable:*

- *Regularization, which adds a penalty to the model complexity as complex models are more likely to overfit*
- *Cross-validation, which monitors training error by comparing the validation error of each partition*
- *Using fewer features or more data, which prevents overfitting by increasing model stability*

Problem E [2 points]: What are training data and test data, and how are they used differently? Why should you never change your model based on information from test data?

Solution E: *Training data is the data used to generate models and is occasionally further partitioned to use for validation. Test data is the data used to evaluate a model to get an accurate representation of how the model would perform on additional data. It is important to never use information from the test data to modify the model*

to avoid the risk of contaminating the model.

Problem F [2 points]: What are the two assumptions we make about how our dataset is sampled?

Solution F: *We assume that the training data is sampled from the same distribution as the target data, and that each data point is sampled independently from the rest. We call such data "independently and identically distributed" or "i.i.d."*

Problem G [2 points]: Consider the machine learning problem of deciding whether or not an email is spam. What could X, the input space, be? What could Y, the output space, be?

Solution G: *A potential input space may be the bag-of-words representation of the emails. The output space is a binary decision of whether an email is spam.*

Problem H [2 points]: What is the k-fold cross-validation procedure?

Solution H: *The dataset is divided into k subsets, and a subset is chosen to be used as a holdout validation set with the other $k-1$ subsets used to train a model. This procedure is repeated k times, once for each subset. The training and validation errors are then averaged across the k models. This reduces the variance from having just a single holdout set, but increases the computation required.*

2 Bias-Variance Tradeoff [34 Points]

Relevant materials: lecture 1

Problem A [5 points]: Derive the bias-variance decomposition for the squared error loss function. That is, show that for a model f_S trained on a dataset S to predict a target $y(x)$ for each x,

$$\mathbb{E}_S\left[E_{\text{out}}\left(f_S\right)\right] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)]$$

given the following definitions:

$$\begin{aligned} F(x) &= \mathbb{E}_S\left[f_S(x)\right] \\ E_{\text{out}}(f_S) &= \mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right] \\ \text{Bias}(x) &= \left(F(x) - y(x)\right)^2 \\ \text{Var}(x) &= \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] \end{aligned}$$

Solution A:

$$\begin{aligned} \mathbb{E}_x\left[\text{Bias}(x) + \text{Var}(x)\right] &= \mathbb{E}_x\left[\left(F(x) - y\right)^2 + \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\left(F(x)\right)^2 - 2F(x)y + \left(y\right)^2 + \mathbb{E}_S\left[\left(f_S(x)\right)^2 - 2f_S(x)F(x) + \left(F(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(F(x)\right)^2 - 2F(x)y + \left(y\right)^2 + \left(f_S(x)\right)^2 - 2f_S(x)F(x) + \left(F(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x)\right)^2 - 2f_S y + \left(y\right)^2\right] + \left(F(x)\right)^2 - 2\left(F(x)\right)^2 + \left(F(x)\right)^2\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - y\right)^2\right]\right] \\ &= \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - y\right)^2\right]\right] \\ &= \mathbb{E}_S\left[E_{out}(f_S)\right] \end{aligned}$$

In the following problems you will explore the bias-variance tradeoff by producing learning curves for polynomial regression models.

A *learning curve* for a model is a plot showing both the training error and the cross-validation error as a function of the number of points in the training set. These plots provide valuable information regarding the bias and variance of a model and can help determine whether a model is over– or under–fitting.

Polynomial regression is a type of regression that models the target y as a degree–d polynomial function of the input x. (The modeler chooses d.) You don't need to know how it works for this problem, just know that it produces a polynomial that attempts to fit the data.

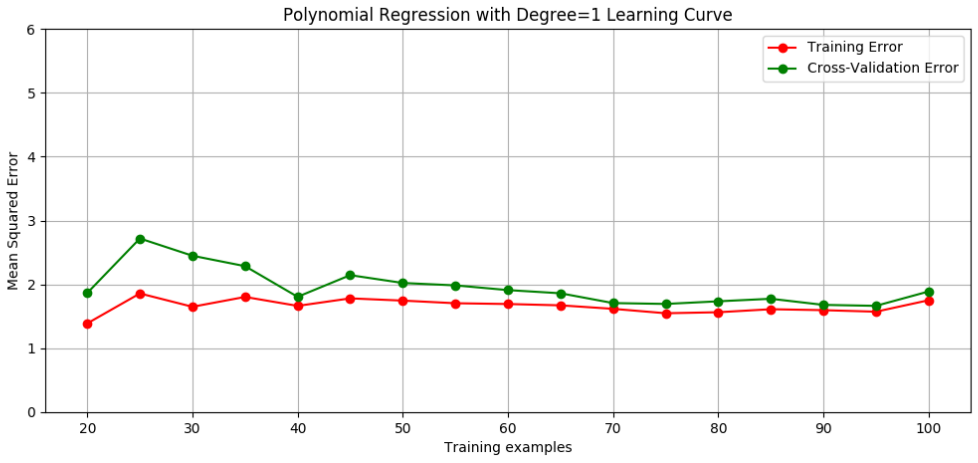
Problem B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and

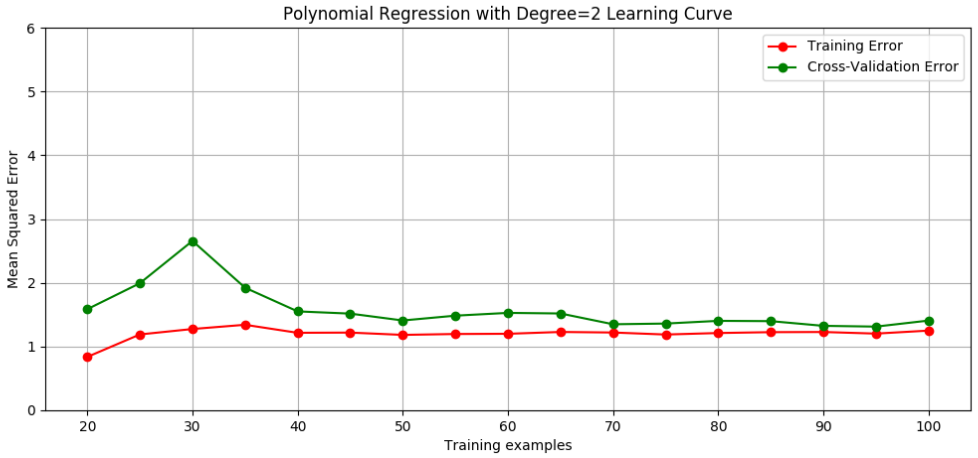
scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing with this problem. Additionally, you may find it helpful to look at the documentation for scikit-learn's learning_curve method for some guidance.

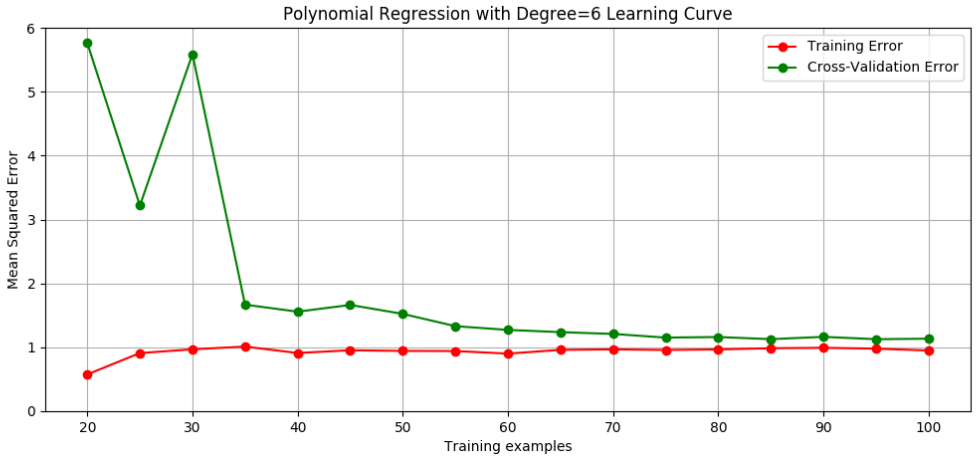
The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

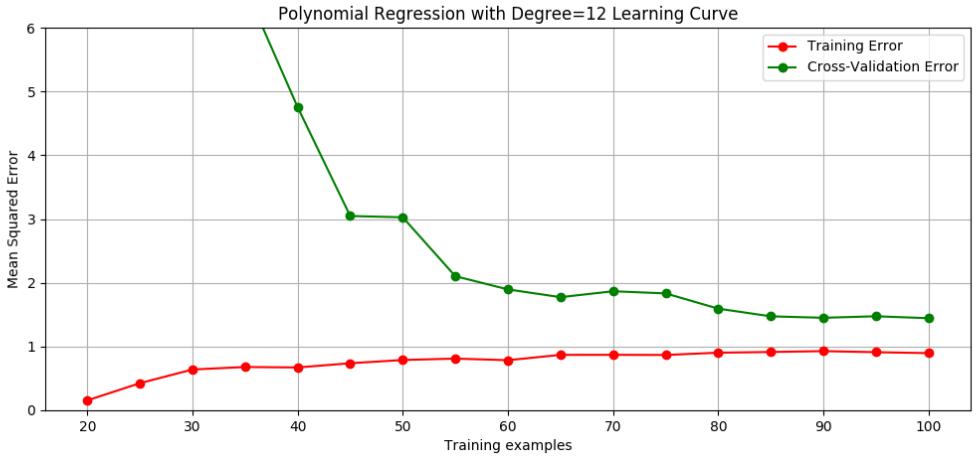
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      - Use NumPy's polyfit method to perform the degree–d polynomial regression and NumPy's polyval method to help compute the errors. (See the example code and NumPy documentation for details.)
      - When partitioning your data into folds, although in practice you should randomize your partitions, for the purposes of this set, simply divide the data into K contiguous blocks.
   ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of N.

Solution B: *See the relevant file for the solution code. The graphs are as follows:*




Polynomial Regression with Degree=2 Learning Curve
Training Error
Cross-Validation Error
Mean Squared Error
Training examples
0
1
2
3
4
5
6
20
30
40
50
60
70
80
90
100


Polynomial Regression with Degree=6 Learning Curve
Training Error
Cross-Validation Error
Mean Squared Error
Training examples
0
1
2
3
4
5
6
20
30
40
50
60
70
80
90
100



Problem C [3 points]: Based on the learning curves, which polynomial regression model (i.e. which degree polynomial) has the highest bias? How can you tell?

Solution C: *The linear model has the highest bias. The training and validation errors are both relatively high in comparison to higher degree polynomials, and the two error values have a high proximity for the same N. Both of these observations indicate that the model is underfitting.*

Problem D [3 points]: Which model has the highest variance? How can you tell?

Solution D: *The polynomial model of degree 12 has the highest variance. This is apparent due to the large discrepancy between training and validation error values for the same N, which indicates that the model is overfitting to random fluctuations in the training data.*

Problem E [3 points]: What does the learning curve of the quadratic model tell you about how much the model will improve if we had additional training points?

Solution E: *The learning curve indicates that model performance will not improve significantly with additional data. This is apparent due to the fact that the slope of the learning curve is close to 0 for large values of N.*

Problem F [3 points]: Why is training error generally lower than validation error?

Solution F: *Training error is lower than validation error since training error indicates the error of the model on the dataset it was trained on, i.e. the dataset on which the model minimized error. Validation error, on the other hand, is an evaluation of the model on data points it was not exposed to during training.*

Problem G [3 points]: Based on the learning curves, which model would you expect to perform best on some unseen data drawn from the same distribution as the training data, and why?

Solution G: *Based on the learning curves, the best model is the polynomial regression of degree 6. This model has both low bias and low variance, as the validation error is low and the gap between training and validation errors is small. As a result, the model likely neither underfits nor overfits the data. Answers selecting degree 2 will also be accepted provided the justification given is similar.*

3 The Perceptron [14 Points]

Relevant materials: lecture 2

The perceptron is a simple linear model used for binary classification. For an input vector $\mathbf{x} \in \mathbb{R}^d$, weights $\mathbf{w} \in \mathbb{R}^d$, and bias $b \in \mathbb{R}$, a perceptron $f : \mathbb{R}^d \to \{-1, 1\}$ takes the form

$$f(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right)$$

The weights and bias of a perceptron can be thought of as defining a hyperplane that divides $\mathbb{R}^d$ such that each side represents an output class. For example, for a two dimensional dataset, a perceptron could be drawn as a line that separates all points of class $+1$ from all points of class -1.

The PLA (or the Perceptron Learning Algorithm) is a simple method of training a perceptron. First, an initial guess is made for the weight vector $\mathbf{w}$. Then, one misclassified point is chosen arbitrarily and the $\mathbf{w}$ vector is updated by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y(t)\mathbf{x}(t)$$
$$b_{t+1} = b_t + y(t),$$

where $\mathbf{x}(t)$ and $y(t)$ correspond to the misclassified point selected at the t^{th} iteration. This process continues until all points are classified correctly.

The following few problems ask you to work with the provided Jupyter notebook for this problem, titled `3_notebook.ipynb`. This notebook utilizes the file `perceptron_helper.py`, but you should not need to modify this file.

Problem A [8 points]: The graph below shows an example 2D dataset. The $+$ points are in the $+1$ class and the $\circ$ point is in the -1 class.

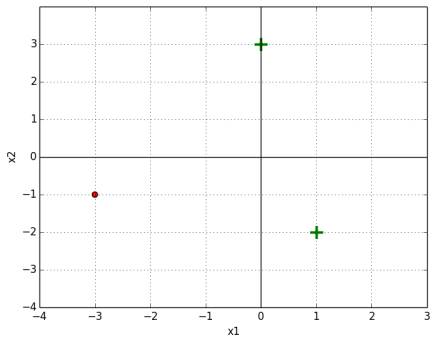


Figure 1: The green $+$ are positive and the red $\circ$ is negative

Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights $w_1 = 0, w_2 = 1, b = 0$.

Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point $([x_1, x_2], y)$ that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check your answer with the table to confirm that your perceptron code is operating correctly.

t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

Solution A: *The table is as follows:*

t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

Problem B [4 points]: A dataset $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^d \times \mathbb{R}$ is *linearly separable* if there exists a perceptron that correctly classifies all data points in the set. In other words, there exists a hyperplane that separates positive data points and negative data points.

In a 2D dataset, how many data points are in the smallest dataset that is not linearly separable, such that no three points are collinear? How about for a 3D dataset such that no four points are coplanar? Please limit your solution to a few lines - you should justify but not prove your answer.

Finally, how does this generalize for an N-dimensional set, in which **no** $<N$-dimensional hyperplane contains a non-linearly-separable subset? For the N-dimensional case, you may state your answer without proof or justification.

Solution B:

- *In 2D, a dataset must have at least 4 data points in order not to be linearly separable. For any 3 non-collinear points, a line can always be drawn that separates any 2 of the points from the 3rd point; however, the points* $\{(0,1), (0,-1)\}$ *cannot be linearly-separated from the points* $\{(-1,0), (1,0)\}$.

- *In 3 dimensions, one can always draw a plane to separate any 4 non-coplanar points (where also, no 3 points are collinear). To see this, consider the 1st 3 points, $\{x_1, x_2, x_3\}$, in the set. These points are linearly separable by a line within the plane intersecting these 3 points. One can then choose a plane which passes through this line and passes on the appropriate side of the 4th point. With 5 points, however, this may not be possible, since points $\{x_4, x_5\}$ can be chosen such that the set of planes separating $\{x_1, x_2, x_3, x_4\}$ does not overlap with the set of planes separating $\{x_1, x_2, x_3, x_5\}$.*
- *In N-dimensions, such a dataset must have at least $N + 2$ data points.*

Problem C [2 points]: Run the visualization code in the Jupyter notebook section corresponding to question C. Assume a dataset is *not* linearly separable. Will the Perceptron Learning Algorithm ever converge? Why or why not?

Solution C: *No. The PLA continues until no data points are misclassified. Since by definition there does not exist a hyperplane that can perfectly divide a non-separable dataset, the PLA algorithm will never converge.*

4 Stochastic Gradient Descent [36 Points]

Relevant materials: lecture 2

Stochastic gradient descent (SGD) is an important optimization method in machine learning, used everywhere from logistic regression to training neural networks. In this problem, you will be asked to first implement SGD for linear regression using the squared loss function. Then, you will analyze how several parameters affect the learning process.

Linear regression learns a model of the form:

$$f(x_1, x_2, \cdots, x_d) = \left(\sum_{i=1}^{d} w_i x_i \right) + b$$

Problem A [2 points]: We can make our algebra and coding simpler by writing $f(x_1, x_2, \cdots, x_d) = \mathbf{w}^T \mathbf{x}$ for vectors $\mathbf{w}$ and $\mathbf{x}$. But at first glance, this formulation seems to be missing the bias term b from the equation above. How should we define $\mathbf{x}$ and $\mathbf{w}$ such that the model includes the bias term?

Hint: *Include an additional element in* $\mathbf{w}$ *and* $\mathbf{x}$.

Solution A: *We can simply add a 1 to the* $\mathbf{x}$ *vector and include the bias term* b *in the weight vector* $\mathbf{w}$*. Thus we write*

$$\mathbf{w} = [b, w_1, w_2, \cdots, w_d], \mathbf{x} = [1, x_1, x_2, \cdots, x_d]$$

Note that the formula then evaluates to $\mathbf{w}^T \mathbf{x} = b + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$*, which is exactly how linear regression is defined.*

Linear regression learns a model by minimizing the squared loss function L, which is the sum across all training data $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ of the squared difference between actual and predicted output values:

$$L(f) = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Problem B [2 points]: SGD uses the gradient of the loss function to make incremental adjustments to the weight vector $\mathbf{w}$. Derive the gradient of the squared loss function with respect to $\mathbf{w}$ for linear regression.

Solution B: *The gradient with respect to the point* $(\mathbf{x}_i, y_i)$ *is:*

$$\begin{aligned} \nabla_{\mathbf{w}} L &= \sum_{i=1}^{N} 2(y_i - \mathbf{w}^T \mathbf{x}_i) \nabla_{\mathbf{w}} (y_i - \mathbf{w}^T \mathbf{x}_i) \\ &= \sum_{i=1}^{N} \left[-2\mathbf{x}_i (y_i - \mathbf{w}^T \mathbf{x}_i) \right] \end{aligned}$$

The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `4_notebook_part1.ipynb`, which includes tools for gradient descent visualization. This notebook utilizes the files `sgd_helper.py` and `multiopt.mp4`, but you should not need to modify either of these files. In addition, to run the animation code provided in this notebook, you may need to install FFmpeg, which includes a library for handling multimedia data. For step-by-step instructions on installing FFmpeg, please refer to the file `installing_ffmpeg.pdf`.

For your implementation of problems C-E, **do not** consider the bias term.

Problem C [8 points]: Implement the `loss`, `gradient`, and `SGD` functions, defined in the notebook, to perform SGD, using the guidelines below:

- Use a squared loss function.
- Terminate the SGD process after a specified number of epochs, where each epoch performs one SGD iteration for each point in the dataset.
- It is recommended, but not required, that you shuffle the order of the points before each epoch such that you go through the points in a random order. You can use `numpy.random.permutation`.
- Measure the loss after each epoch. Your `SGD` function should output a vector with the loss after each epoch, and a matrix of the weights after each epoch (one row per epoch). Note that the weights from all epochs are stored in order to run subsequent visualization code to illustrate SGD.

Problem D [2 points]: Run the visualization code in the notebook corresponding to problem D. How does the convergence behavior of SGD change as the starting point varies? How does this differ between datasets 1 and 2? Please answer in 2-3 sentences.

Solution D: *These different starting points converge to the same global optimum, but SGD converges more quickly when the starting point is closer to the global minimum, and converges more slowly when the starting point lies in a region of the parameter space where the loss function has a flatter slope. Different datasets result in different loss functions over which to optimize, and different loss functions converge at different rates.*

Problem E [6 points]: Run the visualization code in the notebook corresponding to problem E. One of the cells—titled "Plotting SGD Convergence"—must be filled in as follows. Perform SGD on dataset 1 for each of the learning rates $\eta \in \{$1e-6, 5e-6, 1e-5, 3e-5, 1e-4$\}$. On a single plot, show the training error vs. number of epochs trained for each of these values of η. What happens as η changes?

Solution E: *As the learning rate increases, SGD converges faster. If the learning rate is too high, then SGD oscillates rather than converging.*

The following problems consider SGD with the larger, higher-dimensional dataset, `sgd_data.csv`. The file has a header denoting which columns correspond to which values. For these problems, use the Jupyter notebook `4_notebook_part2.ipynb`.

For your implementation of problems F-H, **do** consider the bias term using your answer to problem A.

Problem F [6 points]: Use your SGD code with the given dataset, and report your final weights. Follow the guidelines below for your implementation:

- Use $\eta = e^{-15}$ as the step size.
- Use $\mathbf{w} = [0.001, 0.001, 0.001, 0.001]$ as the initial weight vector and $b = 0.001$ as the initial bias.
- Use at least 800 epochs.
- You should incorporate the bias term in your implementation of SGD and do so in the vector style of problem A.
- Note that for these problems, it is no longer necessary for the `SGD` function to store the weights after all epochs; you may change your code to only return the final weights.

Solution F: *The final weights should look similar to:*

$$\mathbf{w} = [-0.2271 \quad -5.9307 \;\; 3.9303 \quad -11.6852 \;\; 8.7492]$$

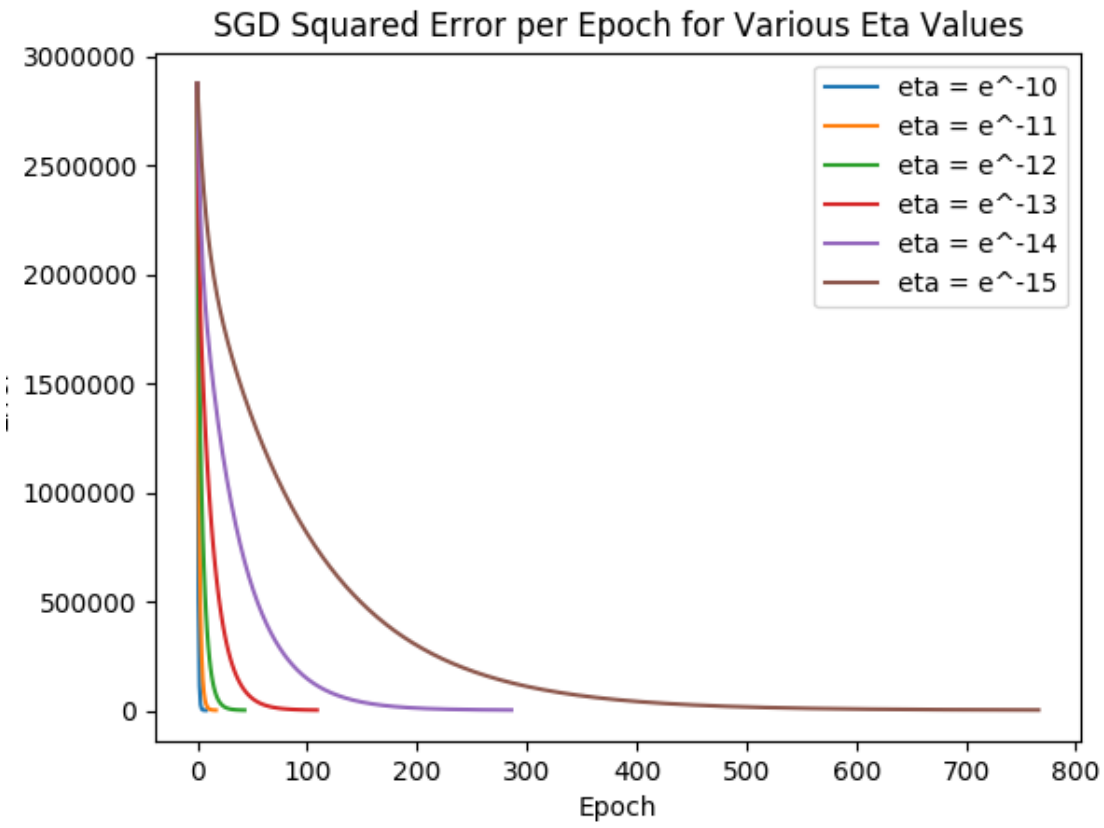
where the first element of the weight vector is w_0 as defined above.

Problem G [2 points]: Perform SGD as in the previous problem for each learning rate η in

$$\{e^{-10}, e^{-11}, e^{-12}, e^{-13}, e^{-14}, e^{-15}\},$$

and calculate the training error at the beginning of each epoch during training. On a single plot, show training error vs. number of epochs trained for each of these values of η. Explain what is happening.

Solution G: *The higher the learning rate, the faster SGD converges and the error decreases. Note that in some situations, if the learning rate is too high, SGD may oscillate around the optimal solution and never converge.*



The total squared error is not important here, as it may be scaled by any constant. What matters is the shape of the plots.

Problem H [2 points]: The closed form solution for linear regression with least squares is

$$\mathbf{w} = \left(\sum_{i=1}^{N} \mathbf{x_i} \mathbf{x_i}^T \right)^{-1} \left(\sum_{i=1}^{N} \mathbf{x_i} y_i \right).$$

Compute this analytical solution. Does the result match up with what you got from SGD?

Solution H: *The two solutions should be reasonably close. The analytical solution should be similar to:*

$$\mathbf{w} = [-0.3164 \quad -5.9916 \quad 4.0151 \quad -11.9333 \quad 8.9906 \ ]$$

using linear regression.

Answer the remaining questions in 1-2 short sentences.

Problem I [2 points]: Is there any reason to use SGD when a closed form solution exists?

Solution I: *Yes. In many cases, calculating the closed form solution is computationally intractable, in which case SGD can be used to get an arbitrarily good approximation. Also, SGD does not require operations on the entire dataset at once, and can return a valid hypothesis at any iteration.*

Problem J [2 points]: Based on the SGD convergence plots that you generated earlier, describe a stopping condition that is more sophisticated than a pre-defined number of epochs.

Solution J: *Answers may vary. For instance, one can keep track of the loss reduction from epoch to epoch, and stop when the relative loss reduction compared to the first epoch is less than some small value* ϵ*, such as* 0.0001*. That is, if* $\Delta_{0,1}$ *denotes the loss reduction from the initial model to end of the first epoch, and* $\Delta_{i-1,i}$ *denotes the loss reduction between the* $(i-1)$*th and* i*th epoch, then stop after epoch* t *if* $\Delta_{t-1,t}/\Delta_{0,1} \leq \epsilon$*.*

Problem K [2 points]: How does the convergence behavior of the weight vector differ between the perceptron and SGD algorithms?

Solution K: *With SGD, the weights converge much more smoothly than with the perceptron algorithm, and the user can control the smoothness of the algorithm's convergence by modifying the learning rate. The perceptron algorithm is relatively jumpy compared to SGD, assuming that the SGD learning rate is not overly large.*