Homework #2

CSE 446/546: Machine Learning
Joseph David, with Kevin Liu
Due: **Wednesday** May 5, 2021 11:59pm

Please review all homework guidance posted on the website before submitting to Gradescope. Reminders:

- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to *[5 points]* . For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in boxed text, are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see website for details). In Gradescope there is a place to submit solutions to A and B problems separately. You are welcome to create just a single PDF that contains answers to both, submit the same PDF twice, but associate the answers with the individual questions in Gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- For every problem involving code, please include the code as part of your PDF for the PDF submission *in addition to* submitting your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of *[1 points]* .

Not adhering to these reminders may result in point deductions.

Conceptual Questions *[10 points]*

A0. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* Suppose that your estimated model for predicting house prices has a large positive weight on the feature `number of bathrooms`. If we remove this feature and refit the model, will the new model have a strictly higher error than before? Why?

b. *[2 points]* Compared to L2 norm penalty, explain why a L1 norm penalty is more likely to result in sparsity (a larger number of 0s) in the weight vector.

c. *[2 points]* In at most one sentence each, state one possible upside and one possible downside of using the following regularizer: $\left(\sum_i |w_i|^{0.5}\right)$

d. *[1 points]* True or False: If the step-size for gradient descent is too large, it may not converge.

e. *[2 points]* In your own words, describe why stochastic gradient descent (SGD) works, even though only a small portion of the data is considered at each update.

f. *[2 points]* In at most one sentence each, state one possible advantage of SGD over GD (gradient descent), and one possible disadvantage of SGD relative to GD.

Solution:

a. Not necessarily. The error may stay the same or decrease.
b. The levels sets of the L1 norm have corners at sparse points and so the minimum of the objective function tends to intersect at the corners.
c. One upside is that this regularizer does not penalize large weights which may be useful in certain contexts. One downside is that it is not convex and may therefore be difficult to optimize.
d. True, we proved in class an upper bound for the step size, after which gradient descent may not converge.
e. Stochastic gradient descent works because we sample our data points uniformly at random so, in expectation, we get the correct gradient.
f. One possible advantage of SGD over GD is that it is much quicker to compute because we only need to take a small fraction of the data points into account. One downside is that SGD may take longer to converge because the steps are not always in the optimal direction due to the aspect of randomness in SGD.

Convexity and Norms

A1. A *norm* $\|\cdot\|$ over $\mathbb{R}^n$ is defined by the properties: (i) non-negativity: $\|x\| \geq 0$ for all $x \in \mathbb{R}^n$ with equality if and only if $x = 0$, (ii) absolute scalability: $\|a\,x\| = |a|\,\|x\|$ for all $a \in \mathbb{R}$ and $x \in \mathbb{R}^n$, (iii) triangle inequality: $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$.

a. *[3 points]* Show that $f(x) = (\sum_{i=1}^n |x_i|)$ is a norm. (Hint: for (iii), begin by showing that $|a+b| \leq |a|+|b|$ for all $a, b \in \mathbb{R}$.)

b. *[2 points]* Show that $g(x) = \left(\sum_{i=1}^n |x_i|^{1/2}\right)^2$ is not a norm. (Hint: it suffices to find two points in $n = 2$ dimensions such that the triangle inequality does not hold.)

Context: norms are often used in regularization to encourage specific behaviors of solutions. If we define $\|x\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$ then one can show that $\|x\|_p$ is a norm for all $p \geq 1$. The important cases of $p = 2$ and $p = 1$ correspond to the penalty for ridge regression and the lasso, respectively.

Solution:
a. Let $x \in \mathbb{R}^n$.

(i) Since $|x_i| \geq 0$ for all $i = 1, \ldots, n$, it follows that $f(x) = \sum_{i=1}^n |x_i| \geq 0$.

(ii) Let $a \in \mathbb{R}$. Then, $f(ax) = \sum_{i=1}^n |ax_i| = \sum_{i=1}^n |a| \cdot |x_i| = |a| \sum_{i=1}^n |x_i| = |a| f(x)$.

(iii) We begin by following the hint and showing that $|a + b| \leq |a| + |b|$ for all $a, b \in \mathbb{R}$. Note that by definition, $|x| = \max\{x, -x\}$ and therefore $x \leq |x|$ for all $x \in \mathbb{R}$. Therefore,

$$a + b \leq |a| + b \leq |a| + |b| \text{ and } -a - b \leq |a| - b \leq |a| + |b|.$$

Thus, $|a + b| = \max\{a + b, -a - b\} \leq |a| + |b|$. Using this fact, we have that if $x, y \in \mathbb{R}^n$ then

$$f(x + y) = \sum_{i=1}^n |x_i + y_i| \leq \sum_{i=1}^n |x_i| + |y_i| = f(x) + f(y),$$

which proves the triangle inequality for f. Since f satisfies conditions $(i), (ii)$, and (iii), f is a norm.

b. Using the case $n = 2$ with points $x = (0, 1)$ and $y = (1, 0)$, we have that

$$g(x + y) = g((1, 1)) = (\sqrt{1} + \sqrt{1})^2 = 2^2 = 4,$$

whereas

$$g(x) + g(y) = g((0, 1)) + g((1, 0)) = 1 + 1 = 2.$$

Thus, $g(x + y) > g(x) + g(y)$ so the triangle inequality does not hold. Therefore , $g(x)$ does not define a norm.

B1. *[6 points]* For any $x \in \mathbb{R}^n$, define the following norms: $\|x\|_1 = \sum_{i=1}^n |x_i|$, $\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$, $\|x\|_\infty := \lim_{p\to\infty} \|x\|_p = \max_{i=1,\ldots,n} |x_i|$. Show that $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$.

Solution: Let $a, b \geq 0$. Then, this implies that $ab \geq 0$ and hence

$$(a+b)^2 = a^2 + 2ab + b^2 \geq a^2 + b^2.$$

Suppose for induction that for any $a_1, \ldots, a_{n-1} \geq 0$ we have that $(a_1 + \cdots + a_{n-1})^2 \geq a_1^2 + \cdots + a_n^2$. Then, if $a_n \geq 0$, then by our base case we have that

$$(a_1 + \cdots + a_{n-1} + a_n)^2 \geq (a_1 + \cdots + a_{n-1})^2 + a_n^2 \geq a_1^2 + \cdots + a_{n-1}^2 + a_n^2.$$

By induction, we have shown for any $a_1, \ldots, a_n \geq 0$ that $(a_1 + \cdots + a_n)^2 \geq a_1^2 + \cdots + a_n^2$. Thus, for any $x \in \mathbb{R}^n$, since $|x_i| \geq 0$ for all i, this implies that

$$||x||_1^2 = \left(\sum_{i=1}^n |x_i|\right)^2 \geq \sum_{i=1}^n |x_i|^2 = \|x\|_2^2.$$

Taking the square root on both sides proves that $||x||_2 \leq ||x||_1$ for all $x \in \mathbb{R}^n$. Next, note that since $|x|_i \geq 0$ for all i, we have that

$$||x||_\infty^2 = \left(\max\{|x_1|, \ldots, |x|_n\}\right)^2 \leq \sum_{i=1}^n |x_i|^2 = ||x||_2^2.$$

Taking the square root of both sides shows that $||x||_\infty \leq ||x||_2$ for all $x \in \mathbb{R}^n$. These two inequalities prove the exercise.

A2. *[3 points]* A set $A \subseteq \mathbb{R}^n$ is *convex* if $\lambda x + (1-\lambda)y \in A$ for all $x, y \in A$ and $\lambda \in [0,1]$.
For each of the grey-shaded sets below (I-III), state whether each one is convex, or state why it is not convex using any of the points a, b, c, d in your answer.

Solution:
I. Non-convex: b and c.
II. Convex
III. Non-convex: a and d.

A3. *[4 points]* We say a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex on a set A if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for all $x, y \in A$ and $\lambda \in [0, 1]$. For each of the functions shown below (I-III), state whether each is convex on the specified interval, or state why not with a counterexample using any of the points a, b, c, d in your answer.

a. Function in panel I on $[a, c]$

b. Function in panel II on $[a, c]$

c. Function in panel III on $[a, d]$

d. Function in panel III on $[c, d]$

Solution:
a. Convex.
b. Non-convex, specifically on the interval $[b, c]$.
c. Non-convex, on the interval $[a, c]$.
d. Convex.

B2. Use just the definitions above and let $\|\cdot\|$ be a norm.

a. *[3 points]* Show that $f(x) = \|x\|$ is a convex function.

b. *[3 points]* Show that $\{x \in \mathbb{R}^n : \|x\| \leq 1\}$ is a convex set.

c. *[2 points]* Draw a picture of the set $\{(x_1, x_2) \ : \ g(x_1, x_2) \leq 4\}$ where $g(x_1, x_2) = \left(|x_1|^{1/2} + |x_2|^{1/2}\right)^2$. (This is the function considered in 1b above specialized to $n = 2$.) We know g is not a norm. Is the defined set convex? Why not?

Context: It is a fact that a function f defined over a set $A \subseteq \mathbb{R}^n$ is convex if and only if the set $\{(x, z) \in \mathbb{R}^{n+1} : z \geq f(x), x \in A\}$ is convex. Draw a picture of this for yourself to be sure you understand it.

Solution:

a. Let $||\cdot||$ be a norm. By definition of a norm, for any $x, y \in \mathbb{R}^n$ we have that $||x+y|| \leq ||x|| + ||y||$. Therefore, for $\lambda \in [0, 1]$ and $x, y \in \mathbb{R}^n$,

$$\begin{aligned} f(\lambda x + (1-\lambda)y) &= ||\lambda x + (1-\lambda)y|| \\ &\leq ||\lambda x|| + ||(1-\lambda)y|| \\ &= |\lambda| \cdot ||x|| + |1-\lambda| \cdot ||y|| \\ &= \lambda||x|| + (1-\lambda)||y|| \\ &= \lambda f(x) + (1-\lambda) f(y) \end{aligned}$$

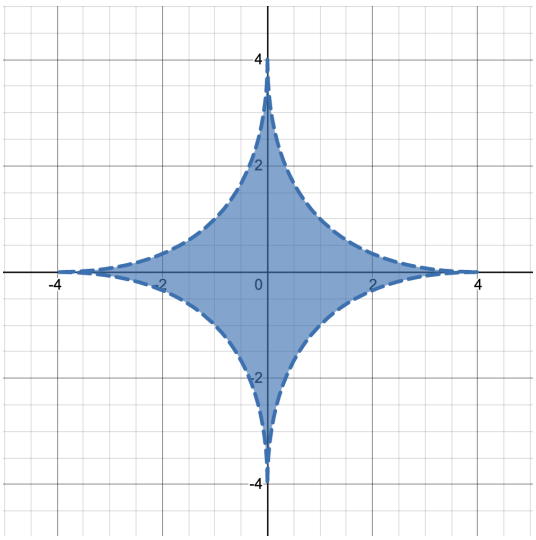
This proves that the function $f(x) = ||x||$ is convex.

b. Let $y, z \in \{x \in \mathbb{R}^n : ||x|| \leq 1\}$ and let $\lambda \in [0, 1]$. Using part (a), we see that

$$\begin{aligned} ||\lambda y + (1-\lambda)z|| &= f(\lambda y + (1-\lambda)z) \\ &\leq \lambda f(y) + (1-\lambda) f(z) \\ &= \lambda \cdot ||y|| + (1-\lambda) \cdot ||z|| \\ &= \lambda \cdot 1 + (1-\lambda) \cdot 1 \\ &= 1. \end{aligned}$$

This shows that $\lambda y + (1-\lambda)z \in \{x \in \mathbb{R}^n : ||x|| \leq 1\}$. Since y and z were arbitrary points with norm less than or equal to 1, this proves that $\{x \in \mathbb{R}^n : ||x|| \leq 1\}$ is convex.

c. The blue region below corresponds to the set $\{(x, y) \in \mathbb{R}^2 : (|x|^{1/2} + |y|^{1/2}))^2 \leq 4\}$.



B3. For $i = 1, \ldots, n$ let $\ell_i(w)$ be convex functions over $w \in \mathbb{R}^d$ (e.g., $\ell_i(w) = (y_i - w^\top x_i)^2$), $\|\cdot\|$ is any norm, and $\lambda > 0$.

a. *[3 points]* Show that

$$\sum_{i=1}^{n} \ell_i(w) + \lambda \|w\|$$

is convex over $w \in \mathbb{R}^d$ (Hint: Show that if f, g are convex functions, then $f(x) + g(x)$ is also convex.)

b. *[1 points]* Explain in one sentence why we prefer to use loss functions and regularized loss functions that are convex.

Solution:

a. Let $\ell_i(w)$ be convex functions over $\mathbb{R}^d$ for $i = 1, \ldots, n$. If $n = 2$, and if $w, v \in \mathbb{R}^d$ and $\lambda \in [0, 1]$, then

$$\begin{aligned}
(\ell_1 + \ell_2)(\lambda w + (1-\lambda)v) &= \ell_1(\lambda w + (1-\lambda)v) + \ell_2(\lambda w + (1-\lambda)v) \\
&\leq \lambda \ell_1(w) + (1-\lambda)\ell_1(v) + \lambda \ell_2(w) + (1-\lambda)\ell_2(v) \quad \text{(by convexity)} \\
&= \lambda(\ell_1(w) + \ell_2(w)) + (1-\lambda)\ell_2(v) \\
&= \lambda(\ell_1 + \ell_2)(w) + (1-\lambda)(\ell_1 + \ell_2)(v).
\end{aligned}$$

This proves that $\ell_1 + \ell_2$ is convex. Next, suppose that for some $n \geq 2$, we have that $\ell_1 + \cdots + \ell_n$ is convex and let ℓ_{n+1} be another convex function. Then, since

$$\sum_{i=1}^{n+1} \ell_i(w) = \sum_{i=1}^{n} \ell_i(w) + \ell_{n+1}(w),$$

the case of $n = 2$ shows that $\sum_{i=1}^{n+1} \ell_i(w)$ is convex. By induction, the sum of finitely many convex functions is still convex. Further, we proved in B.2 that the function $f(w) = ||w||$ is convex and thus $\lambda f(w) = \lambda ||w||$ is convex. Therefore, since $\sum_{i=1}^{n} \ell_i(w) + \lambda ||w||$ is a sum of finitely many convex functions, it is also convex.

b. We prefer to use convex loss and regularized loss functions because we must minimize said functions in order to learn a model and there are effective ways of finding minima of convex functions.

Lasso on a real dataset

A4. We will first try out your solver with some synthetic data. A benefit of the Lasso is that if we believe many features are irrelevant for predicting y, the Lasso can be used to enforce a sparse solution, effectively differentiating between the relevant and irrelevant features. Suppose that $x \in \mathbb{R}^d, y \in \mathbb{R}, k < d$, and data are generated independently according to the model $y_i = w^T x_i + \epsilon_i$ where

$$w_j = \begin{cases} j/k & \text{if } j \in \{1, \ldots, k\} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is noise (note that in the model above $b = 0$). We can see from Equation (1) that since $k < d$ and $w_j = 0$ for $j > k$, the features $k + 1$ through d are irrelevant for predicting y.
Generate a dataset using this model with $n = 500, d = 1000, k = 100$, and $\sigma = 1$. You should generate the dataset such that each $\epsilon_i \sim \mathcal{N}(0, 1)$, and y_i is generated as specified above.

a. *[10 points]* With your synthetic data, solve multiple Lasso problems on a regularization path, starting at λ_{max} where no features are selected (see Equation (**??**)) and decreasing λ by a constant ratio (e.g., 2) until nearly all the features are chosen. In plot 1, plot the number of non-zeros as a function of λ on the x-axis (Tip: use `plt.xscale('log')`).

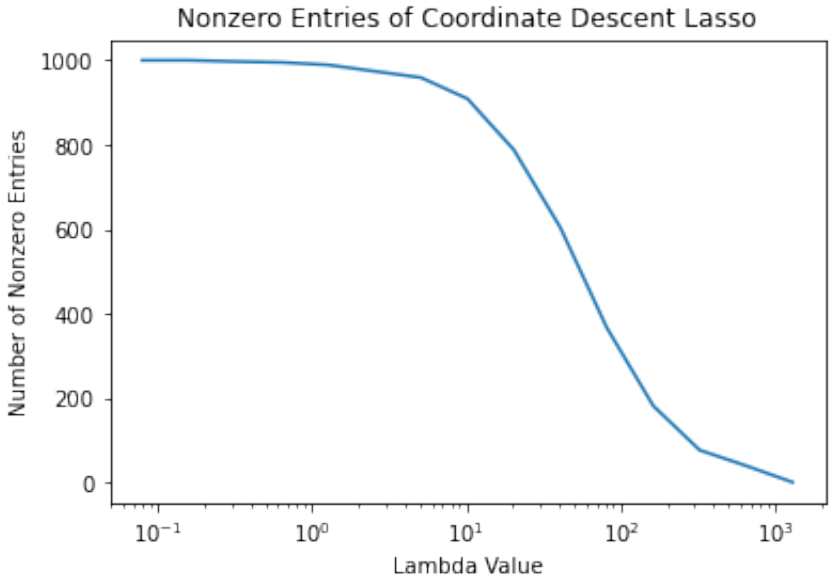
b. *[10 points]* For each value of λ tried, record values for false discovery rate (FDR) (number of incorrect nonzeros in $\widehat{w}$/total number of nonzeros in $\widehat{w}$) and true positive rate (TPR) (number of correct nonzeros in $\widehat{w}$/k). Note: for each j, $\widehat{w}_j$ is an incorrect nonzero if and only if $\widehat{w}_j \neq 0$ while $w_j = 0$. In plot 2, plot these values with the x-axis as FDR, and the y-axis as TPR.

   Note that in an ideal situation we would have an (FDR,TPR) pair in the upper left corner. We can always trivially achieve $(0, 0)$ and $(\frac{d-k}{d}, 1)$.

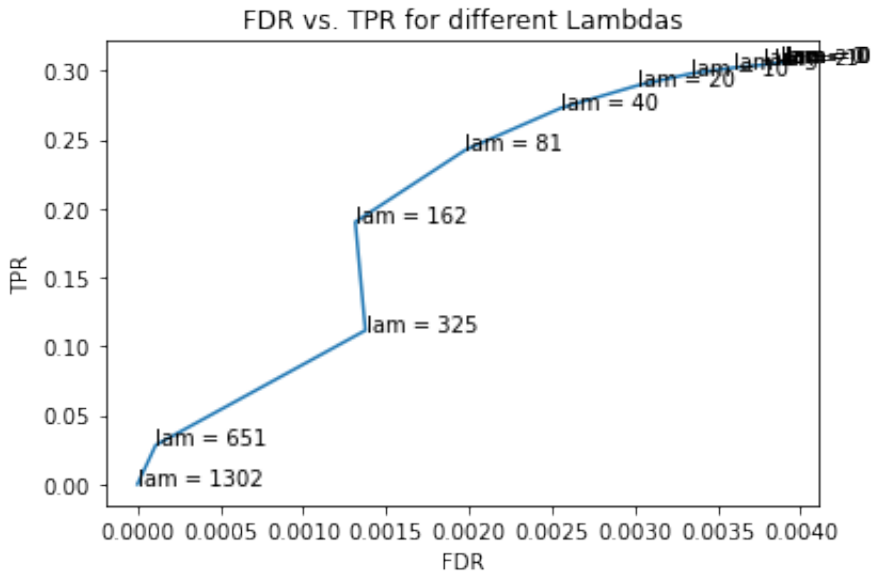
c. *[5 points]* Comment on the effect of λ in these two plots in 1-2 sentences.

Solution:

a. Below is the plot showing how larger regularization parameter λ increases the number of nonzero entries in the model found using coordinate descent lasso.



b. Below shows the False Detection Rate (FDR) vs True Positive Rate (TPR) for the values of lambda used in part (a). Note that the values of lambdas are rounded to the nearest integer to save space.



Code

Below is the code used to generate the graphs in (a) and (b)

```
def coord_desc_lasso(w,X,Y,lam):
    d = len(X[0,:])-1   #d=num features
    b = (1/n)*np.sum(Y - X[:,:d]@w[:d])
    a=2*(np.sum(X*X,axis=0))
    c=np.zeros(d)
    for k in range(d):
        c[k] = 2*X[:,k]@(Y-(b + np.delete(X[:,:d],k,axis=1) @ np.delete(w[:d],k)))
        if c[k] < -lam:
            w[k] = (c[k]+lam)/a[k]
        elif c[k] >= -lam and c[k] <= lam:
            w[k] = 0
        else:
            w[k] = (c[k]-lam)/a[k]
    return w
```

```
n=500
d=1000
k=100
X=np.zeros((n,d+1))   #extra column
X[:,:d]=np.random.multivariate_normal(np.zeros(d),np.identity(d),n)
X[:,d] = np.random.normal(0,1,n)  #last column holds errors epsilon_i
w=np.zeros(d+1)
for j in range(k):
    w[j] = (j+1)/k
w[d] = 1 #set last component to 1 for vectorization
Y = X@w
```

```
avg_y = np.mean(Y)
max_lam = max( [2*abs(np.sum((Y-avg_y)*X[:,k])) for k in range(d)] )
lam = [max_lam/((2)**j) for j in range(15)]
num_variables_zero = np.zeros(15)
keepIterating = True
FDR=np.zeros(15)
TPR=np.zeros(15)
for j in range(15):
    current_w = np.zeros(d+1)
    while keepIterating:
        new_w = coord_desc_lasso(current_w,X,Y,lam[j])
        keepIterating = (max(abs(new_w - current_w)) > .01)
    num_variables_zero[j] = np.sign(new_w).dot(np.sign(new_w))
    FDR[j] = (new_w[k:d].dot(new_w[k:d])) / num_variables_zero[j]
    TPR[j] = (new_w[0:k].dot(new_w[0:k])) / k
    keepIterating = True
```

```
plt.plot(lam,num_variables_zero)
plt.title('Nonzero Entries of Coordinate Descent Lasso')
plt.xscale('log')
plt.xlabel('Lambda Value')
plt.ylabel('Number of Nonzero Entries')
```

```
plt.plot(FDR,TPR)
plt.xlabel("FDR")
plt.ylabel("TPR")
plt.title("FDR vs. TPR for different Lambdas")
for j in range(15):
    plt.annotate("lam = "+ str(int(lam[j])), (FDR[j],TPR[j]))
```

c. In (a), we see that a larger regularization parameter λ leads to more nonzero entries. This is because the lasso method prefers spare solutions. In (b), we see that larger λ corresponds to worse FDR and TPR. This makes sense given the graph in (a), and because we know that the true model w has only 100 nonzero entries, so when λ is too small there are too many nonzero entries.

A5. We'll now put the Lasso to work on some real data. Download the training data set "crime-train.txt" and the test data set "crime-test.txt" from the website. Store your data in your working directory, ensure you have the `pandas` library for Python installed, and read in the files with:

```
import pandas as pd
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")
```

This stores the data as Pandas `DataFrame` objects. `DataFrame`s are similar to Numpy `array`s but more flexible; unlike `array`s, `DataFrame`s store row and column indices along with the values of the data. Each column of a `DataFrame` can also store data of a different type (here, all data are floats). Here are a few commands that will get you working with Pandas for this assignment:

```
df.head()                   # Print the first few lines of DataFrame df.
df.index                    # Get the row indices for df.
df.columns                  # Get the column indices.
df[''foo'']                 # Return the column named ''foo''.
df.drop(''foo'', axis = 1)  # Return all columns except ''foo''.
df.values                   # Return the values as a Numpy array.
df[''foo''].values          # Grab column foo and convert to Numpy array.
df.iloc[:3,:3]              # Use numerical indices (like Numpy) to get 3 rows and cols.
```

The data consist of local crime statistics for 1,994 US communities. The response y is the rate of violent crimes reported per capita in a community. The name of the response variable is `ViolentCrimesPerPop`, and it is held in the first column of `df_train` and `df_test`. There are 95 features. These features include many variables. Some features are the consequence of complex political processes, such as the size of the police force and other systemic and historical factors. Others are demographic characteristics of the community, including self-reported statistics about race, age, education, and employment drawn from Census reports.

The goals of this problem are threefold: (i) to encourage you to think about how data collection processes affect the resulting model trained from that data; (ii) to encourage you to think deeply about models you might train and how they might be misused; and (iii) to see how Lasso encourages sparsity of linear models in settings where d is large relative to n. **We emphasize that training a model on this dataset can suggest a degree of correlation between a community's demographics and the rate at which a community experiences and reports violent crime. We strongly encourage students to consider why these correlations may or may not hold more generally, whether correlations might result from a common cause, and what issues can result in misinterpreting what a model can explain.**

The dataset is split into a training and test set with 1,595 and 399 entries, respectively[1]. We will use this training set to fit a model to predict the crime rate in new communities and evaluate model performance on the test set. As there are a considerable number of input variables and fairly few training observations, overfitting is a serious issue. In order to avoid this, use the coordinate descent Lasso algorithm implemented in the previous problem.

a. *[4 points]* Read the documentation for the original version of this dataset: `http://archive.ics.uci.edu/ml/datasets/communities+and+crime`. Report 3 features included in this dataset for which historical *policy* choices in the US would lead to variability in these features. As an example, the *number of police* in a community is often the consequence of decisions made by governing bodies, elections, and amount of tax revenue available to decisionmakers.

b. *[4 points]* Before you train a model, describe 3 features in the dataset which might, if found to have nonzero weight in model, be interpreted as *reasons* for higher levels of violent crime, but which might actually be a *result* rather than (or in addition to being) the cause of this violence.

[1]The features have been standardized to have mean 0 and variance 1.

Now, we will run the Lasso solver. Begin with $\lambda = \lambda_{\max}$ defined in Equation (**??**). Initialize all weights to 0. Then, reduce λ by a factor of 2 and run again, but this time initialize $\hat{w}$ from your $\lambda = \lambda_{\max}$ solution as your initial weights, as described above. Continue the process of reducing λ by a factor of 2 until $\lambda < 0.01$. For all plots use a log-scale for the λ dimension (Tip: use `plt.xscale('log')`).

c. *[4 points]* Plot the number of nonzero weights of each solution as a function of λ.

d. *[4 points]* Plot the regularization paths (in one plot) for the coefficients for input variables `agePct12t29`, `pctWSocSec`, `pctUrban`, `agePct65up`, and `householdsize`.

e. *[4 points]* On one plot, plot the squared error on the training and test data as a function of λ.

f. *[4 points]* Sometimes a larger value of λ performs nearly as well as a smaller value, but a larger value will select fewer variables and perhaps be more interpretable. Inspect the weights $\hat{w}$ for $\lambda = 30$. Which feature had the largest (most positive) Lasso coefficient? What about the most negative? Discuss briefly.

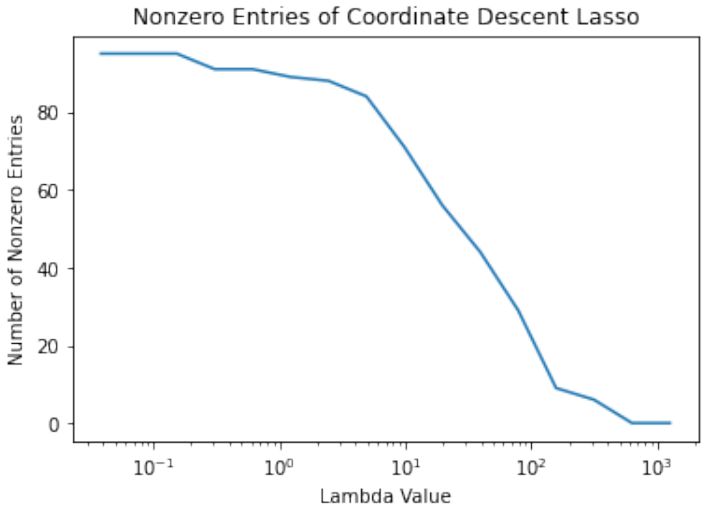
g. *[4 points]* Suppose there was a large negative weight on `agePct65up` and upon seeing this result, a politician suggests policies that encourage people over the age of 65 to move to high crime areas in an effort to reduce crime. What is the (statistical) flaw in this line of reasoning? (Hint: fire trucks are often seen around burning buildings, do fire trucks cause fire?)

Solution:

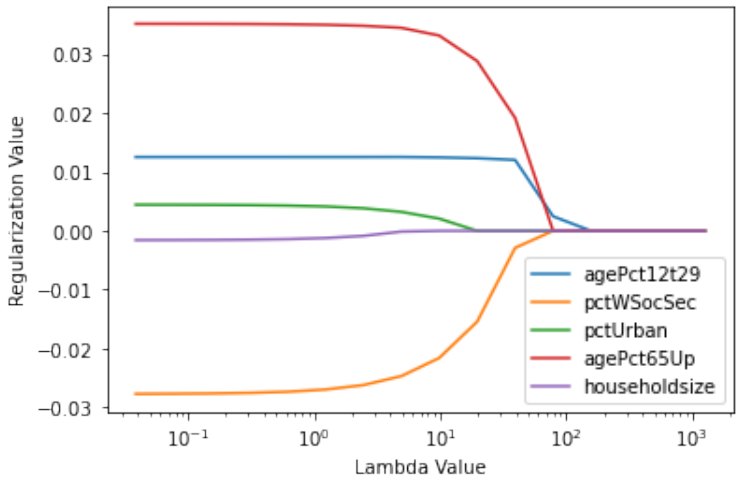
a. Number under the poverty line, percent unemployed, and percent of the police that are African American are features that can be affected by historical policy choices.

b. Number of per captia police officers, police operating budget, and number of unemployed may seem correlated to violent crime while being in fact caused by it.

c. Below is the plot showing the number of nonzero entries in each solution of coordinate descent as a function of λ.



d. Here we see the coefficient values for various features as we increase λ.



Code
For (c) and (d)

```
import pandas as pd
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")
X_train = df_train.values[:,1:96]
Y_train = df_train.values[:,0]
X_test = df_test.values[:,1:96]
Y_test = df_test.values[:,0]
```

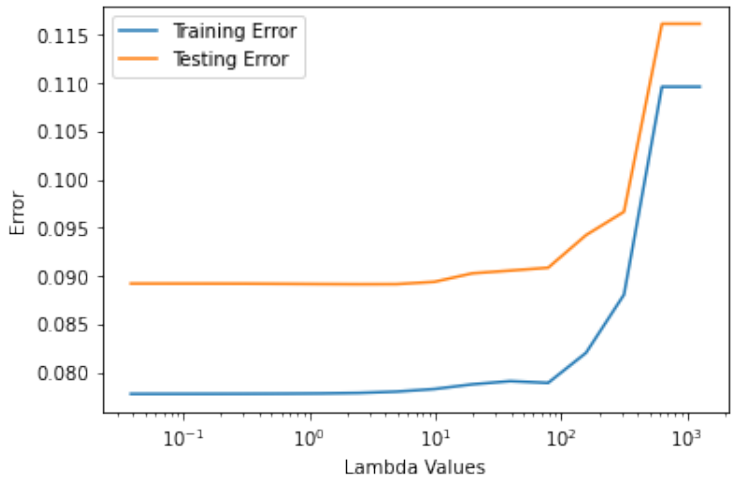
```
n = len(Y_train)
d = len(X_train[0,:])
avg_y = np.mean(Y_train)
max_lam = max( [2*abs(np.sum((Y-avg_y)*X[:,k])) for k in range(d)] )
```

```
lam = [max_lam/((2)**j) for j in range(16)]
num_variables_zero = np.zeros(16)
keepIterating = True
models = np.zeros((16,d))
for j in range(16):
    current_w = np.zeros(d)
    while keepIterating:
        new_w = coord_desc_lasso_crime(current_w,X_train,Y_train,lam[j])
        keepIterating = (max(abs(new_w - current_w)) > .1)
    num_variables_zero[j] = np.sign(new_w).dot(np.sign(new_w))
    models[j,:] = new_w
    keepIterating = True
```

```
plt.plot(lam,num_variables_zero)
plt.title('Nonzero Entries of Coordinate Descent Lasso')
plt.xscale('log')
plt.xlabel('Lambda Value')
plt.ylabel('Number of Nonzero Entries')
```

```
plt.plot(lam, models[:,3], label = "agePct12t29")
plt.plot(lam, models[:,12], label = "pctWSocSec")
plt.plot(lam, models[:,7], label = "pctUrban")
plt.plot(lam, models[:,5], label = "agePct65Up")
plt.plot(lam, models[:,1], label = "householdsize")
plt.xscale('log')
plt.xlabel('Lambda Value')
plt.ylabel('Regularization Value')
plt.legend()
```

e. Here we show the squared training and testing error of each solution as a function of λ.



Code

```
training_error = np.zeros(16)
test_error = np.zeros(16)
for j in range(16):
    train_predict = X_train @ models[j,:].transpose()
    training_error[j] = ((Y_train - train_predict).dot(Y_train - train_predict))/n
    test_predict = X_test @ models[j,:].transpose()
    test_error[j] = ((Y_test - test_predict).dot(Y_test - test_predict))/len(X_test[:,0])
plt.plot(lam, training_error, label = "Training Error")
plt.plot(lam, test_error, label = "Testing Error")
plt.xscale('log')
plt.xlabel('Lambda Values')
plt.ylabel('Error')
plt.legend()
```

f. We found that the most positively correlated feature was population and the most negatively correlated was median income. It makes sense that in crowded cities there will be more violent crime, and that violent crime would be negatively correlated to median income.

Code

```
lam = 30
keepIterating = True
current_w = np.zeros(d)
while keepIterating:
    new_w = coord_desc_lasso_crime(current_w,X_train,Y_train,lam)
    keepIterating = (max(abs(new_w - current_w)) > .01)
least_corr = np.argmin(new_w)
most_corr = np.argmax(new_w)
print(least_corr)
print(most_corr)
```

Logistic Regression

Binary Logistic Regression

A6. Here we consider the MNIST dataset, but for binary classification. Specifically, the task is to determine whether a digit is a 2 or 7. Here, let $Y = 1$ for all the "7" digits in the dataset, and use $Y = -1$ for "2". We will use regularized logistic regression. Given a binary classification dataset $\{(x_i, y_i)\}_{i=1}^n$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ we showed in class that the regularized negative log likelihood objective function can be written as

$$J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i(b + x_i^T w))) + \lambda ||w||_2^2$$

Note that the offset term b is not regularized. For all experiments, use $\lambda = 10^{-1}$. Let $\mu_i(w, b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$.

a. *[8 points]* Derive the gradients $\nabla_w J(w, b)$, $\nabla_b J(w, b)$ and give your answers in terms of $\mu_i(w, b)$ (your answers should not contain exponentials).

b. *[8 points]* Implement gradient descent with an initial iterate of all zeros. Try several values of step sizes to find one that appears to make convergence on the training set as fast as possible. Run until you feel you are near to convergence.

   (i) For both the training set and the test, plot $J(w, b)$ as a function of the iteration number (and show both curves on the same plot).

   (ii) For both the training set and the test, classify the points according to the rule $\text{sign}(b + x_i^T w)$ and plot the misclassification error as a function of the iteration number (and show both curves on the same plot).

   Reminder: Make sure you are only using the test set for evaluation (not for training).

c. *[7 points]* Repeat (b) using stochastic gradient descent with a batch size of 1. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a). Show both plots described in (b) when using batch size 1. Take careful note of how to scale the regularizer.

d. *[7 points]* Repeat (b) using stochastic gradient descent with batch size of 100. That is, instead of approximating the gradient with a single example, use 100. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a).

Solution:

a. Let $j = 1, \ldots, d$. Then, we see that

$$\frac{\partial}{\partial w_j} J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_{ij} \exp\left(-y_i(b + x_i^T w)\right)}{1 + \exp\left(-y_i(b + x_i^T w)\right)} + 2\lambda w_j$$

and therefore

$$\begin{aligned}
\nabla_w J(w, b) &= \frac{1}{n} \sum_{i=1}^{n} \left[\Big(- y_i \exp\left(-y_i(b + x_i^T w)\right) \mu_i(w, b) \Big) x_i \right] + 2\lambda w \\
&= \frac{1}{n} \sum_{i=1}^{n} \left[\Big(- y_i \Big(\frac{1}{\mu_i(w, b)} - 1 \Big) \mu_i(w, b) \Big) x_i \right] + 2\lambda w \\
&= -\frac{1}{n} \sum_{i=1}^{n} \Big[y_i (1 - \mu_i(w, b)) x_i \Big] + 2\lambda w
\end{aligned}$$

Since $b \in \mathbb{R}$,

$$\begin{aligned}
\nabla_b J(w,b) &= \frac{\partial}{\partial b} J(w,b) \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i \exp\left(-y_i(b + x_i^T w)\right)}{1 + \exp\left(-y_i(b + x_i^T w)\right)} \\
&= \frac{1}{n}\sum_{i=1}^{n} -y_i \exp\left(-y_i(b + x_i^T w)\right)\mu_i(w,b) \\
&= -\frac{1}{n}\sum_{i=1}^{n} -y_i(1 - \mu_i(w,b)).
\end{aligned}$$

Code

Here are the functions written for this problem. There is one for gradient descent and stochastic gradient descent, as well as `loss` which evaluates $J(w,b)$ and `classify` which returns a prediction given a model.

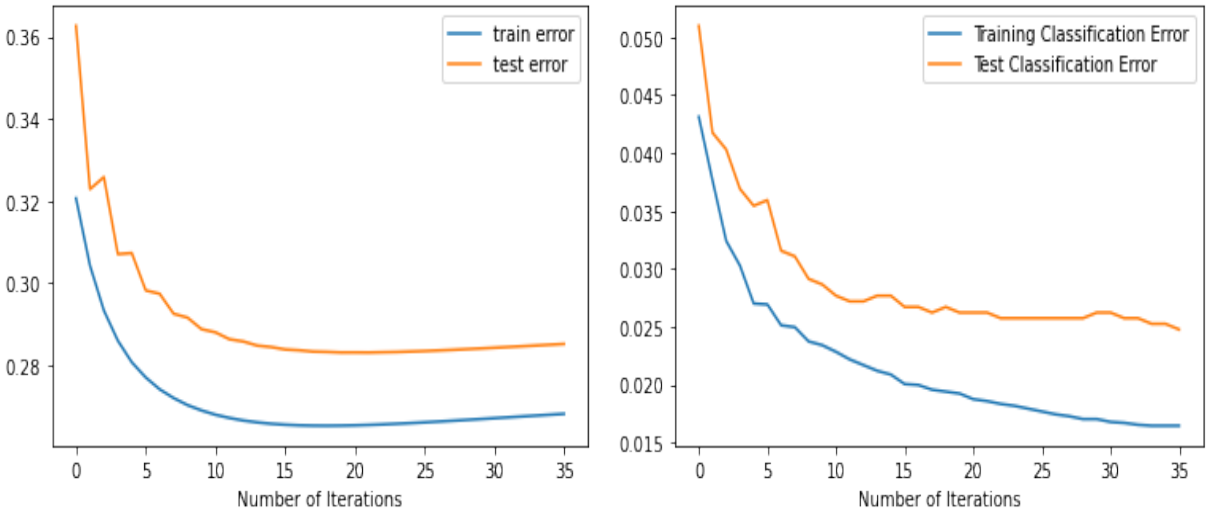
```
#computes gradient descent, returns whether grad is small enough
#w = model
#y = labels, +-1
#X = data
#lam = regularization constant
#eta = step size
def grad_descent(w,X,y,lam,eta):
    n = len(w)-1  #785-1=784
    mu = 1/(1+np.exp(-y*(X@w)))
    grad = np.zeros(n+1)    #gradient
    temp = y*(1-mu)
    grad[:n] = -(1/n)*(  np.transpose(temp)  @X[:,:n]) + 2*lam*w[:n]
    grad[n] = -(1/n)*(sum(temp))
    keepIterating = (la.norm(grad)>.1)
    return [w - eta*grad, keepIterating]

#computes iteration of stochastic gradient descent
#batch size = number of variables to use
def stoc_grad_descent(w,X,y,lam,eta,batch_size):
    n = len(w)-1  #785-1=784
    random_variables = np.random.randint(0,n,size=batch_size)
    grad = np.zeros(n+1)    #gradient
    mu = 1/(1+np.exp(-y[random_variables]*(X[random_variables,:]@w)))
    temp = y[random_variables]*(1-mu)
    grad[:n] = -(1/batch_size)*(  np.transpose(temp)  @X[random_variables,:n]) + 2*lam*w[:n]
    grad[n] = -(1/batch_size)*(sum(temp))
    keepIterating = (la.norm(grad)>.1)
    return [w - eta*grad, keepIterating]

#computes J(w,b) for part b(i)
def loss(w,X,y,lam):
    n = len(X[:,0])
    J = (1/n)*np.sum(np.log(1+np.exp(-y*(X@w)))) + lam*la.norm(w)
    return J
```

```
#computes classification error for b(ii)
def classify(w,X):
    return np.sign(X@w)
```

b. Below are the results showing the negative log likelihood objective function $J(w,b)$ (left) and the iterating model's classification error's (right) vs iteration number using gradient descent. We used a step-size of 0.1 and found that it did not take many iterations before the error started to converge. We see that the near-optimal model has low testing error under 3%.



Code

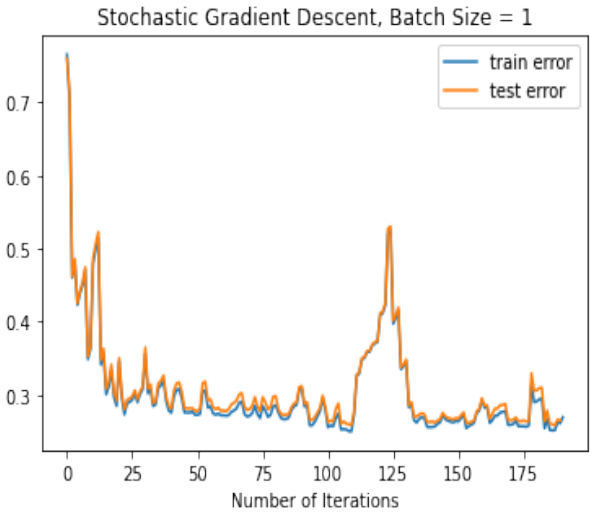
```
n=len(labels_train)
n2=len(labels_test)
d=784
X_train_new = np.zeros([n,d+1])
X_test_new = np.zeros([n2,d+1])
X_train_new[:,:d] = X_train
X_test_new[:,:d] = X_test
X_train_new[:,d] = np.ones(n)
X_test_new[:,d] = np.ones(n2)
```

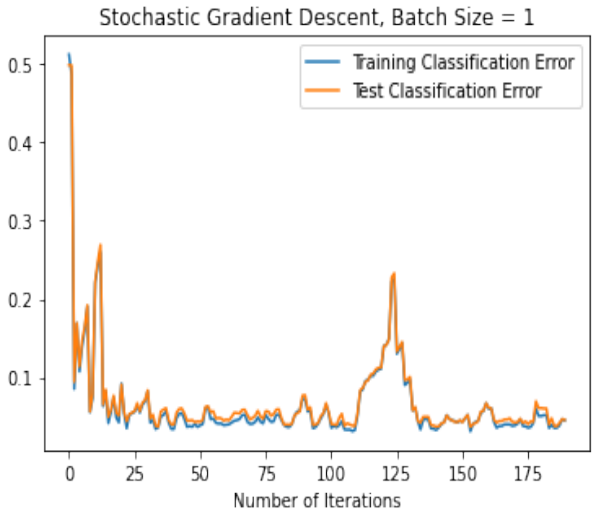
```
current_w = np.zeros(d+1)
new_w = np.zeros(d+1)
eta = .1
lam = .1
error_train=[]
error_test=[]
train_class_err = []
test_class_err = []
keepIterating = True
while keepIterating:
    current_w = new_w
    [new_w, keepIterating] = grad_descent(current_w,X_train_new,labels_train,lam,eta)
    error_train.append(loss(new_w,X_train_new,labels_train,lam))
    error_test.append(loss(new_w,X_test_new,labels_test,lam))
    predict_train = classify(new_w, X_train_new)
    predict_test = classify(new_w, X_test_new)
    train_class_err.append(sum((1/2)*abs(labels_train-predict_train))/len(X_train_new[:,0]))
    test_class_err.append(sum((1/2)*abs(labels_test-predict_test))/len(X_test_new[:,0]))
```

```
plt.plot(error_train,label="train error")
plt.plot(error_test, label="test error")
plt.xlabel("Number of Iterations")
plt.legend()
```

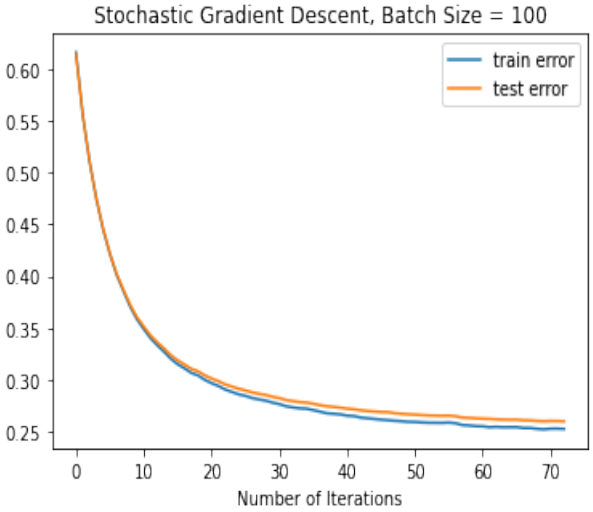
```
plt.plot(train_class_err, label="Training Classification Error")
plt.plot(test_class_err, label="Test Classification Error")
plt.xlabel("Number of Iterations")
plt.legend()
```

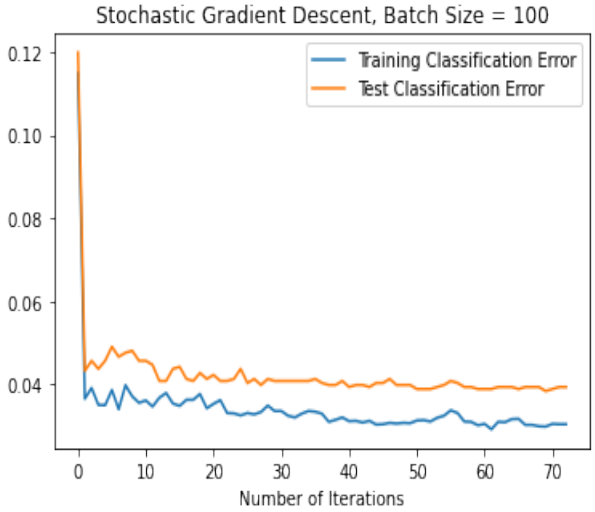
c. Below are the analogous results using stochastic gradient descent on a batch size of 1. We see that the error is sporadic. Unlucky selection of which variable to include in the calculation of $\nabla J(w, b)$ sometimes causes an iteration to be much worse than previous. However, we see improvement overall in the error.





d. Below are the results using stochastic gradient descent with a batch size of 100. We see that these results are much better than using a batch size of 1, and closely resemble our results using standard gradient descent. This, along with results from (c), suggest that stochastic gradient descent with a batch size large enough is an effective method of speeding up gradient descent and still achieving good results.





Code

Below is the code used to create the figures in (c) and (d). My function for stochastic gradient descent took a `batch-size` variable as input so it was used for both (c) and (d).

```
# (c) stochastic gradient descent, batch size = 1
batch_size = 100
current_w = np.zeros(d+1)
new_w = np.zeros(d+1)
eta = .05
lam = .1
error_train=[]
error_test=[]
train_class_err = []
test_class_err = []
keepIterating = True
while keepIterating:
    current_w = new_w
    [new_w, keepIterating] = stoc_grad_descent(current_w,X_train_new,labels_train,lam,eta,batch_size)
    error_train.append(loss(new_w,X_train_new,labels_train,lam))
    error_test.append(loss(new_w,X_test_new,labels_test,lam))
    predict_train = classify(new_w, X_train_new)
    predict_test = classify(new_w, X_test_new)
    train_class_err.append(sum((1/2)*abs(labels_train-predict_train))/len(X_train_new[:,0]))
    test_class_err.append(sum((1/2)*abs(labels_test-predict_test))/len(X_test_new[:,0]))
```

Multinomial Logistic Regression *[25 points]*

B4.
We've talked a lot about binary classification, but what if we have $k > 2$ classes, like the 10 digits of MNIST? Concretely, suppose you have a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, k\}$ (i.e., k column vectors of length d, stacked side-by-side). Like in our least squares classifier of homework 1 for MNIST, we will assign a separate weight vector $\mathbf{w}^{(\ell)}$ for each class $\ell = 1, \ldots, k$; let $W = [\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(k)}] \in \mathbb{R}^{d \times k}$. We can generalize the binary classification probabilistic model to multiple classes as follows: let

$$\mathbb{P}_W(y_i = \ell | W, \mathbf{x}_i) = \frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}$$

The negative log-likelihood function is equal to

$$\mathcal{L}(W) = -\sum_{i=1}^n \sum_{\ell=1}^k \mathbf{1}\{y_i = \ell\} \log \left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)} \right)$$

Define the $\texttt{softmax}(\cdot)$ operator to be the function that takes in a vector $\theta \in \mathbb{R}^k$ and outputs a vector in $\mathbb{R}^k$ whose mth component is equal to $\frac{\exp(\theta_m)}{\sum_{j=1}^k \exp(\theta_j)}$. Clearly, this vector is nonnegative and sums to one. If for any m we have $\theta_m \gg \max_{j \neq m} \theta_j$ then $\texttt{softmax}(\theta)$ approximates $\mathbf{e}_m$, a vector of all zeros with a one in the mth component.
For each y_i let $\mathbf{y}_i$ be the one-hot encoding of y_i (i.e., $\mathbf{y}_i \in \{0, 1\}^k$ is a vector of all zeros aside from a 1 in the y_ith index, or $\mathbf{e}_{y_i}$).

a. *[5 points]* If $\widehat{\mathbf{y}}_i^{(W)} = \texttt{softmax}(W^\top \mathbf{x}_i)$, show that $\nabla_W \mathcal{L}(W) = -\sum_{i=1}^n \mathbf{x}_i (\mathbf{y}_i - \widehat{\mathbf{y}}_i^{(W)})^\top$.

b. *[5 points]* Recall *Ridge Regression on MNIST* problem in Homework 1 and define $J(W) = \frac{1}{2} \sum_{i=1}^n \|\mathbf{y}_i - W^\top \mathbf{x}_i\|_2^2$. If $\widetilde{\mathbf{y}}_i^{(W)} = W^\top \mathbf{x}_i$, show that $\nabla_W J(W) = -\sum_{i=1}^n \mathbf{x}_i (\mathbf{y}_i - \widetilde{\mathbf{y}}_i^{(W)})^\top$. Remark: Comparing the least squares linear regression gradient step of this part to the gradient step of minimizing the negative log likelihood of the logistic model of part a may shed light on why we call this classification problem *logistic regression*.

c. *[15 points]* Using the original representations of the flattened MNIST images $\mathbf{x}_i \in \mathbb{R}^d$ ($d = 28 \times 28 = 764$) and all $k = 10$ classes, implement gradient descent (or stochastic gradient descent) for both $J(W)$ and $\mathcal{L}(W)$ and run until convergence on the training set. For each of the two solutions, report the classification accuracy on the training and test sets using the classification rule $\arg\max_j \mathbf{e}_j W^\top \mathbf{x}_i$.

Solution:

a. Let $i = 1, \ldots, n$ be fixed. If $m \neq y_i$, then

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}^{(m)}} \left[\sum_{\ell=1}^k \mathbf{1}\{y_i = \ell\} \log \left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)} \right) \right] &= \frac{\partial}{\partial \mathbf{w}^{(m)}} \left[\log \left(\frac{\exp(\mathbf{w}^{(y_i)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)} \right) \right] \\
&= \frac{\partial}{\partial \mathbf{w}^{(m)}} \left[\mathbf{w}^{(y_i)} \cdot \mathbf{x}_i - \log \sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i) \right] \\
&= -\frac{\mathbf{x}_i \exp(\mathbf{w}^{(m)} \cdot_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}.
\end{aligned}$$

On the other hand, for $m = y_i$ we have

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}^{(y_i)}}\left[\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \log\left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}\right)\right] &= \frac{\partial}{\partial \mathbf{w}^{(y_i)}}\left[\mathbf{w}^{(y_i)} \cdot \mathbf{x}_i - \log \sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)\right] \\ &= \mathbf{x}_i - \frac{\mathbf{x}_i \exp(\mathbf{w}^{(y_i)} \cdot \mathbf{x}_i)}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)} \\ &= \mathbf{x}_i\left(1 - \frac{\exp(\mathbf{w}^{(y_i)} \cdot \mathbf{x}_i)}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}\right)^T.\end{aligned}$$

Notice however that the mth term of $W^T\mathbf{x}_i$ is $\mathbf{w}^{(m)} \cdot \mathbf{x}_i$ and thus

$$\frac{\exp(\mathbf{w}^{(m)} \cdot \mathbf{x}_i)}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}$$

is the mth term of $\texttt{softmax}(W^T\mathbf{x}_i)$. Therefore,

$$\nabla_W \mathcal{L}(W) = -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - \widehat{\mathbf{y}}_i^{(W)})^T.$$

b. Let $J(W) = \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{y}_i - W^T\mathbf{x}_i\|_2^2$ and let $\widetilde{\mathbf{y}}_i^{(W)} = W^T\mathbf{x}_i$. Similarly as part (a), fix $i = 1, \ldots, n$ and let $m = 1, \ldots, k$ be such that $m \neq y_i$. Note that

$$\begin{aligned}J(W) &= \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{y}_i - W^T\mathbf{x}_i\|_2^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{\ell=1}^{k} (\mathbf{y}_{i\ell} - \mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)^2,\end{aligned}$$

and therefore,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}^{(m)}} \frac{1}{2}\sum_{\ell=1}^{k} (\mathbf{y}_{i\ell} - \mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)^2 &= \frac{1}{2}\Big(2(\mathbf{y}_{im} - \mathbf{w}^{(m)} \cdot \mathbf{x}_i)(-\mathbf{x}_i)\Big) \\ &= -(\mathbf{y}_{im} - \mathbf{w}^{(m)} \cdot \mathbf{x}_i)\mathbf{x}_i \\ &= -(0 - \mathbf{w}^{(m)} \cdot \mathbf{x}_i)\mathbf{x}_i \\ &= (\mathbf{w}^{(m)} \cdot \mathbf{x}_i)\mathbf{x}_i,\end{aligned}$$

where the second to last equality follows since $m \neq y_i$ so $\mathbf{y}_{im} = 0$. Now if $m = y_i$, then $\mathbf{y}_{im} = \mathbf{y}_{iy_i} = 1$ and therefore

$$\frac{\partial}{\partial \mathbf{w}^{(y_i)}} \frac{1}{2}\sum_{\ell=1}^{k} (\mathbf{y}_{i\ell} - \mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)^2 = -(1 - \mathbf{w}^{(y_i)} \cdot \mathbf{x}_i)\mathbf{x}_i.$$

Since the mth term of $W^T\mathbf{x}_i$ is equal to $\mathbf{w}^{(m)} \cdot \mathbf{x}_i$, we have that

$$\nabla_W J(W) = -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - \widehat{\mathbf{y}}_i^{(W)})^T.$$

c. For both parts of this problem, we use a learning rate of .01 as well as stochastic gradient descent with a batch size of 100. We found that after training the model using negative log-likelihood $\mathcal{L}$ as our loss function, our model had an accuracy rate of 88.5% on the training data and a 89.3% accuracy rate on the test data. Using least squares linear regression and loss function $\mathcal{J}$, we found that our model had a training accuracy of

83.5% and test accuracy of 84.6% Below is the code, which only shows training using $\mathcal{L}$ as the loss function. We do this to save space, because the only difference in the code is changing `loss = nn.CrossEntropyLoss()` to `loss = nn.MSELoss()`.

```
import torch
from torch import nn
from torch import optim
from torchvision import datasets, transforms
from torch.utils.data import random_split, DataLoader
import numpy as np
import matplotlib.pyplot as plt
```

```
X_train = datasets.MNIST('mnist2', train=True, download = True, transform=transforms.ToTensor())
train_loader = DataLoader(X_train, batch_size=100)
```

```
model = nn.Sequential(
    nn.Linear(784,10)
)
```

```
optimizer = optim.SGD(model.parameters(), lr=.01)  #use stochastic gradient descent
```

```
loss = nn.CrossEntropyLoss()
```

```
epochs = 5
for epoch in range(epochs):
    for batch in train_loader:
        image, label = batch
        b = image.size(0)
        image = image.view(b,-1)

        y_hat = model(image) #prediction
        J = loss(y_hat, label)  #compute objective

        model.zero_grad()   #zero out weights
        J.backward()

        optimizer.step()   #apply gradient descent

    print(f'Epoch {epoch+1}, train loss: {J.item()}')
```

```
Epoch 1, train loss: 0.870591938495636
Epoch 2, train loss: 0.6635335683822632
Epoch 3, train loss: 0.5731456875801086
Epoch 4, train loss: 0.5208339095115662
Epoch 5, train loss: 0.4862547218799591
```

```
X_train2 = datasets.MNIST('mnist2', train=True, download = True, transform=transforms.ToTensor())
train_loader2 = DataLoader(X_train2, batch_size=60000)
```

```
for batch in train_loader2:
    image, label = batch
    b = image.size(0)
    image = image.view(b,-1)

    y_hat = model(image)
    predictions = torch.zeros((60000,1))
    for j in range(60000):
        predictions[j] = torch.argmax(y_hat[j,:])

    accuracy = 60000
    for j in range(60000):
        if predictions[j] != label[j]:
            accuracy = accuracy - 1
    accuracy = accuracy / 60000
print(accuracy)
```

```
0.8854166666666666
```

```
X_test = datasets.MNIST('mnist2', train=False, download = True, transform=transforms.ToTensor())
train_loader_test = DataLoader(X_test, batch_size=10000)
```

```
for batch in train_loader_test:
    image, label = batch
    b = image.size(0)
    image = image.view(b,-1)

    y_hat = model(image)
    predictions = torch.zeros((10000,1))
    for j in range(10000):
        predictions[j] = torch.argmax(y_hat[j,:])

    accuracy = 10000
    for j in range(10000):
        if predictions[j] != label[j]:
            accuracy = accuracy - 1
    accuracy = accuracy / 10000
print(accuracy)
```

```
0.8929
```

Confidence Interval of Least Squares Estimation

Bounding the Estimate *[30 points]*

B5. Let us consider the setting, where we have n inputs, $X_1, ..., X_n \in \mathbb{R}^d$, and n observations $Y_i = \langle X_i, \beta^* \rangle + \epsilon_i$, for $i = 1, ..., n$. Here, β^* is a ground truth vector in $\mathbb{R}^d$ that we are trying to estimate, the noise $\epsilon_i \sim \mathcal{N}(0, 1)$, and the n examples piled up — $X \in R^{n \times d}$. To estimate, we use the least squares estimator $\widehat{\beta} = \min_\beta \|X\beta - Y\|_2^2$. Moreover, we will use $n = 20000$ and $d = 10000$ in this problem.

a. *[3 points]* Show that $\widehat{\beta}_j \sim \mathcal{N}(\beta_j^*, (X^T X)^{-1}_{j,j})$ for each $j = 1, ..., d$. *(Hint: see notes on confidence intervals from lecture.)*

b. *[4 points]* Fix $\delta \in (0, 1)$ suppose $\beta^* = 0$. Applying the proposition from the notes, conclude that for each $j \in [d]$, with probability at least $1 - \delta$, $|\widehat{\beta}_j| \leq \sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$. Can we conclude that with probability at least $1 - \delta$, $|\widehat{\beta}_j| \leq \sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$ for all $j \in [d]$ simultaneously? Why or why not?

c. *[5 points]* Let's explore this question empirically. Assume data is generated as $x_i = \sqrt{(i \mod d) + 1} \cdot e_{(i \mod d)+1}$ where e_i is the ith canonical vector and $i \mod d$ is the remainder of i when divided by d. Generate each y_i according to the model above. Compute $\widehat{\beta}$ and plot each $\widehat{\beta}_j$ as a scatter plot with the x-axis as $j \in \{1, \ldots, d\}$. Plot $\pm\sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$ as the upper and lower confidence intervals with $1 - \delta = 0.95$. How many $\widehat{\beta}_j$'s are outside the confidence interval? *Hint: Due to the special structure of how we generated x_i, we can compute $(X^T X)^{-1}$ analytically without computing an inverse explicitly.*

d. *[5 points]* For events $E_1, ..., E_m$ that are possibly dependent the "union bound" says that $P(\cup_i E_i) \leq \sum_i P(E_i)$. Use the union bound to come up with a threshold $\gamma > 0$ such that the probability that any of the coefficients $|\widehat{\beta}_j - \beta_j^*|$ exceed γ is less than δ.

e. *[7 points]* Fix $\mu > 0$. Assume each x_i is generated as above, and assume $\beta_i^* = \mu$ for $i \leq d/2$ and $\beta_i^* = 0$ for $i > d/2$. Using the threshold γ derived in part 3, find a sufficient condition for how big n needs to be to ensure that $\{j \in \{1, \ldots, d\} : |\widehat{\beta}_j| > \gamma\} = \{j \in \{1, \ldots, d\} : \beta_j^* \neq 0\}$ with probability at least $1 - \delta$.

f. *[6 points]* Is the union bound tight? Let $Z_1, \cdots, Z_d$ be i.i.d. random variables that follow $\mathcal{N}(0, 1)$. Show that there exist absolute constants c, C such that $c\sqrt{\log(d)} \leq \mathbb{E}[\max_{i=1,\cdots,d} Z_i] \leq C\sqrt{\log(d)}$. *Hint: Use the bounds on $\int_{x=t}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$ given in the notes. For the lower bound use $E[Y] \geq t\mathbb{P}(Y \geq t)$. For the upper bound use the fact that $\mathbb{E}[Y] \leq \mathbb{E}[|Y|] = \int_{t=0}^{\infty} \mathbb{P}(|Y| \geq t) dt$.*

Solution:

a. Since our observations have additive Gaussian noise, the error analysis shows that $\beta_{MLE} = \widehat{\beta} \sim N(\beta^*, (X^T X)^{-1})$ and hence the $\widehat{\beta}_j \sim N(\beta_j^*, (X^T X)^{-1}_{j,j})$.

b. Suppose that $\beta^* = 0$. This implies $\beta_j^* = 0$ for all j. By part (a), $\widehat{\beta}_j \sim N(0, (X^T X)^{-1}_{j,j}$ and thus

$$\frac{\widehat{\beta}_j}{\sqrt{(X^T X)^{-1}_{j,j}}} \sim N(0, 1).$$

From Proposition 2 from the notes on confidence intervals, we have that

$$\begin{aligned} P\left(|\widehat{\beta}_j| > \sqrt{2(X^TX)^{-1}_{j,j}\log(2/\delta)}\right) &= P\left(|\frac{|\widehat{\beta}_j|}{\sqrt{(X^TX)^{-1}_{j,j}}} > \sqrt{2\log(2/\delta)}\right) \\ &\leq 2\min\{1, 1/\sqrt{2\pi}, 1/t\}e^{-2\log(2/\delta}/2 \\ &= 2\min\{1, 1/\sqrt{2\pi}, 1/t\}\delta/2 \\ &< \delta. \end{aligned}$$

Therefore, we find that

$$P\left(|\widehat{\beta}_j| \leq \sqrt{2(X^TX)^{-1}_{j,j}\log(2/\delta)}\right) \geq 1 - \delta.$$

However, it is not true that this will hold for all j simultaneously. Without further knowledge of the covariances, we cannot guarantee that this will hold.

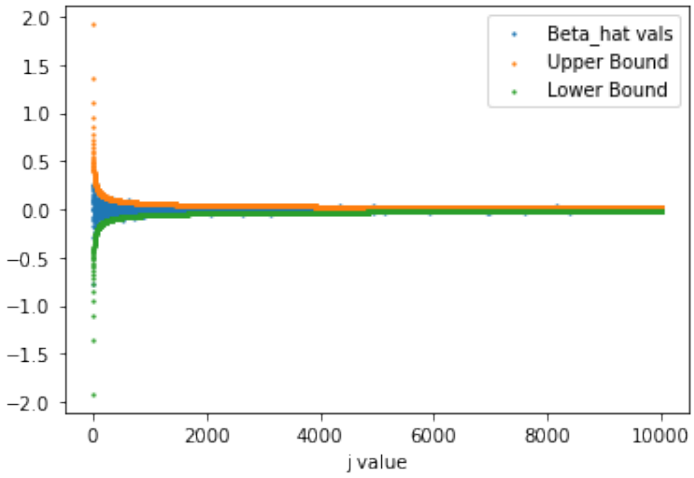
c. Using the definition $x_i = \sqrt{(i \mod d) + 1} \cdot e_i$, a calculation shows that

$$X^TX = 2 \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ & & \ddots & \\ & & & d. \end{pmatrix}$$

Therefore,

$$(X^TX)^{-1} = .5 \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1/2 & \cdots & 0 \\ & & \ddots & \\ & & & 1/d. \end{pmatrix}.$$

Here we see the plotted $\widehat{\beta}_j$ as well as the upper and lower bounds of the confidence interval.



We see that 62 out of 10000 of the $\widehat{\beta}_j$'s were outside of our confidence interval. Below is the code that was used to generate the above figure.

```
n = 20000
d = 10000
X = np.zeros((n,d))
for j in range(n):
    X[j,j%d] = math.sqrt((j%d)+1)

Y = np.random.normal(0,1,n)
inv = np.zeros((d,d))
for j in range(d):
    inv[j,j] = 1/(j+1)   #explicitly compute (X^TX)^{-1}
inv = 0.5 * inv
beta = inv @ (X.transpose() @ Y)
```

```
jvals = range(1,d+1)
lower = np.zeros(d)
upper = np.zeros(d)
for j in range(d):
    lower[j] = -math.sqrt(inv[j,j]*2*math.log(2/.05))
    upper[j] = math.sqrt(inv[j,j]*2*math.log(2/.05))

out_of_range=0
for j in range(d):
    if beta[j] > upper[j] or beta[j] < lower[j]:
        out_of_range+=1
print('The number of beta j that are out of given interval is ' + str(out_of_range))
```

```
The number of beta j that are out of given interval is 62
```

```
plt.scatter(jvals,beta, s=1.6, label = 'Beta_hat vals')
plt.scatter(jvals, upper, s=1.6, label='Upper Bound')
plt.scatter(jvals, lower, s=1.6, label='Lower Bound')
plt.xlabel('j value'); plt.legend()
```

d. Let $\delta > 0$ and let $\gamma_j = \sqrt{2(X^TX)^{-1}_{j,j}\log(2d/\delta)}$. By Proposition 2 from the Confidence Interval notes, and a similar calculation as in part (b), we find that

$$P(|\widehat{\beta}_j - \beta^*_j| > \gamma_j) < \delta/d.$$

Letting $\gamma = \max_j \gamma_j$, we apply the union bound and find that

$$P(\text{there exists } j \text{ such that } |\widehat{\beta}_j - \beta^*_j| > \gamma_j) \leq \sum_{j=1}^{d} P(|\widehat{\beta}_j - \beta^*_j| \geq \gamma_j) < \sum_{j=1}^{d} \delta/d = \delta.$$

e. Let $\beta^*_i = \mu$ for $i \leq d/2$ and $\beta^*_i = 0$ for $i > d/2$. Thus, the problem asks to find a sufficiently large n such that

$$P(|\widehat{\beta}_i| < \gamma \text{ for some } i \leq d/2 \text{ or } |\widehat{\beta}_i| > \gamma \text{ for some } i > d/2) < \delta.$$

By the union bound, we may instead bound

$$\sum_{i=1}^{d/2} P(\widehat{\beta}_j| < \gamma) + \sum_{i=d/2+1}^{d} P(|\widehat{\beta}_j| > \gamma) < \delta.$$

However, by part (d) applied to $i = d/2+1, \ldots, d$, we know that $\sum_{i=d/2+1}^{d} P(|\widehat{\beta}_j| > \gamma) < \delta/2$. Therefore, it suffices to show that each term in the first sum is bounded by δ/d.
Next, let n be a multiple of d such that

$$n > \frac{20,000\log(20,000/\delta)}{(\mu - \sqrt{20,000/\delta})^2}.$$

From a similar calculation as in part (c), we know that $(X^TX)^{-1} = (d/n)D$, where D is the diagonal matrix such that $D_{j,j} = 1/j$. Therefore, we have that

$$|\mu - \gamma| > \sqrt{\frac{20,000}{n}\log(20,000/\delta)}.$$

If $\mu > \gamma$, then

$$\begin{aligned} P(|\widehat{\beta}_j| < \gamma) &\leq P(\widehat{\beta}_j < \gamma) \\ &\leq P(|\mu - \widehat{\beta}_j| > |\mu - \gamma|) \\ &\leq P\left(|\mu - \widehat{\beta}_j| > \sqrt{2(X^T X)^{-1}_{j,j} \log(20,000/\delta)}\right) \\ &< \delta/d. \end{aligned}$$

Therefore, $\sum_{i=1}^{d/2} P(|\widehat{\beta}_j| < \gamma) < \sum_{i=1}^{d/2} \delta/d = \delta/2$. This proves the statement.

f. The union bound is not tight. It is possible that the events are disjoint in which equality would hold. Let $Z_1, \ldots, Z_d$ be i.i.d random variables that follow $N(0,1)$ and let $Y = \max_{j=1}^{d}\{Z_1, \ldots, Z_d\}$. We have that

$$\begin{aligned} E[Y] \geq tP(Y \geq t) & \\ &= (1 - P(Z_1 \leq t)^d) \\ &= t(1 - (1 - P(Z_1 \geq t))^d). \end{aligned}$$

Letting $t = \sqrt{\log(d)}$, the lower bound from the Confidence Interval notes shows that

$$E[Y] \geq \sqrt{\log(d)}\left(1 - \left(1 - \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\log(d)}}{(\log(d)+1)\sqrt{d}}\right)^d\right).$$

However, one can show that

$$\lim_{d\to\infty} \left(1 - \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\log(d)}}{(\log(d)+1)\sqrt{d}}\right)^d = 0$$

and hence as $d \to \infty$

$$\sqrt{\log(d)}\left(1 - \left(1 - \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\log(d)}}{(\log(d)+1)\sqrt{d}}\right)^d\right) \to \sqrt{\log(d)}(1-0) = \sqrt{\log(d)}.$$

This gives the lower bound with $c = 1$. For the upper bound, we have

$$\begin{aligned} E[|Y|] &= \int_0^\infty P(|Y| \geq t)\ dt \\ &= \int_0^{\sqrt{2\log d}} P(|Y| \geq t)\ dt + \int_{\sqrt{\log d}}^\infty P(|Y| \geq t)\ dt \\ &\leq \sqrt{2\log d} + \int_{\sqrt{2\log d}}^\infty (1 - (1 - dP(Z_1 \geq t)))\ dt \\ &= \sqrt{2\log d} + d\int_{\sqrt{2logd}}^\infty P(|Z_1| \geq t)\ dt \\ &\leq \sqrt{2\log d} + d\int_{\sqrt{2logd}}^\infty 2e^{-t^2/2}\ dt \\ &= \sqrt{2\log d} + d \cdot 2e^{-2\log d/2} \\ &= \sqrt{2\log d} + 2. \end{aligned}$$

This proves the upper bound.