Homework #1

CSE 446/546: Machine Learning
Professor Kevin Jamieson and Professor Jamie Morgenstern
Due: 10/23/20 11:59 PM
90 A points, 50 B points

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

- Please provide succinct answers along with succinct reasoning for all your answers. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. In other words, all your explanations, tables, and figures for any particular part of a question must be grouped together.

- 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 point deductions. 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 seperately. 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.

Short Answer and "True or False" Conceptual questions

A.0 The answers to these questions should be answerable without referring to external materials.

- *[2 points]* In your own words, describe what bias and variance are? What is bias-variance tradeoff?

- *[2 points]* What happens to bias and variance when the model complexity increases/decreases?

- *[1 points]* True or False: The bias of a model increases as the amount of training data available increases.

- *[1 points]* True or False: The variance of a model decreases as the amount of training data available increases.

- *[1 points]* True or False: A learning algorithm will generalize better if we use less features to represent our data

- *[2 points]* To get better generalization, should we use the train set or the test set to tune our hyperparameters?

- *[1 points]* True or False: The training error of a function on the training set provides an overestimate of the true error of that function.

Maximum Likelihood Estimation (MLE)

A.1. You're a Reign FC fan, and the team is five games into its 2018 season. The number of goals scored by the team in each game so far are given below:

$$[2, 0, 1, 1, 2].$$

Let's call these scores $x_1, \ldots, x_5$. Based on your (assumed iid) data, you'd like to build a model to understand how many goals the Reign are likely to score in their next game. You decide to model the number of goals scored per game using a *Poisson distribution*. The Poisson distribution with parameter λ assigns every non-negative integer $x = 0, 1, 2, \ldots$ a probability given by

$$\text{Poi}(x|\lambda) = e^{-\lambda} \frac{\lambda^x}{x!}.$$

So, for example, if $\lambda = 1.5$, then the probability that the Reign score 2 goals in their next game is $e^{-1.5} \times \frac{1.5^2}{2!} \approx 0.25$. To check your understanding of the Poisson, make sure you have a sense of whether raising λ will mean more goals in general, or fewer.

  a. *[5 points]* Derive an expression for the maximum-likelihood estimate of the parameter λ governing the Poisson distribution, in terms of your goal counts $x_1, \ldots, x_5$. (Hint: remember that the log of the likelihood has the same maximum as the likelihood function itself.)

  b. *[5 points]* Suppose the team scores 4 goals in its sixth game. Derive the same expression for the estimate of the parameter λ as in the prior example, now using the 6 games $x_1, \ldots, x_5, x_6 = 4$.

  c. *[5 points]* Given the goal counts, please give numerical estimates of λ after 5 and 6 games.

A.2. *[10 points]* In World War 2, the Allies attempted to estimate the total number of tanks the Germans had manufactured by looking at the serial numbers of the German tanks they had destroyed. The idea was that if there were n total tanks with serial numbers $\{1, \ldots, n\}$ then its reasonable to expect the observed serial numbers of the destroyed tanks constituted a uniform random sample (without replacement) from this set. The exact maximum likelihood estimator for this so-called *German tank problem* is non-trivial and quite challenging to work out (try it!). For our homework, we will consider a much easier problem with a similar flavor.
Let $x_1, \ldots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. What is the Maximum likelihood estimate for θ?

Overfitting

A.3. Suppose we have N labeled samples $S = \{(x_i, y_i)\}_{i=1}^N$ drawn i.i.d. from an underlying distribution $\mathcal{D}$. Suppose we decide to break this set into a set S_{train} of size N_{train} and a set S_{test} of size N_{test} samples for our training and test set, so $N = N_{\text{train}} + N_{\text{test}}$, and $S = S_{\text{train}} \cup S_{\text{test}}$. Recall the definition of the true least squares error of f:
$$\epsilon(f) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[(f(x) - y)^2],$$
where the subscript $(x, y) \sim \mathcal{D}$ makes clear that our input-output pairs are sampled according to $\mathcal{D}$. Our training and test losses are defined as:

$$\widehat{\epsilon}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{(x,y) \in S_{\text{train}}} (f(x) - y)^2$$

$$\widehat{\epsilon}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{(x,y) \in S_{\text{test}}} (f(x) - y)^2$$

We then train our algorithm (for example, using linear least squares regression) using the training set to obtain $\widehat{f}$.

  a. *[3 points]* (bias: the test error) For all fixed f (before we've seen any data) show that

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)] = \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)] = \epsilon(f).$$

  Use a similar line of reasoning to show that the test error is an unbiased estimate of our true error for $\hat{f}$. Specifically, show that:
$$\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})] = \epsilon(\widehat{f})$$

b. *[4 points]* (bias: the train/dev error) Is the above equation true (in general) with regards to the training loss? Specifically, does $\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f})]$ equal $\mathbb{E}_{\text{train}}[\epsilon(\widehat{f})]$? If so, why? If not, give a clear argument as to where your previous argument breaks down.

c. *[8 points]* Let $\mathcal{F} = (f_1, f_2, \dots)$ be a collection of functions and let $\widehat{f}_{\text{train}}$ minimize the training error such that $\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}}) \leq \widehat{\epsilon}_{\text{train}}(f)$ for all $f \in \mathcal{F}$. Show that

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}})] \leq \mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})].$$

(Hint: note that

$$\mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(f)\mathbf{1}\{\widehat{f}_{\text{train}} = f\}]$$

$$= \sum_{f \in \mathcal{F}} \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)]\mathbb{E}_{\text{train}}[\mathbf{1}\{\widehat{f}_{\text{train}} = f\}] = \sum_{f \in \mathcal{F}} \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f)$$

where the second equality follows from the independence between the train and test set.)

Bias-Variance tradeoff

B.1. For $i = 1, \dots, n$ let $x_i = i/n$ and $y_i = f(x_i) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ for some unknown f we wish to approximate at values $\{x_i\}_{i=1}^n$. We will approximate f with a step function estimator. For some $m \leq n$ such that n/m is an integer define the estimator

$$\widehat{f}_m(x) = \sum_{j=1}^{n/m} c_j \mathbf{1}\{x \in \left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]\} \quad \text{where} \quad c_j = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} y_i.$$

Note that this estimator just partitions $\{1, \dots, n\}$ into intervals $\{1, \dots, m\}, \{m+1, \dots, 2m\}, \dots, \{n-m+1, \dots, n\}$ and predicts the average of the observations within each interval (see Figure 1).
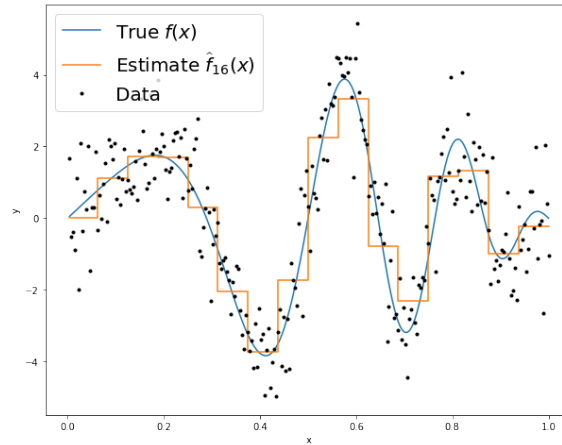

Figure 1: Step function estimator with $n = 256$, $m = 16$, and $\sigma^2 = 1$.

By the bias-variance decomposition at some x_i we have

$$\mathbb{E}\left[(\widehat{f}_m(x_i) - f(x_i))^2\right] = \underbrace{(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2}_{\text{Bias}^2(x_i)} + \underbrace{\mathbb{E}\left[(\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2\right]}_{\text{Variance}(x_i)}$$

3

a. *[5 points]* Intuitively, how do you expect the bias and variance to behave for small values of m? What about large values of m?

b. *[5 points]* If we define $\bar{f}^{(j)} = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} f(x_i)$ and the *average bias-squared* as $\frac{1}{n} \sum_{i=1}^{n} (\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2$, show that

$$\frac{1}{n} \sum_{i=1}^{n} (\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\bar{f}^{(j)} - f(x_i))^2$$

c. *[5 points]* If we define the *average variance* as $\mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} (\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2\right]$, show (both equalities)

$$\mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} (\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2\right] = \frac{1}{n} \sum_{j=1}^{n/m} m\mathbb{E}[(c_j - \bar{f}^{(j)})^2] = \frac{\sigma^2}{m}$$

d. *[15 points]* Let $n = 256$, $\sigma^2 = 1$, and $f(x) = 4\sin(\pi x)\cos(6\pi x^2)$. For values of $m = 1, 2, 4, 8, 16, 32$ plot the average empirical error $\frac{1}{n} \sum_{i=1}^{n} (\widehat{f}_m(x_i) - f(x_i))^2$ using randomly drawn data as a function of m on the x-axis. On the same plot, using parts b and c of above, plot the *average bias-squared*, the *average variance*, and their sum (the average error). Thus, there should be 4 lines on your plot, each described in a legend.

e. *[5 points]* By the Mean-Value theorem we have that

$$\min_{i=(j-1)m+1,\ldots,jm} f(x_i) \leq \bar{f}^{(j)} \leq \max_{i=(j-1)m+1,\ldots,jm} f(x_i)$$

. Suppose f is L-Lipschitz so that $|f(x_i) - f(x_j)| \leq \frac{L}{n}|i - j|$ for all $i, j \in \{1, \ldots, n\}$ for some $L > 0$. Show that the average bias-squared is $O(\frac{L^2 m^2}{n^2})$. Using the expression for average variance above, the total error behaves like $O(\frac{L^2 m^2}{n^2} + \frac{\sigma^2}{m})$. Minimize this expression with respect to m. Does this value of m, and the total error when you plug this value of m back in, behave in an intuitive way with respect to n, L, σ^2? That is, how does m scale with each of these parameters? It turns out that this simple estimator (with the optimized choice of m) obtains the best achievable error rate up to a universal constant in this setup for this class of L-Lipschitz functions (see Tsybakov's *Introduction to Nonparametric Estimation* for details).

This setup of each x_i deterministically placed at i/n is a good approximation for the more natural setting where each x_i is drawn uniformly at random from $[0, 1]$. In fact, one can redo this problem and obtain nearly identical conclusions, but the calculations are messier.

Polynomial Regression

Relevant Files[1]

- **polyreg.py**
- **linreg_closedform.py**

- test_polyreg_univariate.py
- test_polyreg_learningCurve.py
- data/polydata.dat

A.4.*[10 points]* Recall that polynomial regression learns a function $h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d$. In this case, d represents the polynomial's degree. We can equivalently write this in the form of a linear model

$$h_{\boldsymbol{\theta}}(x) = \theta_0 \phi_0(x) + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \ldots + \theta_d \phi_d(x) \ , \tag{1}$$

using the basis expansion that $\phi_j(x) = x^j$. Notice that, with this basis expansion, we obtain a linear model where the features are various powers of the single univariate x. We're still solving a linear regression problem,

---

[1]**Bold text** indicates files or functions that you will need to complete; you should not need to modify any of the other files.

but are fitting a polynomial function of the input.

Implement regularized polynomial regression in `polyreg.py`. You may implement it however you like, using gradient descent or a closed-form solution. However, I would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation of linear regression in `linreg_closedform.py` (you are welcome to build upon this implementation, but make CERTAIN you understand it, since you'll need to change several lines of it). You are also welcome to build upon your implementation from the previous assignment, but you must follow the API below. Note that all matrices are actually 2D numpy arrays in the implementation.

- `__init__(degree=1, regLambda=1E-8)` : constructor with arguments of d and λ
- `fit(X,Y)`: method to train the polynomial regression model
- `predict(X)`: method to use the trained polynomial regression model for prediction
- `polyfeatures(X, degree)`: expands the given $n \times 1$ matrix X into an $n \times d$ matrix of polynomial features of degree d. Note that the returned matrix will not include the zero-th power.

Note that the `polyfeatures(X, degree)` function maps the original univariate data into its higher order powers. Specifically, X will be an $n \times 1$ matrix ($X \in \mathbb{R}^{n \times 1}$) and this function will return the polynomial expansion of this data, a $n \times d$ matrix. Note that this function will **not** add in the zero-th order feature (i.e., $x_0 = 1$). You should add the x_0 feature separately, outside of this function, before training the model.

By not including the x_0 column in the matrix `polyfeatures()`, this allows the `polyfeatures` function to be more general, so it could be applied to multi-variate data as well. (If it did add the x_0 feature, we'd end up with multiple columns of 1's for multivariate data.)

Also, notice that the resulting features will be badly scaled if we use them in raw form. For example, with a polynomial of degree $d = 8$ and $x = 20$, the basis expansion yields $x^1 = 20$ while $x^8 = 2.56 \times 10^{10}$ – an absolutely huge difference in range. Consequently, we will need to standardize the data before solving linear regression. Standardize the data in `fit()` after you perform the polynomial feature expansion. You'll need to apply the same standardization transformation in `predict()` before you apply it to new data.
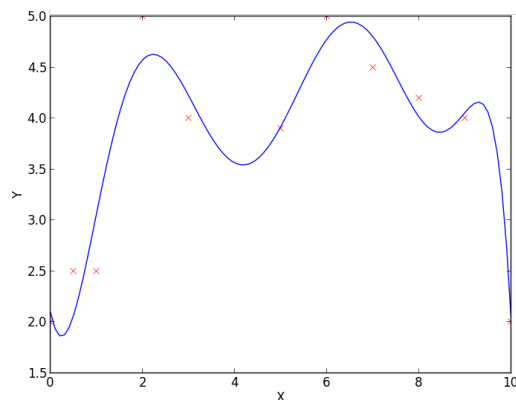

Figure 2: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

Run `test_polyreg_univariate.py` to test your implementation, which will plot the learned function. In this case, the script fits a polynomial of degree $d = 8$ with no regularization $\lambda = 0$. From the plot, we see that the function fits the data well, but will not generalize well to new data points. Try increasing the amount of regularization, and examine the resulting effect on the function.

A.5. *[10 points]* In this problem we will examine the bias-variance tradeoff through learning curves. Learning curves provide a valuable mechanism for evaluating the bias-variance tradeoff. Implement the `learningCurve()` function in `polyreg.py` to compute the learning curves for a given training/test set. The `learningCurve(Xtrain, ytrain, Xtest, ytest, degree, regLambda)` function should take in the training data (`Xtrain`, `ytrain`), the testing data (`Xtest`, `ytest`), and values for the polynomial degree d and regularization parameter λ.

The function should return two arrays, `errorTrain` (the array of training errors) and `errorTest` (the array of testing errors). The ith index (start from 0) of each array should return the training error (or testing error) for learning with $i + 1$ training instances. Note that the 0th index actually won't matter, since we typically start displaying the learning curves with two or more instances.

When computing the learning curves, you should learn on `Xtrain[0:i]` for $i = 1, \ldots,$ numInstances(`Xtrain`) $+ 1$, each time computing the testing error over the **entire** test set. There is no need to shuffle the training data, or to average the error over multiple trials – just produce the learning curves for the given training/testing sets

5

with the instances in their given order. Recall that the error for regression problems is given by

$$\frac{1}{n} \sum_{i=1}^{n} (h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 \ . \tag{2}$$

Once the function is written to compute the learning curves, run the `test_polyreg_learningCurve.py` script to plot the learning curves for various values of λ and d. You should see plots similar to the following:
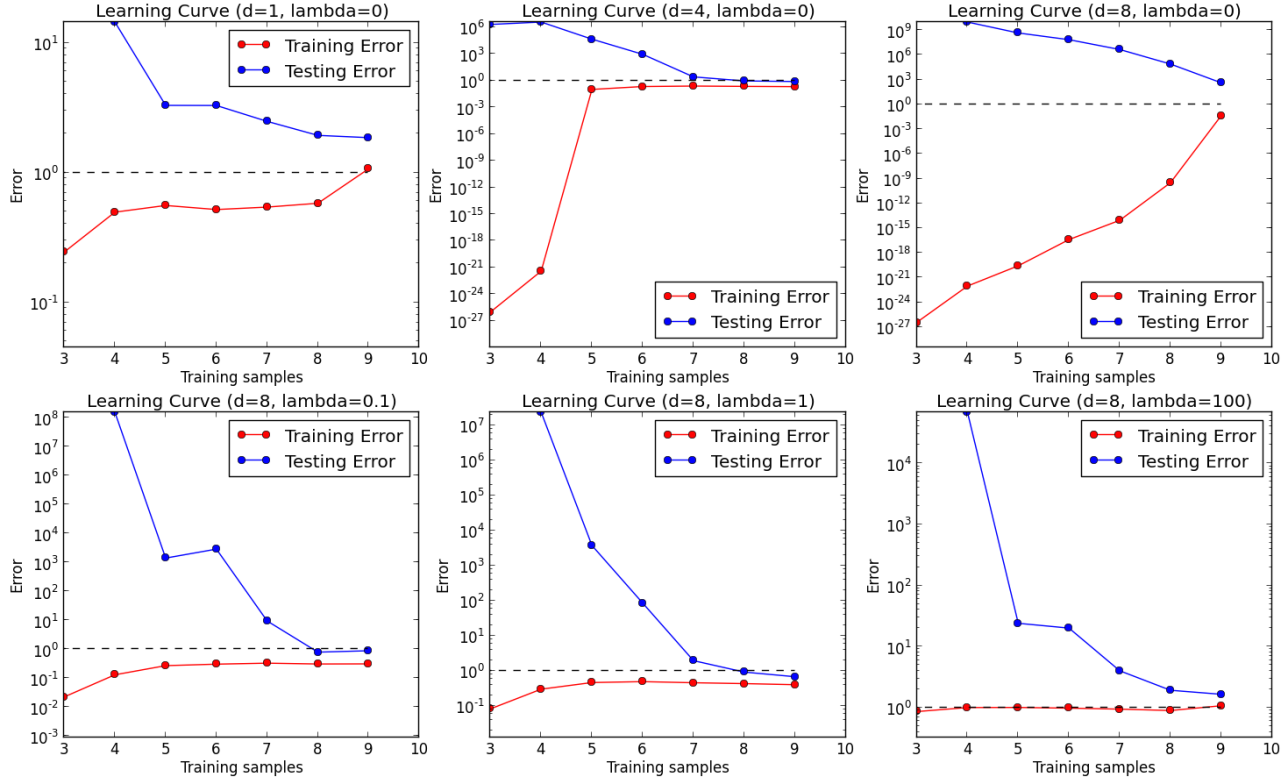
Notice the following:

- The y-axis is using a log-scale and the ranges of the y-scale are all different for the plots. The dashed black line indicates the $y = 1$ line as a point of reference between the plots.

- The plot of the unregularized model with $d = 1$ shows poor training error, indicating a high bias (i.e., it is a standard univariate linear regression fit).

- The plot of the unregularized model ($\lambda = 0$) with $d = 8$ shows that the training error is low, but that the testing error is high. There is a huge gap between the training and testing errors caused by the model overfitting the training data, indicating a high variance problem.

- As the regularization parameter increases (e.g., $\lambda = 1$) with $d = 8$, we see that the gap between the training and testing error narrows, with both the training and testing errors converging to a low value. We can see that the model fits the data well and generalizes well, and therefore does not have either a high bias or a high variance problem. Effectively, it has a good tradeoff between bias and variance.

- Once the regularization parameter is too high ($\lambda = 100$), we see that the training and testing errors are once again high, indicating a poor fit. Effectively, there is too much regularization, resulting in high bias.

Make absolutely certain that you understand these observations, and how they relate to the learning curve plots. In practice, we can choose the value for λ via cross-validation to achieve the best bias-variance tradeoff.

Ridge Regression on MNIST

A.6. In this problem we will implement a regularized least squares classifier for the MNIST data set. The task is to classify handwritten images of numbers between 0 to 9.

You are **NOT** allowed to use any of the prebuilt classifiers in `sklearn`. Feel free to use any method from `numpy` or `scipy`. Remember: if you are inverting a matrix in your code, you are probably doing something wrong (Hint: look at `scipy.linalg.solve`).

Get the data from `https://pypi.python.org/pypi/python-mnist`.
Load the data as follows:

```
from mnist import MNIST

def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
```

Each example has features $x_i \in \mathbb{R}^d$ (with $d = 28 * 28 = 784$) and label $z_j \in \{0, \ldots, 9\}$. You can visualize a single example x_i with `imshow` after reshaping it to its original 28×28 image shape (and noting that the label z_j is accurate). We wish to learn a predictor $\widehat{f}$ that takes as input a vector in $\mathbb{R}^d$ and outputs an index in $\{0, \ldots, 9\}$. We define our training and testing classification error on a predictor f as

$$\widehat{\epsilon}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{(x,z) \in \text{Training Set}} \mathbf{1}\{f(x) \neq z\}$$

$$\widehat{\epsilon}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{(x,z) \in \text{Test Set}} \mathbf{1}\{f(x) \neq z\}$$

We will use one-hot encoding of the labels, i.e. of (x, z) the original label $z \in \{0, \ldots, 9\}$ is mapped to the standard basis vector e_z where e_z is a vector of all zeros except for a 1 in the zth position. We adopt the notation where we have n data points in our training objective with features $x_i \in \mathbb{R}^d$ and label one-hot encoded as $y_i \in \{0, 1\}^k$ where in this case $k = 10$ since there are 10 digits.

a. *[10 points]* In this problem we will choose a linear classifier to minimize the regularized least squares objective:

$$\widehat{W} = \text{argmin}_{W \in \mathbb{R}^{d \times k}} \sum_{i=0}^n \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2$$

Note that $\|W\|_F$ corresponds to the Frobenius norm of W, i.e. $\|W\|_F^2 = \sum_{i=1}^d \sum_{j=1}^k W_{i,j}^2$. To classify a point x_i we will use the rule $\arg\max_{j=0,\ldots,9} e_j^T \widehat{W}^T x_i$. Note that if $W = \begin{bmatrix} w_1 & \ldots & w_k \end{bmatrix}$ then

$$\sum_{i=0}^n \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 = \sum_{j=0}^k \left[\sum_{i=1}^n (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda \|W e_j\|^2 \right]$$

$$= \sum_{j=0}^k \left[\sum_{i=1}^n (w_j^T x_i - e_j^T y_i)^2 + \lambda \|w_j\|^2 \right]$$

$$= \sum_{j=0}^k \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right]$$

where $X = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}^\top \in \mathbb{R}^{n \times d}$ and $Y = \begin{bmatrix} y_1 & \ldots & y_n \end{bmatrix}^\top \in \mathbb{R}^{n \times k}$. Show that

$$\widehat{W} = (X^T X + \lambda I)^{-1} X^T Y$$

b. *[10 points]*

- Code up a function `train` that takes as input $X \in \mathbb{R}^{n \times d}$, $Y \in \{0,1\}^{n \times k}$, $\lambda > 0$ and returns $\widehat{W}$.
- Code up a function `predict` that takes as input $W \in \mathbb{R}^{d \times k}$, $X' \in \mathbb{R}^{m \times d}$ and returns an m-length vector with the ith entry equal to $\arg\max_{j=0,\dots,9} e_j^T W^T x_i'$ where x_i' is a column vector representing the ith example from X'.
- Train $\widehat{W}$ on the MNIST training data with $\lambda = 10^{-4}$ and make label predictions on the test data. **What is the training and testing error?** Note that they should both be about 15%.

B.2

a. *[10 points]* We just fit a classifier that was linear in the pixel intensities to the MNIST data. For classification of digits the raw pixel values are very, very bad features: it's pretty hard to separate digits with linear functions in pixel space. The standard solution to this is to come up with some transform $h : \mathbb{R}^d \to \mathbb{R}^p$ of the original pixel values such that the transformed points are (more easily) linearly separable. In this problem, you'll use the feature transform:

$$h(x) = \cos(Gx + b).$$

where $G \in \mathbb{R}^{p \times d}$, $b \in \mathbb{R}^p$, and the cosine function is applied elementwise. We'll choose G to be a *random* matrix, with each entry sampled i.i.d. from a Gaussian with mean $\mu = 0$ and variance $\sigma^2 = 0.1$, and b to be a random vector sampled i.i.d. from the uniform distribution on $[0, 2\pi]$. The big question is: *how do we choose p?* Using cross-validation, of course!

Randomly partition your training set into proportions 80/20 to use as a new training set and validation set, respectively. Using the `train` function you wrote above, train a $\widehat{W}^p$ for different values of p and plot the classification training error and validation error on a single plot with p on the x-axis. Be careful, your computer may run out of memory and slow to a crawl if p is too large ($p \leq 6000$ should fit into 4 GB of memory that is a minimum for most computers, but if you're having trouble you can set p in the several hundreds). You can use the same value of λ as above but feel free to study the effect of using different values of λ and σ^2 for fun.

b. *[5 points]* Instead of reporting just the test error, which is an unbiased estimate of the *true* error, we would like to report a *confidence interval* around the test error that contains the true error.

Lemma 1. *(Hoeffding's inequality) Fix $\delta \in (0, 1)$. If for all $i = 1, \dots, m$ we have that X_i are i.i.d. random variables with $X_i \in [a, b]$ and $\mathbb{E}[X_i] = \mu$ then*

$$\mathbb{P}\left(\left| \left(\frac{1}{m} \sum_{i=1}^{m} X_i \right) - \mu \right| \geq \sqrt{\frac{(b-a)^2 \log(2/\delta)}{2m}} \right) \leq \delta$$

We will use the above equation to construct a confidence interval around the true classification error $\epsilon(\widehat{f}) = \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})]$ since the test error $\widehat{\epsilon}_{\text{test}}(\widehat{f})$ is just the average of indicator variables taking values in $\{0, 1\}$ corresponding to the ith test example being classified correctly or not, respectively, where an error happens with probability $\mu = \epsilon(\widehat{f}) = \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})]$, the *true* classification error.

Let $\widehat{p}$ be the value of p that approximately minimizes the validation error on the plot you just made and use $\widehat{f}(x) = \arg\max_j x^T \widehat{W}^{\widehat{p}} e_j$ to compute the classification test error $\widehat{\epsilon}_{\text{test}}(\widehat{f})$. Use Hoeffding's inequality, of above, to compute a confidence interval that contains $\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})]$ (i.e., the *true* error) with probability at least 0.95 (i.e., $\delta = 0.05$). Report $\widehat{\epsilon}_{\text{test}}(\widehat{f})$ and the confidence interval.