EEP596A Advanced Intro. to Machine Learning **University of Washington**
Winter 2021 **Dept. of Electrical & Computer Engineering**

Homework 2. Due **Friday Feb 5th, 6:00pm** Electronically

Prof: J. Bilmes <bilmes@uw.edu> *Tuesday, Jan 26th 2021*
TA: Lilly Kumari <lkumari@uw.edu>

General Instructions.

This homework consists of two parts, a write-up part that needs to be turned in using a single pdf file, and also a programming part that needs to be turned in using a zipped Jupyter notebook file (or files). A Jupyter notebook file has extenion `.ipynb` and a zip file has extension `.zip` and should contain one or more `.ipynb` files.

Doing your homework by hand and then converting to a PDF file (by say taking high quality photos using a digital camera and then converting that to a PDF file) is fine, as there are many jpg to pdf converters on the web. Alternatively, you are welcome to use Latex (great for math and equations), Microsoft Word, and Google Docs, or hand-written paper, as long as the final submitted format is a single pdf.

For the plots requested in the programming session, you can either save them as pictures and insert them manually into the writeup, or directly export the completed jupyter notebook to a pdf file (in jupyter notebook, "File→Download as→PDF via LaTex") and copy it in to your writeup.

Some of the problems below might require that you look at some of the lecture slides at our web page (https://canvas.uw.edu/courses/1431528).

Note that the due dates and times are often in the evenings.

As mentioned above, for the programming problems, you need to submit your code (written in python as a Jupyter notebook) and the answers to the non-coding questions should also be included in the pdf write-up. Your code answers must to be in python, no other language is accepted.

Neatness and clarity count! : Answers to your questions must be clearly indicated in all cases. Not only correctness, but clarity and completeness is necessary to receive full credit. Justify you answers. A correct answer does not guarantee full credit and a wrong answer does not guarantee poor credit, hence show all work and justify each step, thinking "clarity" and "neatness" along the way. If we can't understand your answer, or if your answers are not well and neatly organized, you will not receive full credit.

All homework is due electronically via the link https://canvas.uw.edu/courses/1431528/assignments. This means that on canvas you turn in two files: (1) **a pdf file** with answers to the writeup questions, and (2) **a zip file with python code (in jupyter notebook files)**. **Please do not submit any fewer or any more than these two files.**

Problem 1. Ridge Regression [45 points]

Recall that linear regression solves

$$\min_w \|Xw - y\|_2^2, \tag{1}$$

Where X is the $n \times m$ "design matrix", where every row of X corresponds to an m-dimensional data point, y refers to the length-n vector of labels, and w is the weight vector we aim to learn using mathematical optimization. In other words, X is an $n \times m$ data matrix with n data samples (rows) and m features (columns), and y is an n-dimensional column vector of labels, one for each sample.

Ridge regression is very similar, and is defined as

$$\min_w \|Xw - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2, \tag{2}$$

where we add an additional regularization term $\frac{\eta}{2}\|w\|_2^2$ to encourage the weights to be small and has other benefits as well which we discuss in class.

Please make sure you answer every question clearly and completely.

Problem 1(a). [2 points] Describe (with drawings and an intuitive description) one setting for (X, y), where standard linear regression is preferred over ridge regression. The drawing should show: (1) the data points (X, y); (2) the expected linear regression solution (e.g., a line); (3) expected ridge regression solution (also, e.g., a line). You need to clearly explain the reason for why standard linear regression is preferred over ridge regression. You need not do any actual calculation here.

One setting that linear regression is preferred over ridge regression is that the data lies perfectly on a straight line (see Fig 1). Since the data is already on a line, linear regression will properly compute the slope and intercept, but ridge regression will "shrink" the coefficients leading to a line that does not fit the line defined by the data.

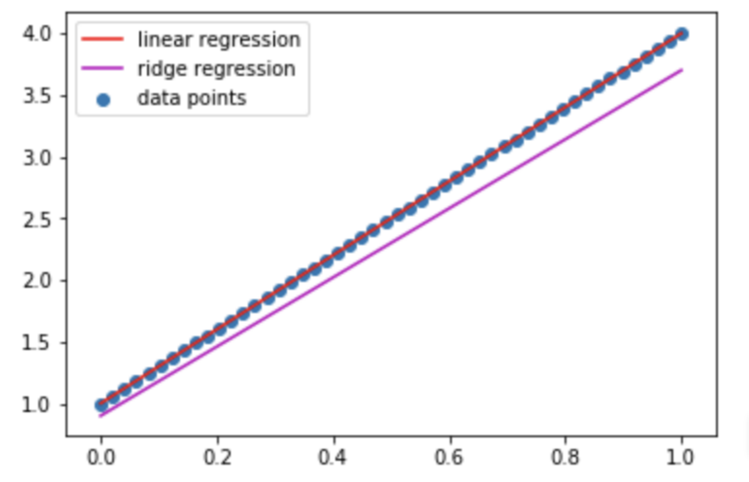

Figure 1: Problem 1 (a)

Problem 1(b). [3 points] Describe (with drawings and intuitive description) one setting for (X, y), where ridge regression is preferable to linear regression. Your answer should fulfill the same requirements in part 1(a).

One setting that ridge regression is preferred over linear regression is that there are some noisy data points above the straight line(see Fig 2). Note this is not the only setting as there are many cases where ridge regression is preferred over linear regression, i.e., whenever the data is a bit noisy.

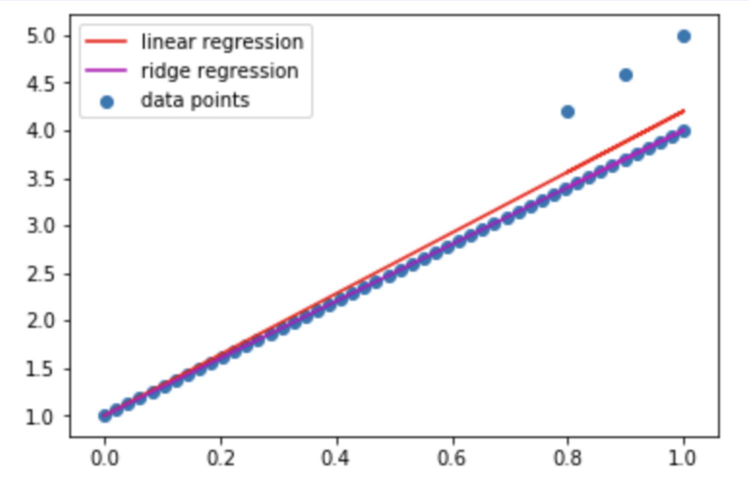

Figure 2: Problem 1 (b)

Problem 1(c). [5 points] What's the effect of the regularization hyper-parameter η on the optimization solution in terms of bias and variance? I.e., how will the bias and variance change if you increase η and vice versa?

An increase of η increases bias and decreases variance. A decrease of η decreases the bias and increases the variance. As we add more weight to regularization, the solution pays less attention to the loss term and more on the regularization term, and therefore the overall loss on the data get worse even if the overall objective (loss plus regularizer)is minimized. However, as we decrease the weight of the regularization, the loss becomes more important, the bias decreases, the model tends to fit the noise better, and the variance therefore increases.

Problem 1(d). [15 points] Solve for the closed form solution for ridge regression. To get the closed-form solution, you can set the gradient of the objective function $F(w)$ in the above minimization problem to be zero, i.e., $\frac{\partial F(w)}{\partial w} = 0$, and solve this equation of w. If you are not familiar about how to compute the gradient (or such derivatives), please refer to Section 2.4 of the Matrix Cookbook (https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf).

$$F(w) = (Xw - y)^T(Xw - y) + \frac{\eta}{2} w^T w \tag{3}$$

$$\frac{\partial F(w)}{\partial w} = 2X^T(Xw - y) + \eta w \tag{4}$$

$$\tag{5}$$

Setting it to 0, we get:

$$X^T y = X^T X w + \frac{\eta}{2} w \tag{6}$$

$$\tag{7}$$

Note $X^T Xw$ has the same dimension as w. We can factor out w with an identity matrix:

$$X^T y = (X^T X + \frac{\eta}{2} I)w \tag{8}$$

$$w = (X^T X + \frac{\eta}{2} I)^{-1} X^T y \tag{9}$$

Problem 1(e). [5 points] Normally $m \ll n$ and the features are fairly independent. Consider now, two special cases that sometimes come up in practice and one should be aware of: (1) where the columns (or features) of X are more than the rows (or samples) meaning where $m > n$; and (2) where the columns (or features) of X are highly correlated (an extreme case is where many features are identical to each other). For each of the above:

1. Can you still compute the closed-form solution of the vanilla linear regression? If so, show how, otherwise show why not.
2. Assuming you found a closed form solution for vanilla linear regression above, compare it to the solution for ridge regression. Do you discover other benefits of ridge regression?

If either the columns are highly correlated (i.e. when the X matrix is not of full rank), or if $m > n$, we end up in a situation where $X^T X$ is not full rank. In this case, we cannot compute the inverse of $X^T X$ as is required in the linear regression solution. However, the ridge regression solution adds a multiple of the identity matrix to $X^T X$ in the solution, which makes the inversion possible. This is another benefit of ridge regression.

Problem 1(f). [15 points] In the above formulation of ridge regression we regularized all of the learnt parameters, but in many cases we do not wish to do that. Recall that when we want to have an offset/intercept/bias weight, we can just set one of the features to be identically to 1 (this is identical, say, to the right-most column of the design matrix X being set to a column vector of all ones). In the following configuration of ridge regression, we do not regularize the offset/intercept/bias term and let it be free:

$$\min_{w,w_0} \|Xw + w_0 - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2$$

Problem 1(f)-i. Derive a closed form solution for this regression problem by first setting the gradient of the objective function with respect to w_0 to zero, i.e., $\frac{\partial F(w,w_0)}{\partial w_0} = 0$ to obtain w_0 and then substitute the value of w_0 in $\frac{\partial F(w,w_0)}{\partial w} = 0$ to obtain w.

$$F(w, w_0) = \sum_{i=1}^{n} (x^{(i)\top} w + w_0 - y^{(i)})^2 + \frac{\eta}{2}\|w\|_2^2 \tag{10}$$

$$\frac{\partial F(w, w_0)}{\partial w_0} = 2\sum_{i=1}^{n} (x^{(i)\top} w + w_0 - y^{(i)}) \tag{11}$$

Setting it to 0, we get:

$$w_0 = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - x^{(i)\top} w) \tag{12}$$

$$= \overline{y} - \overline{x}^\top w \tag{13}$$

where $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}$ and $\overline{x}^\top = \frac{1}{n}\sum_{i=1}^{n} x^{(i)\top}$

Substituting the value of w_0 in $F(w, w_0)$, we get:

$$\begin{aligned} F(w) &= \|Xw + \vec{1}\overline{y} - \vec{1}\overline{x}^\top w - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2 && (14)\\ &= \|(X - \vec{1}\overline{x}^\top)w - (y - \vec{1}\overline{y})\|_2^2 + \frac{\eta}{2}\|w\|_2^2 && (15)\\ &= \|\widetilde{X}w - \widetilde{y}\|_2^2 + \frac{\eta}{2}\|w\|_2^2 && (16) \end{aligned}$$

where $\widetilde{X} = X - \vec{1}\overline{x}^\top$ and $\widetilde{y} = y - \vec{1}\overline{y}$

Taking its derivative with respect to w as done in Problem 1(d), we get a similar closed form solution to this problem such that:

$$w = (\widetilde{X}^\top\widetilde{X} + \frac{\eta}{2}I)^{-1}\widetilde{X}^\top\widetilde{y} \qquad (17)$$

Problem 1(f)-ii. Briefly explain the benefits of not penalizing the offset parameter.

The offset/intercept/bias parameter adjusts to the magnitude of the data. If we penalize the bias term as well, then adding a constant c to each of the targets $y^{(i)}$ would not simply result in a shift of their predictions by the same amount c. Additionally, a large offset can also provide a good fit for the data with the remaining parameters (w) being small.

Problem 2. Bias and Variance [Extra Credit : 20 points]

For a Gaussian noise Linear Least Squares regression model, we have $\vec{y} = X\theta + \vec{\epsilon}$ where X denotes the $n \times m$ design matrix, and $\vec{\epsilon}$ denotes a length-n vector of Gaussians, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Using the MLE parameter estimate $\tilde{\theta} = (X^\top X)^{-1}X^\top\vec{y}$, prove the following two things.

Problem 2(a). 10 points Show that $E_{\mathcal{D}}[h_{\mathcal{D}}(x)] = E[Y \mid x]$ i.e., the least squares regression model is unbiased.

$$\begin{aligned} E_{\mathcal{D}}[h_{\mathcal{D}}(x)] &= E_{\mathcal{D}}[x^\top\tilde{\theta}] && (18)\\ &= E_{\mathcal{D}}[x^\top(X^\top X)^{-1}X^\top\vec{y}] && (19)\\ &= E_{\mathcal{D}}[x^\top(X^\top X)^{-1}X^\top(X\theta + \vec{\epsilon})] && (20)\\ &= E_{\mathcal{D}}[x^\top\theta + x^\top(X^\top X)^{-1}X^\top\vec{\epsilon}] && (21)\\ &= x^\top\theta + x^\top(X^\top X)^{-1}X^\top E_{\mathcal{D}}[\vec{\epsilon}] && (22)\\ &= x^\top\theta \qquad\qquad (\text{since } E_{\mathcal{D}}[\vec{\epsilon}] = 0) && (23)\\ &= E[Y \mid x] && (24) \end{aligned}$$

Problem 2(b). 10 points Show that $E_{\mathcal{D}}[(h_{\mathcal{D}}(x) - E_{\mathcal{D}}[h_{\mathcal{D}}(x)])^2] = \sigma^2 x^\top(X^\top X)^{-1}x$

$$
\begin{aligned}
E_{\mathfrak{D}}[(h_{\mathfrak{D}}(x) - E_{\mathfrak{D}}[h_{\mathfrak{D}}(x)])^2] &= E_{\mathfrak{D}}[(x^\top \tilde{\theta} - x^\top \theta)^2] && (25)\\
&= E_{\mathfrak{D}}[(x^\top (X^\top X)^{-1} X^\top \vec{y} - x^\top \theta)^2] && (26)\\
&= E_{\mathfrak{D}}[(x^\top (X^\top X)^{-1} X^\top (X\theta + \vec{\epsilon}) - x^\top \theta)^2] && (27)\\
&= E_{\mathfrak{D}}[(x^\top \theta + x^\top (X^\top X)^{-1} X^\top \vec{\epsilon} - x^\top \theta)^2] && (28)\\
&= E_{\mathfrak{D}}[(x^\top (X^\top X)^{-1} X^\top \vec{\epsilon})^2] && (29)
\end{aligned}
$$

Using the fact that for a scalar c, $c^2 = cc^\top$, we get:

$$
\begin{aligned}
E_{\mathfrak{D}}[(h_{\mathfrak{D}}(x) - E_{\mathfrak{D}}[h_{\mathfrak{D}}(x)])^2] &= E_{\mathfrak{D}}[(x^\top (X^\top X)^{-1} X^\top \vec{\epsilon})(x^\top (X^\top X)^{-1} X^\top \vec{\epsilon})^\top] && (30)\\
&= x^\top (X^\top X)^{-1} X^\top E_{\mathfrak{D}}[\vec{\epsilon}\vec{\epsilon}^\top](x^\top (X^\top X)^{-1} X^\top)^\top && (31)\\
&= x^\top (X^\top X)^{-1} X^\top \sigma^2 I (x^\top (X^\top X)^{-1} X^\top)^\top && (32)\\
&= \sigma^2 x^\top (X^\top X)^{-1} X^\top X (X^\top X)^{-1} x && (33)\\
&= \sigma^2 x^\top (X^\top X)^{-1} x && (34)
\end{aligned}
$$

Problem 3. Programming Problem: Linear Regression [30 points]

Before you start: if you have not yet done so, please install anaconda python (python 3.x version is recommended) by following the instructions at `https://www.anaconda.com/download/`, and then install scikit-learn, numpy, matplotlib, seaborn, pandas and jupyter notebook in anaconda, for example, by running command "conda install seaborn". Note some of the above packages may have already been installed in anaconda, depending on which version of anaconda you just installed.

In this problem, you will implement the closed-form solvers of linear regression and ridge regression from scratch (which means that you cannot use built-in linear/ridge regression modules in scikit-learn or any other packages). Then you will try your implementation on a small dataset, the Boston housing price dataset, to predict the house prices in Boston ("MEDV") based on some related feature attributes.

We provide an ipython notebook "`linear_regression_boston.ipynb`" for you to complete. In your terminal, please go to the directory where this file is located, and run command "jupyter notebook". A local webpage will be automatically opened in your web browser, click the above file to open the notebook. You need to complete the scripts below the "TODOs" (please search for every "TODO"), and submit the completed ipynb file (inside your .zip file). In your writeup, you also need to include the plots and answers to the questions required in this session.

The first part of this notebook serves as a quick tutorial of loading dataset, using pandas to get summary and statistics of dataset, using seaborn and matplotlib for visualization, and some commonly used functionalities of scikit-learn. You can explore more functionalities of these tools by yourself. You will use these tools in future homeworks.

Problem 3(a). [5 points] Below "2.1 how does each feature relate to the price" in the ipynb file, we show a 2D scatter plot for each feature, where each point associates with a sample, and the two coordinates are the feature value and the house price of the sample. Please find the three top features that are most correlated (i.e., linearly related) to the house price ("MEDV").

Problem 3(b). [5 points] Below "2.2 correlation matrix", we compute the correlation matrix by pandas, and visualize the matrix using a heatmap of seaborn. Please find the three top features that are most correlated to the house price ("MEDV") according to the correlation matrix. Are they the same as the ones in in problem 3(a)?

Problem 3(c). [10 points] Below "2.3 linear regression and ridge regression", please implement the closed-form solver of linear regression and ridge regression (linear regression with L2 regularization). You are only allowed to use numpy here (but you can use existing solutions to debug that your code is correct). Recap: linear regression solves

$$\min_w F(w) = \|Xw - y\|_2^2, \tag{35}$$

while ridge regression solves

$$\min_w F(w) = \|Xw - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2, \tag{36}$$

where X is an $n \times m$ data design matrix with n data samples and m features, and y is an n-dim vector storing the prices of the n data samples, and $F(w)$ is the objective function. Run the linear regression and ridge regression on the randomly train-test split training set, and report the obtained coefficients w. For ridge regression, it is recommend to try different η values.

Problem 3(d). [5 points] Below "2.4 evaluation", implement prediction function and root mean square error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{37}$$

where $\hat{y}_i$ is the predicted price and y_i is the true price of sample i. Apply the implementation and report the RMSE of linear regression and ridge regression on training set and test set. Compare the training RMSE of linear regression and ridge regression, what do you find? How about the comparison of their test RMSE? Can you explain the difference?

Problem 3(e). [5 points] Below "2.5 linear models of top-3 features", train a linear regression model and a ridge regression model by using the top-3 features you achieved in 3.2, and then report the RMSE on training set and test set. Compare the RMSE of using all the 13 features: what is the difference? what does this indicate?

Problem 3(f). [Extra credit : 20 points] Now try some feature engineering (e.g., add combinations of features such as multiplication/division of two features, square root of one feature, and etc.) and regularization techniques, (for example, L2 regularization) and get the test set RMSE as low as you can. Also note that when you try different techniques, you should randomly select 20% of the **training set** as the **validation set** (also often named as the **development set**), and keep the rest 80% as the **training set**. You should tune your techniques purely based on the RMSE of the validation set to avoid overfitting. Keep in mind that the improved RMSE on the validation set may not result in consistent improvements on the test set, but do experiment to get experience yourself with this phenomenon and report back to us everything you find and why. Also report the techniques you use, and your improved test set RMSE.

(a) LSTAT, RM, and PTRATIO.

(b) LSTAT, RM, and PTRATIO.

(c) Please see the solution Jupyter notebook.

(d) Ridge regression has larger training RMSE, but smaller test RMSE. L2 regularization in ridge regression can improve generalization performance on test set and avoid overfitting via reducing the variance (but introduces extra bias that increases the training error).

(e) The RMSE of using only top-3 features is slightly worse but still close to the RMSE of using all the 13 features. This indicates the importance of feature selection.

Problem 4. Programming Problem: Logistic Regression [40 points]

In this problem, you will implement the gradient descent algorithm and mini-batch stochastic gradient descent algorithm for multi-class logistic regression from scratch (which means that you cannot use built-in logistic regression modules in scikit-learn or any other packages). Then you will be asked to try your implementation on a hand-written digits dataset to recognize the hand-written digits in the given images.

We provide an ipython notebook "`logistic_regression_digits.ipynb`" for you to complete. You need to complete the scripts below "TODO" (please search for every "TODO"), and submit the completed ipynb file. In your writeup, you also need to include the plots and answers to the questions required in this session.

In class, you learned logistic regression for binary classification and softmax regression as its generalization. In this problem, you will implement the more general softmax form of logistic regression model for multi-class classification (i.e., not just binary classification, but can allow multiple classes). Given features of a sample x, a multi-class logistic regression produces class probability (i.e., the probability of the sample belonging to class k)

$$\Pr(y = k|x; W, b) = \frac{\exp(xW_k + b_k)}{\sum_{j=1}^{c} \exp(xW_j + b_j)} = \frac{\exp(z_k)}{\sum_{j=1}^{c} \exp(z_j)}, \ \forall\, k = 1, 2, \cdots, c \tag{38}$$

where c is the number of possible classes, model parameter W is a $m \times c$ matrix and b is a c-dimensional vector. We call the c values $z_k = xW_k + b_k$ for each $k \in \{1, 2, \ldots, c\}$ the c logits associated with the c classes. A binary logistic regression model ($c = 2$) is a special case of the above multi-class logistic regression model (you can figure out why by yourself with a few derivation, and as we did in class). The predicted class $\hat{y}$ of x is

$$\hat{y} = \operatorname*{argmax}_{k=1,2,\cdots,c} \Pr(y = k|x; W, b). \tag{39}$$

For simplicity of implementation, we can extend each x by adding an extra dimension (and feature) with fixed constant value 1, i.e., $x \leftarrow [x, 1]$, and accordingly add an extra row to W, i.e, $W \leftarrow [W; b]$. Thus, we get a bias shift implemented very easily this way.

After using the extended representation of x and W, the logits z take the form $z_k = xW_k$. Logistic regression solves the following optimization for maximum likelihood estimation.

$$\min_{W} F(W) \text{ where } F(W) = \frac{1}{n} \sum_{i=1}^{n} -\log[\Pr(y = y_i|x = X_i; W)] + \frac{\eta}{2} \|W\|_F^2, \tag{40}$$

where we use a regularization similar to the ℓ_2-norm in ridge regression, i.e., the Frobenius norm $\|W\|_F^2 = \sum_{j=1}^{c} \|W_{\cdot,j}\|_2^2 = \sum_{j=1}^{c} \sum_{i=1}^{m} (W_{i,j})^2$. Note that $W_{\cdot,j}$ is the j^{th} column of W.

Problem 4(a). [5 points] Derive the gradient of $F(W)$ w.r.t. W, i.e., $\frac{\partial F(W)}{\partial W}$, and write down the gradient descent rule for W. Compare it with the LMS (least mean square) update rule in class. Where are they similar to each other?

Problem 4(b). [15 points] Below "3.2 batch gradient descent (GD) for Logistic regression", implement the batch gradient descent algorithm with constant learning rate. To avoid numerical problems when computing the exponential in the probability $\Pr(y = k|x; W, b)$, you can use a modification of the logits z', i.e.,

$$z' = z - \max_{j} z_j. \tag{41}$$

When the change of objective $F(W)$ comparing to $F(W)$ in the previous iteration is less than $\epsilon = 1.0e-4$, i.e., $|F_t(W) - F_{t-1}(W)| \leq \epsilon$, stop the algorithm. Please record the value of $F(W)$ after each iteration of gradient descent.

Please run the implemented algorithm to train a logistic regression model on the randomly split training set. We recommend to use $\eta = 0.1$. Try three different learning rates $[5.0e-3, 1.0e-2, 5.0e-2]$, report the final value of $F(W)$ and training/test accuracy in these three cases, and draw the three convergence curves (i.e., $F_t(W)$ vs. iteration t) in a 2D plot.

Problem 4(c). [5 points] Compare the convergence curves: what are the advantages and disadvantages of large and small learning rates?

Problem 4(d). [10 points] Below "3.3 stochastic gradient descent (SGD) for Logistic regression", implement the mini-batch stochastic gradient descent (SGD) for logistic regression. You can reuse some code from the previous gradient descent implementation.

Tuning hyper-parameters is critical to get good models. For the mini-batch SGD algorithm, the hyper-parameters consist of 1) the initial learning rate, 2) the learning rate schedule, or how do we change the learning rate over time, 3) the number of epochs[1] to train, and 4) the mini-batch size.

We suggest using the following automatic tuning strategy for the learning rate schedule and number of epochs. For the learning rate schedule, record the objective $F(W)$ over epochs, and if the objective has not become better than 10 epochs ago, reduce the learning rate by a factor of two. For the number of epochs, or stopping criteria, continue training until $F(W)$ has not improved over 20 epochs.

You can start by using an initial learning rate of $1.0e-2$ and a mini-batch size of 100 for this problem. You can discard the last mini-batch of every epoch if it is not full. Please remember to record the value of $F(W)$ after each epoch and the final training and test accuracy.

Run your code for different mini-batch sizes: [10, 50, 100]. Report the final value of $F(W)$ and final training/test accuracy, and draw the three convergence curves ($F_t(W)$ vs. epoch t) in a 2D plot.

Problem 4(e). [5 points] Compare the convergence curves: do they (logistic regression with the three different batch sizes) show the same convergence speed, when the same initial learning rate is used? For different batch sizes, you may need to tune the initial learning rate. In general, the rule of thumb is to scale the learning rate linearly with the batch size. Please draw the new convergence curves after tuning the learning rate in a 2D plot. What is the difference as compared to the old convergence curves? Can you give some mathematical explanations based on the SGD you implemented? Also, what learning rate yielded the overall fastest convergence in terms of wall clock time?

(a) We will start by computing the gradient of $-\log[\Pr(y = y_i | x = X_i; W)]$. For simplicity, w.l.o.g., let $y_i = k$ and $x = X_i$, by the definition of softmax,

$$-\log[\Pr(y = k | x; W)] = \log \sum_{j=1}^{c} \exp(z_j) - z_k. \tag{42}$$

Then we compute the gradients of the two terms in the right hand side above separately. For the first

[1] An **epoch** corresponds to training the model with every data point once, so each epoch is one complete **pass** over the training data set. In most theoretical analyses, stochastic gradient descent (SGD) samples data points with replacement. In practice, however, a widely adopted approach is to random shuffle the dataset once, and iterate over this random order. Also, we typically form mini-batches according to this model. We then let the model learn by doing an epoch over all data points once according to this order. This process is repeated until convergence.

term, by chain rule, we have

$$\begin{aligned}\frac{\partial \log \sum_{j=1}^{c} \exp(z_j)}{\partial W_j} &= \frac{1}{\sum_{j=1}^{c} \exp(z_j)} \times \frac{\partial \sum_{j=1}^{c} \exp(z_j)}{\partial W_j} && (43)\\ &= \frac{1}{\sum_{j=1}^{c} \exp(z_j)} \times \frac{\partial \exp(z_j)}{\partial W_j} && (44)\\ &= \frac{1}{\sum_{j=1}^{c} \exp(z_j)} \times \frac{\partial \exp(z_j)}{\partial z_j} \times \frac{\partial z_j}{\partial W_j} && (45)\\ &= \frac{1}{\sum_{j=1}^{c} \exp(z_j)} \times \exp(z_j) \times x && (46)\\ &= \frac{\exp(z_j)}{\sum_{j=1}^{c} \exp(z_j)} \times x && (47)\\ &= \Pr(y = j|x; W) \times x, \quad \forall j = 1, \ldots, c. && (48)\end{aligned}$$

For the second term, it is trivial to have

$$\frac{\partial z_k}{\partial W_j} = \begin{cases} x, & j = k \\ 0, & j \neq k \end{cases} \tag{49}$$

Note here that since both of these terms are gradients w.r.t. W_j, they are of dimension $m \times 1$.

Combing the above results yields

$$\frac{\partial - \log[\Pr(y = k|x; W)]}{\partial W_j} = \begin{cases} [\Pr(y = j|x; W) - 1] \times x, & j = k \\ \Pr(y = j|x; W) \times x, & j \neq k \end{cases} \tag{50}$$

Therefore, define $n \times c$ classification probability matrix P such that $P_{i,j} = \Pr(y = j|x = X_i; W)$, and an $n \times c$ ground truth label matrix Y such that $Y_{i,j} = 1$ if $y_i = j$ otherwise $Y_{i,j} = 0$. Stacking together all classes j, we can express the gradient of logistic regression which is an $m \times c$ matrix as

$$\frac{\partial F(W)}{\partial W} = \frac{1}{n} \times X^T (P - Y) + \eta W. \tag{51}$$

(b) Please see the solution Jupyter notebook.
(Note that in these solutions, the loss $L(W)$ is plotted at each iteration, as opposed to $F(W)$ as asked in the problem. We give credit to plotting either of these.)

(c) Training with large learning rate converges faster but is unstable especially in early iterations, while training with small learning rate converges more smoothly but is slower.

(d) Please see the solution Jupyter notebook.

(e) When using the same initial learning rate, training with smaller batch size converges faster.