Homework #1

CSE 546: Machine Learning
Cassia Cai
October 19, 2022

Collaborators: Apoorva Kalaskar, Madeleine Grunde-McLaughlin, and Aaleyah Lewis

Short Answer and "True or False" Conceptual questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

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

The bias is the expected difference between our model estimates and our ground truth (our correct values). The variance is the expected value fo the square difference between our model estimates and the expected value of the estimate. For example, if we use different parts of the data set (with the same distribution), the variance tells us how much the model will change. Bias and variance has an inverse relationship. The bias-variance tradeoff is this inverse relationship. So when a model has a lower bias, it tends to have higher variance. When a model has a higher bias, it tends to have lower variance.

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

Model complexity ↑, variance ↑ and bias ↓

Model complexity ↓, variance ↓ and bias ↑

c. *[2 points]* True or False: Suppose you're given a fixed learning algorithm. If you collect more training data from the same distribution, the variance of your predictor increases.

This is false. Given a fixed learning algorithm, if we collect more training data from the same distribution, we should expect the variance of our predictor to decrease.

d. *[2 points]* Suppose that we are given train, validation, and test sets. Which of these sets should be used for hyperparameter tuning? Explain your choice and detail a procedure for hyperparameter tuning.

Given train, validation, and test sets, we should use both the train and validation set for hyperparameter tuning. We use the train set to train our model and use our validation set to tune our hyperparameter. We use the train set to optimize our model's parameter values and the validation set to optimize our model's architecture. For ridge regression, we use the validation set to optimize our lambda value.

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

This is false. The training error of a function on the training set provides an underestimate of the true error of that function.

Maximum Likelihood Estimation (MLE)

A2. You're the Reign FC manager, and the team is five games into its 2021 season. The numbers of goals scored by the team in each game so far are given below:

$$[2, 4, 6, 0, 1].$$

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*. Recall that the Poisson distribution with parameter λ assigns every non-negative integer $x = 0, 1, 2, \ldots$ a probability given by

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

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

$$L(\lambda) = \prod_{i=1}^{n} \text{Poisson}(x_i|\lambda) = \prod_{i=1}^{n} e^{-\lambda}\frac{\lambda_i^x}{x_i!} \tag{1}$$

From the hint: the log of the likelihood has the same maximiser as the likelihood function itself.

$$\log(L(\lambda)) = \log(\prod_{i=1}^{n} \text{Poisson}(x_i|\lambda)) = \sum_{i=1}^{n} \log(e^{-\lambda}\frac{\lambda_i^x}{x_i!}) = \sum_{i=1}^{n}(\log(e^{-\lambda}) + \log(\lambda^{x_i}) - \log(x_i!)) \tag{2}$$

$$\log(L(\lambda)) = \log(\prod_{i=1}^{n} \text{Poisson}(x_i|\lambda)) = \sum_{i=1}^{n}(-\lambda + x_i\log(\lambda) - \log(x_i!)) \tag{3}$$

We set the derivative of Eq. 3 to 0.

$$\frac{\mathrm{d}\log(L(\lambda))}{\mathrm{dx}} = 0 = \sum_{i=1}^{n}(-1 + \frac{x_i}{\lambda}) \Rightarrow \sum_{i=1}^{n} 1 = \sum_{i=1}^{n}\frac{x_i}{\lambda} \tag{4}$$

$$\lambda = \sum_{i=1}^{n}\frac{x_i}{n} \tag{5}$$

b. *[2 points]* Give a numerical estimate of λ after the first five games. Given this λ, what is the probability that the Reign score exactly 6 goals in their next game?

$$\lambda = \sum_{i=1}^{5}\frac{x_i}{5} = \frac{1}{5}(2+4+6+0+1) = \frac{13}{5} \tag{6}$$

$$\text{Poi}(6|\frac{13}{5}) = e^{-\frac{13}{5}}\frac{\frac{13}{5}^6}{6!} = 0.032 \tag{7}$$

c. *[2 points]* Suppose the Reign score 8 goals in their 6th game. Give an updated numerical estimate of λ after six games and compute the probability that the Reign score 6 goals in their 7th game.

$$\lambda = \sum_{i=1}^{6}\frac{x_i}{6} = \frac{1}{6}(2+4+6+0+1+8) = \frac{21}{6} \tag{8}$$

$$\text{Poi}(6|\frac{21}{6}) = e^{-\frac{21}{6}}\frac{\frac{21}{6}^6}{6!} = 0.077 \tag{9}$$

Polynomial Regression

Relevant Files[1]:

- **polyreg.py**
- `linreg_closedform.py`
- `plot_polyreg_univariate.py`
- `plot_polyreg_learningCurve.py`

A3. Recall that polynomial regression learns a function $h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d$, where d represents the polynomial's highest degree. We can equivalently write this in the form of a linear model with d features

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \ldots + \theta_d \phi_d(x) \ , \tag{10}$$

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, but are fitting a polynomial function of the input.

a. *[8 points]* Implement regularized polynomial regression in `polyreg.py`.

```
def __init__(self, degree = int(1), reg_lambda = float(1e-8)):
    self.degree = int(degree)
    self.reg_lambda = float(reg_lambda)
    self.weight = np.ndarray(None)
    self.mean = None
    self.std = None

def polyfeatures(X = np.ndarray, degree = int):
    poly_features_X = X
    to_expand = [X.flatten()]
    for i in range(1, degree):
        to_expand.append((X**(i+1)).flatten())
    poly_features_X = np.array(to_expand).T
    return poly_features_X

def fit(self, X = np.ndarray, y = np.ndarray):
    X = self.polyfeatures(X, self.degree)
    self.mean = np.mean(X, axis = 0)
    self.std = np.std(X, axis = 0)
    X = (X - self.mean) / self.std
    X = np.c_[np.ones([len(X), 1]), X]
    n, d = X.shape
    reg_matrix = self.reg_lambda * np.eye(d)
    reg_matrix[0, 0] = 0
    self.weight = np.linalg.pinv(X.T.dot(X) + reg_matrix).dot(X.T).dot(y)

def predict(self, X = np.ndarray):
    X = self.polyfeatures(X, self.degree)
    X = (X - self.mean) / self.std
    X = np.c_[np.ones([len(X), 1]), X]
    return X.dot(self.weight)
```

b. *[2 points]* Run `plot_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 in 1-2 sentences, describe the resulting effect on the function (you may also provide an additional plot to support your analysis).

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

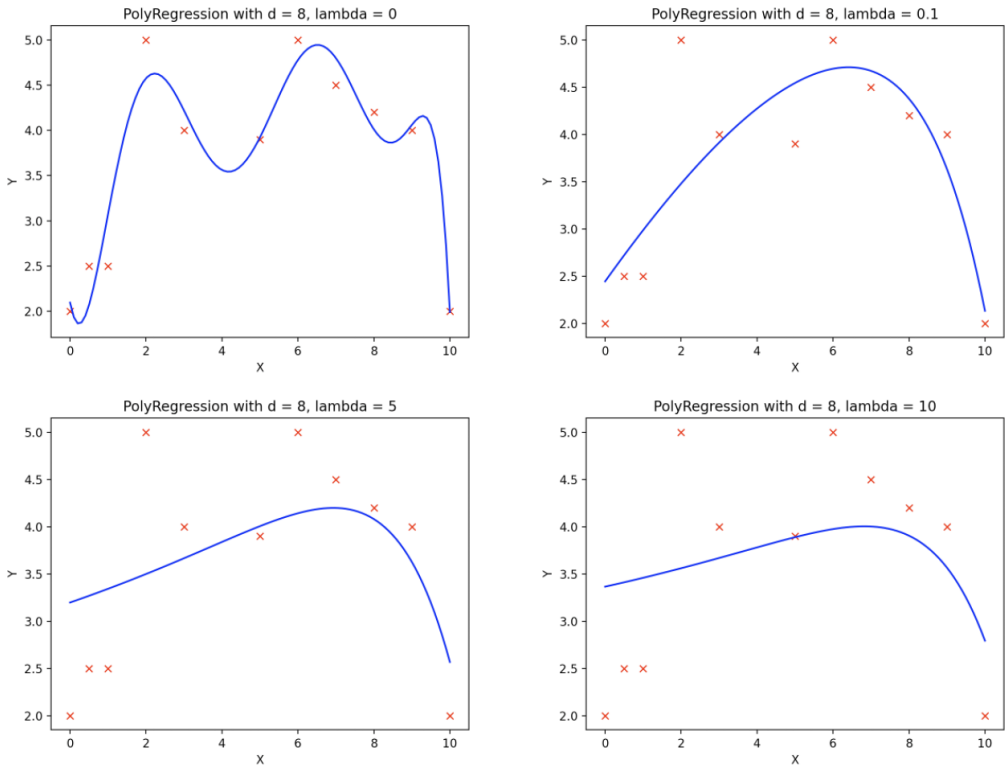

Figure 1: Plots before and after increase in regularization

As we increase the amount of regularization, we are decreasing the sum of squares in linear regression. Our bias increases and the model variance decreases. If the amount of regularization is too large, then our model is too simple, and we risk under-fitting our data.

A4. *[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 i^{th} 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 0^{th} 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`), 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 with the instances in their given order. Recall that the error for regression problems is given by

$$\frac{1}{n}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 \ . \tag{11}$$

Once the function is written to compute the learning curves, run the `plot_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 (almost) unregularized model ($\lambda = 10^{-6}$) 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.

Submit plots for the same values of d and λ shown here. 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.

```
def learningCurve(Xtrain, Ytrain, Xtest, Ytest, reg_lambda, degree):
    n = len(Xtrain)

    errorTrain = np.zeros(n)
    errorTest = np.zeros(n)

    model = PolynomialRegression(degree=degree, reg_lambda=reg_lambda)

    for i in range(1,n):
        train_features = Xtrain[0:(i+1)]
        train_labels = Ytrain[0:(i+1)]

        model.fit(train_features, train_labels)
```

```
        train_pred = model.predict(train_features)
        test_pred = model.predict(Xtest)

        errorTrain[i] = mean_squared_error(train_pred, train_labels)
        errorTest[i] = mean_squared_error(test_pred, Ytest)
    return errorTrain, errorTest
```

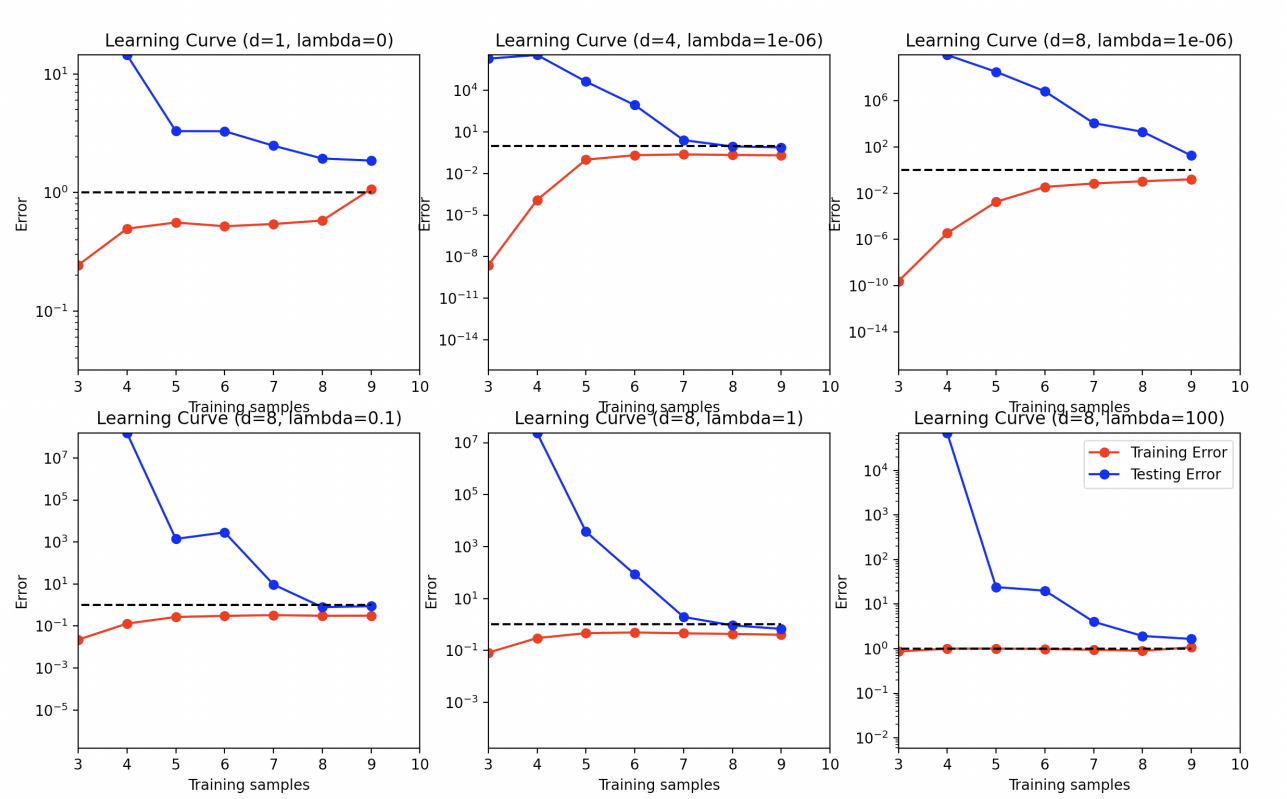

Figure 2: Plots (or single plot with many subplots) of learning curves for $(d, \lambda) \in \{(1, 0), (4, 10^{-6}), (8, 10^{-6}), (8, 0.1), (8, 1), (8, 100)\}$

Ridge Regression on MNIST

Relevant Files[2]:

- **ridge_regression.py**

A5. 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.

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). Checkout figure **??** for some sample images. 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: for each observation (x, z), the original label $z \in \{0, \ldots, 9\}$ is mapped to the standard basis vector e_{z+1} where e_i is a vector of size k containing all zeros except for a 1 in the i^{th} position (positions in these vectors are indexed starting at one, hence the $z + 1$ offset for the digit labels). 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 $y_i \in \{0, 1\}^k$. Here, $k = 10$ since there are 10 digits.

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

$$\widehat{W} = \text{argmin}_{W \in \mathbb{R}^{d \times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2$$

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 $\arg\max_{j=0,\ldots,9} e_{j+1}^T \widehat{W}^T x_i$. Note that if $W = \begin{bmatrix} w_1 & \ldots & w_k \end{bmatrix}$ then

$$\begin{aligned}
\sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 &= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda \|W e_j\|^2 \right] \\
&= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (w_j^T x_i - e_j^T y_i)^2 + \lambda \|w_j\|^2 \right] \\
&= \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right]
\end{aligned}$$

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

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

We should express $\widehat{W}$, which is a k by d matrix by stacking our single decompositions, where each row i of $\widehat{W}$ is $(X^T X + \lambda I)^{-1} X^T y_i$. Let us start from:

$$\sum_{j=1}^{k} \left[\|X w_j - y_j\|^2 + \lambda \|w_j\|^2 \right] \tag{12}$$

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

We take the partial derivative with respect to w.

$$0 = \frac{\partial}{\partial w_j}\left[\sum_{j=1}^{k} \|Xw_j - y_j\|^2 + \lambda\|w_j\|^2\right] = \frac{\partial}{\partial w_j}\left[(Xw_j - y_j)^T(Xw_j - y_j) + \lambda w^T w\right] \tag{13}$$

$$= (2X^TXw_j - 2X^Ty_j + 2\lambda w_j) = 2(X^TXw_j - X^Ty_j + \lambda I w_j) = 0 \tag{14}$$

$$(X^TXw_j - X^Ty_j + \lambda w_j) = 0 \tag{15}$$

$$(X^TXw_j + \lambda I w_j) = (X^Ty_j) \tag{16}$$

$$(X^TX + \lambda I)w_j = (X^Ty_j) \tag{17}$$

$$w_j = \frac{(X^Ty_j)}{X^TX + \lambda I} = (X^TX + \lambda I)^{-1}(X^Ty_j) \tag{18}$$

$$w_j = (X^TX + \lambda I)^{-1}(X^Ty_j) \tag{19}$$

Here, we see that by stacking our w_j elements, we can build $\widehat{W}$. We have therefore shown that:

$$\widehat{W} = \begin{bmatrix} (X^TX + \lambda I)^{-1}(X^Ty_1) \\ . \\ . \\ . \\ (X^TX + \lambda I)^{-1}(X^Ty_k) \end{bmatrix} = (X^TX + \lambda I)^{-1}X^TY \tag{20}$$

b. *[9 points]*

- Implement 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 $\widehat{W} \in \mathbb{R}^{d\times k}$.
- Implement a function `one_hot` that takes as input $Y \in \{0, ..., k-1\}^n$, and returns $Y \in \{0,1\}^{n\times k}$.
- Implement 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 $\arg\max_{j=0,...,9} e_j^T W^T x_i'$ where $x_i' \in \mathbb{R}^d$ is a column vector representing the ith example from X'.
- Using the functions you coded above, train a model to estimate $\widehat{W}$ on the MNIST training data with $\lambda = 10^{-4}$, and make label predictions on the test data. This behavior is implemented in the `main` function provided in a zip file.

```
def train(x: np.ndarray, y: np.ndarray, _lambda: float) -> np.ndarray:
    n_rows, n_cols = x.shape
    a = x.T @ x + _lambda*np.eye(n_cols)
    b = x.T @ y
    w_hat = np.linalg.solve(a, b)
    return w_hat

def predict(x: np.ndarray, w: np.ndarray) -> np.ndarray:
    pred = np.argmax(np.dot(x, w),axis=1)
    return pred

def one_hot(y: np.ndarray, num_classes: int) -> np.ndarray:
    arr = np.zeros([y.shape[0],num_classes])
    for i in range(y.shape[0]):
        arr[i][y[i]] = 1
    return arr

def main():
    (x_train, y_train), (x_test, y_test) = load_dataset("mnist")
```

```
y_train_one_hot = one_hot(y_train.reshape(-1), 10)
_lambda = 1e-4
w_hat = train(x_train, y_train_one_hot, _lambda)
y_train_pred = predict(x_train, w_hat)
y_test_pred = predict(x_test, w_hat)
```

c. *[1 point]* What are the training and testing errors of the classifier trained as above?

The training error is: 14.805 %. The testing error is 14.66 %.

Once you finish this problem question, you should have a very powerful classifier for handwritten digits! Curious to know how it compares to other models, including the almighty *Neural Networks*? Check out the **linear classifier (1-layer NN)** on the official MNIST leaderboard. (The model we just built is actually a 1-layer neural network: more on this soon!)

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework?

I spent about 10 hours on this homework.