

HW3 ML

Kumari Himansu Sinha

B01027707

1) LFD Exercise 4.3

Exercise 4.3

Deterministic noise depends on \mathcal{H} , as some models approximate f better than others.

- (a) Assume \mathcal{H} is fixed and we increase the complexity of f . Will deterministic noise in general go up or down? Is there a higher or lower tendency to overfit?
- (b) Assume f is fixed and we decrease the complexity of \mathcal{H} . Will deterministic noise in general go up or down? Is there a higher or lower tendency to overfit? *[Hint: There is a race between two factors that affect overfitting in opposite ways, but one wins.]*

Answer->

(a) For a given H , we will have 2 cases:

(I) If the best approximation from H is less complex than the initial target function, then when we increase the complexity of f , the deterministic - i.e noise generally should increase. Since, it will be harder for functions in ' H ' to fit the target function. There will be a higher tendency to overfit.

(II) When best approximation from ' H ' is more complex than the initial target function and when we increase the complexity of f , the deterministic noise in general may decrease first. It will be reducing the deterministic noise and there will be a lower tendency to overfit. But when the complexity of f exceeds the best function approximation from H and if we continue increase the complexity of f , we will increase the deterministic noise and thus increase the tendency of overfit. Because we made our model more complex.

⑤ For a given fixed f , we will have 2 cases:

case 1 \Rightarrow When best approximation from H is lower (less) complex than the target function, then when we decrease the complexity of H , we increase the deterministic noise. This will increase overfit.

case 2 \Rightarrow If the best approximation from H is more complex than the target function, when we decrease the complexity of H , we will decrease the deterministic noise thus decrease the tendency of overfit. Well, if we continue to decrease the complexity of H , passing the point where its complexity equal to f , we start to increase the deterministic noise again and thus increase overfit.

2) LFD Exercise 4.6.

Exercise 4.6

We have seen both the hard-order constraint and the soft-order constraint. Which do you expect to be more useful for binary classification using the perceptron model? [Hint: $\text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(\alpha \mathbf{w}^T \mathbf{x})$ for any $\alpha > 0$.]

- ②
- ⇒ I agree that soft order constraint is more useful for binary classification. Because it allows more flexibility in the separation between two classes. In the context of perceptron model, since the prediction is based on $\text{sign}(\mathbf{w}^T \mathbf{x})$, the model only cares about the value is (+)ve or (-)ve. The hint $\text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(\alpha \mathbf{w}^T \mathbf{x})$ for $\alpha > 0$, illustrates that scaling the weights by a positive scalar does not change the classification. It will not change the sign of $(\mathbf{w}^T \mathbf{x})$ even when \mathbf{w} is small. So, we will still be able to classify.
- ⇒ In contrast, a hard order constraint imposes a strict separation betⁿ classes. If we use the hard order constraint with less parameters, the perceptron VC dimension decreases and it will be less likely to classify the same number of points with more parameters. Also, it leads to cause of overfitting. Because the model fits the noise in the training data rather than generalizing well to new data points.

3. LFD Exercise 4.8

Exercise 4.8

Is E_m an unbiased estimate for the out-of-sample error $E_{out}(g_m)$?

Answer->

③ For each model h_m , \hat{g}_m is independently learned from the validation set. When we take the expectation w.r.t. validation data set, the \hat{g}_m does not change. The expectation depends only on x_n , as in the derivation of equⁿ ① below-

$$E_{val}(\hat{g}) = \frac{1}{K} \sum_{x_n \in D_{val}} e(\hat{g}(x_n), y_n)$$

$$E_{D_{val}}[E_{val}(\hat{g})] = \frac{1}{K} \sum_{x_n \in D_{val}} E_{D_{val}}[e(\hat{g}(x_n), y_n)]$$

$$E_{D_{val}}[E_{val}(\hat{g})] = \frac{1}{K} \sum_{x_n \in D_{val}} E_{out}(\hat{g})$$

$$E_{D_{val}}[E_{val}(\hat{g})] = E_{out}(\hat{g}) \quad \text{--- (1)}$$

$$\therefore E_{D_{val}}[e(\hat{g}(x_n), y_n)] = E_{x_n}[e(\hat{g}(x_n), y_n)]$$

$$\Rightarrow E_{D_{val}}[e(\hat{g}(x_n), y_n)] = E_{out}(\hat{g})$$

So, I think E_m is an unbiased estimate for the out of sample error $E_{out}(\hat{g}_m)$.

4. LFD Exercise 4.11

Exercise 4.11

In this particular experiment, the black curve (E_{cv}) is sometimes below and sometimes above the red curve (E_{out}). If we repeated this experiment many times, and plotted the average black and red curves, would you expect the black curve to lie above or below the red curve?

④ In this particular experiment, the black curve (E_{cv}) represents the leave out cross validation error and the red curve represents the out-of-sample error. The relationship between these two curves can provide insights into how well the model generalizes. The black curve (E_{cv}) typically serves as a more optimistic estimate of the model's performance because it uses all available data points for validation during cross validation.

In contrast, the red curve (E_{out}) is an estimate of how well the model will perform on unseen data i.e. out of sample data.

When we take average of the red and black curves over multiple repetition of the experiment, we would generally expect the black curve E_{cv} to be below of the red curve E_{out} . This is because, E_{cv} will get benefit from the full dataset for validation, which will lead to an optimistic estimate of the model's generalization performance.

And therefore, we expect Black curve E_{cv} below red curve (E_{out}) often.

5.

(5) [1000 points] An End-to-End Learning System with Regularization and Validation: Predicting 1s vs. Not 1s.

We revisit the MNIST Handwritten Digits Dataset we worked with in the last homework to solve the problem of predicting whether a given image of a handwritten digit represents either the digit 1 or not the digit 1, i.e., if the n -th example is labeled as being the digit 1, then $y_n = +1$, and otherwise, $y_n = -1$.

First, you must perform the following steps to prepare to solve the problem:

- (1) [Combine Data] Combine the training and test sets (in `ZipDigits.train` and `ZipDigits.test` respectively) into a single dataset.

Combining test and training dataset into a **single dataset**

```
import numpy as np

# Combine the training and test data
combined_X = np.concatenate([Xdigitstrain, Xdigitstest], axis=0)
combined_y = np.concatenate([ydigitstrain, ydigitstest], axis=0)

print(f'Combined X shape: {combined_X.shape}')
print(f'Combined y shape: {combined_y.shape}')
```

```
Combined X shape: (9298, 256)
Combined y shape: (9298, 1)
```

Combining data after normalization and feature selection of the data, so matrix size reduced from 256 feature to 3 features.

```
Xaug = np.concatenate((Xaugtrain, Xaugtest), axis=0)
Naug, d = Xaug.shape
ycombined = np.concatenate((ydigitstrain, ydigitstest), axis=0)

assert(Xaug.shape[0] == Ndigitstrain + Ndigitstest)
assert(ycombined.shape[0] == Ndigitstrain + Ndigitstest)

print(f'Xaug shape {Xaug.shape}, ycombined shape {ycombined.shape}')
```

Xaug shape (9298, 3), ycombined shape (9298, 1)

- (2) [Compute Features] Use the algorithms you developed in the previous homework to compute two features for each example in the dataset.

For training dataset, computing 2 features: intensity and symmetry

```
print('Computing augmented training feature matrix')

Xaugtrain = computeAugmentedXWithFeatures(Xdigitstrain)

Naugtrain, d = Xaugtrain.shape

print(f'Xaugtrain shape {Xaugtrain.shape}')
```

```
→ Computing augmented training feature matrix
computing intensity feature
Input shape 7291, 256
Output shape (7291, 1)
computing symmetry feature
Input shape 7291, 256
Output shape (7291, 1)
Shape of augmented feature matrix: (7291, 3)
Xaugtrain shape (7291, 3)
```

For testing dataset, computing 2 features: intensity and symmetry

```
print('Computing augmented test feature matrix')

Xaugtest = computeAugmentedXWithFeatures(Xdigitstest)

Naugtest, d = Xaugtest.shape

print(f'Xaugtest shape {Xaugtest.shape}')
```

```
→ Computing augmented test feature matrix
computing intensity feature
Input shape 2007, 256
Output shape (2007, 1)
computing symmetry feature
Input shape 2007, 256
Output shape (2007, 1)
Shape of augmented feature matrix: (2007, 3)
Xaugtest shape (2007, 3)
```


- (3) **[Normalize Features]** For each data point, *shift* and then *rescale* the values of each feature across the entire dataset so that for each feature, so that the values of every feature are in the range $[-1, 1]$.

✓ Setup Labels for the 1s vs Non 1s Classification Task

```
def normalize(X, a=-1, b=1):
    N, d = X.shape
    Xnorm = np.ones((N, d))
    for col in range(1, d):
        Xcol = X[:, col]
        # Compute the min and max values of the current feature
        min_val = np.min(Xcol)
        max_val = np.max(Xcol)

        # Normalize the feature to the range [a, b]
        Xcolnorm = a + ((b - a) * (Xcol - min_val)) / (max_val - min_val)

        Xnorm[:, col] = Xcolnorm
    return Xnorm
```

```
[14] # Normalize the augmented feature matrix Xaug to obtain Xnorm.
Xnorm = normalize(Xaug)
N, d = Xnorm.shape

# Ensuring whether the shape of Xnorm matches the shape of Xaug.
assert(Xnorm.shape == Xaug.shape)
# Ensuring whether the first column of Xnorm consists of all ones
assert(np.all( Xnorm[:, 0] == np.ones((N, 1)) ))
```

- (4) **[Create Input and Test Datasets]** Select 300 data points from the dataset *uniformly at random* (and without replacement) to form the dataset \mathcal{D} . Use the remaining data points to form the test dataset $\mathcal{D}_{\text{test}}$. You must leave aside $\mathcal{D}_{\text{test}}$ and not touch it till the end of this exercise when we will compute the final output g of our learning system. We will use $\mathcal{D}_{\text{test}}$ to estimate $E_{\text{out}}(g)$.

For convenience, we will treat this as a regression problem with real valued targets ± 1 , until we output our final hypothesis g for classifying handwritten images as either the digit 1 or not the digit 1, at which point we will use $\text{sign}(g(x))$ to predict the class of a test data point x .

The standard polynomial feature transform generates features which are not ‘orthogonal’, making the columns in the data matrix dependent. This can be problematic for the one-step linear regression algorithm as it requires computing the (pseudo-)inverse of a matrix. An ‘orthogonal’ polynomial transform is

$$(x_1, x_2) \rightarrow (1, L_1(x_1), L_1(x_2), L_2(x_1), L_1(x_1)L_1(x_2), L_2(x_2), L_3(x_1), L_2(x_1)L_1(x_2), \dots),$$

where $L_k(x_i)$ is the k -th order polynomial transform applied to the i -th feature of the input data point x . See LFD Problem 4.3 for a recursive expression that defines the Legendre polynomials which can be implemented as an efficient iterative algorithm.

We will use the one-step (pseudo-inverse) algorithm for linear regression algorithm with weight decay regularization for learning. This corresponds to minimizing the augmented error $E_{\text{aug}}(w) = E_{\text{in}}(w) + \lambda w^T w$, where $E_{\text{in}}(w)$ is the sum of squared errors. The weights that minimize $E_{\text{aug}}(w)$ are $w_{\text{lin}}(\lambda) = (Z^T Z + \lambda I)^{-1} Z^T y$, where Z is the $N \times \tilde{d}$ matrix generated from the polynomial transform of the data points X in the data set $\mathcal{D} = (X, y)$ and w_{lin} is the $\tilde{d} \times 1$ regularized weight vector.

Now, complete the following tasks to arrive at the final hypothesis g .

```

▶ X, y, Xtest, ytest = splitDataSelectKRandomly(Xnorm, ycombined, 300)


print(f'Xnorm shape {Xnorm.shape} ycombined shape {ycombined.shape}\n\
X shape {X.shape} y shape {y.shape}\n\
Xtest shape {Xtest.shape}, ytest shape {ytest.shape}')

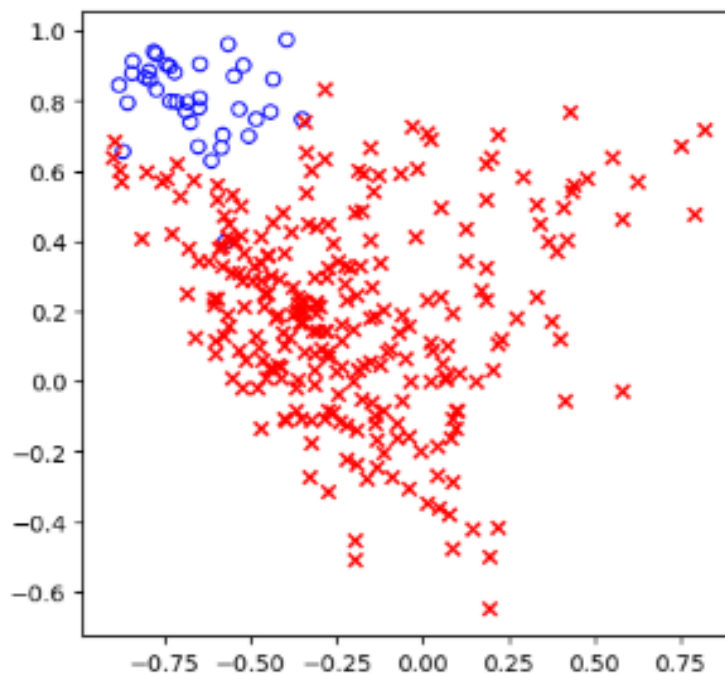
⇒ Xnorm shape (9298, 3) ycombined shape (9298, 1)
   X shape (300, 3) y shape (300, 1)
   Xtest shape (8998, 3), ytest shape (8998, 1)

```


Visualize the Data

```
[18] fig, axs = plt.subplots(figsize=(5, 5))
      plus1s = np.where(y == 1)[0]
      minus1s = np.where(y == -1)[0]
      Xplus1s = X[plus1s, :]
      Xminus1s = X[minus1s, :]
      axs.scatter(Xplus1s[:, 1], Xplus1s[:, 2], marker='o',
                  color='blue', facecolors='none', label='1')
      axs.scatter(Xminus1s[:, 1], Xminus1s[:, 2],
                  marker='x', color='red', label='Not 1')
```

 <matplotlib.collections.PathCollection at 0x7ec6cafaad40>



(Task 1) [100 points] 10-th order Polynomial Transform. Use the 10-th order Legendre polynomial feature transform to compute Z . Report the dimensions of Z .

✓ Question 5

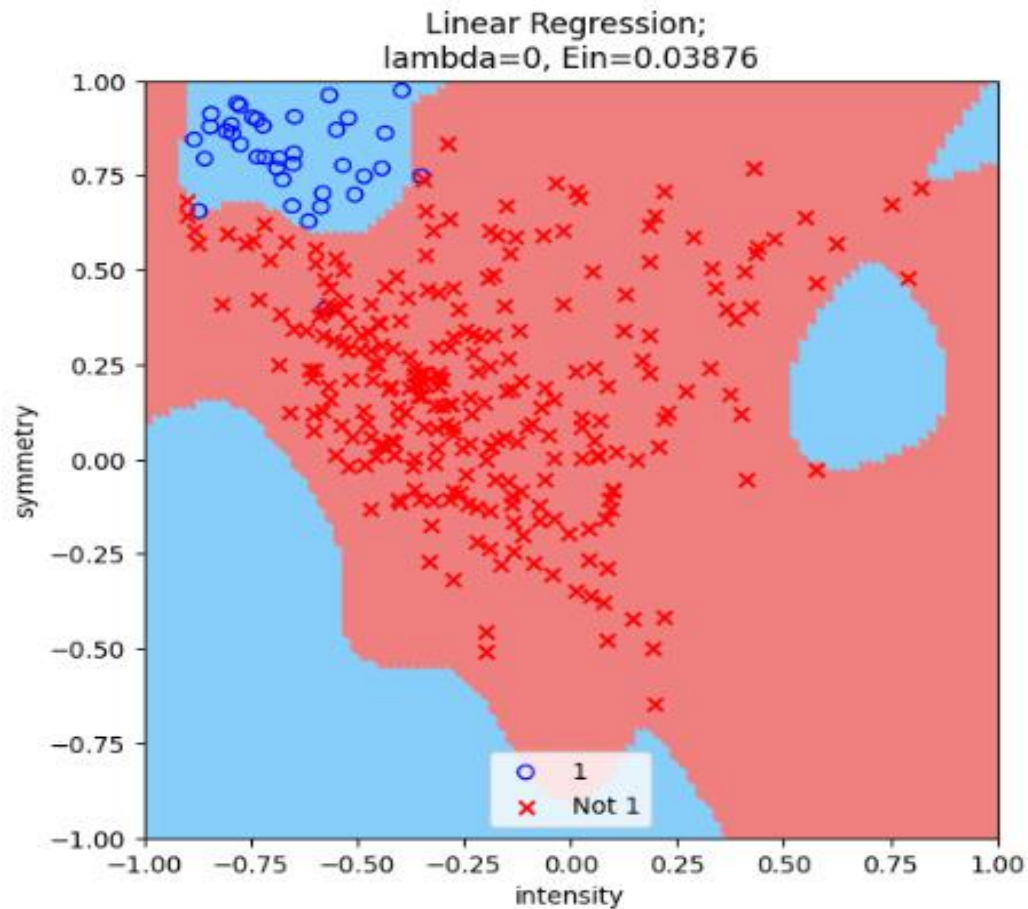
Task1

```
Q = 10 #degree of polynomial

Z = polynomialTransformLegendre(X, Q)
print(f'Z shape {Z.shape}')
Ztest = polynomialTransformLegendre(Xtest, Q)
print(f'Ztest shape {Ztest.shape}')
```

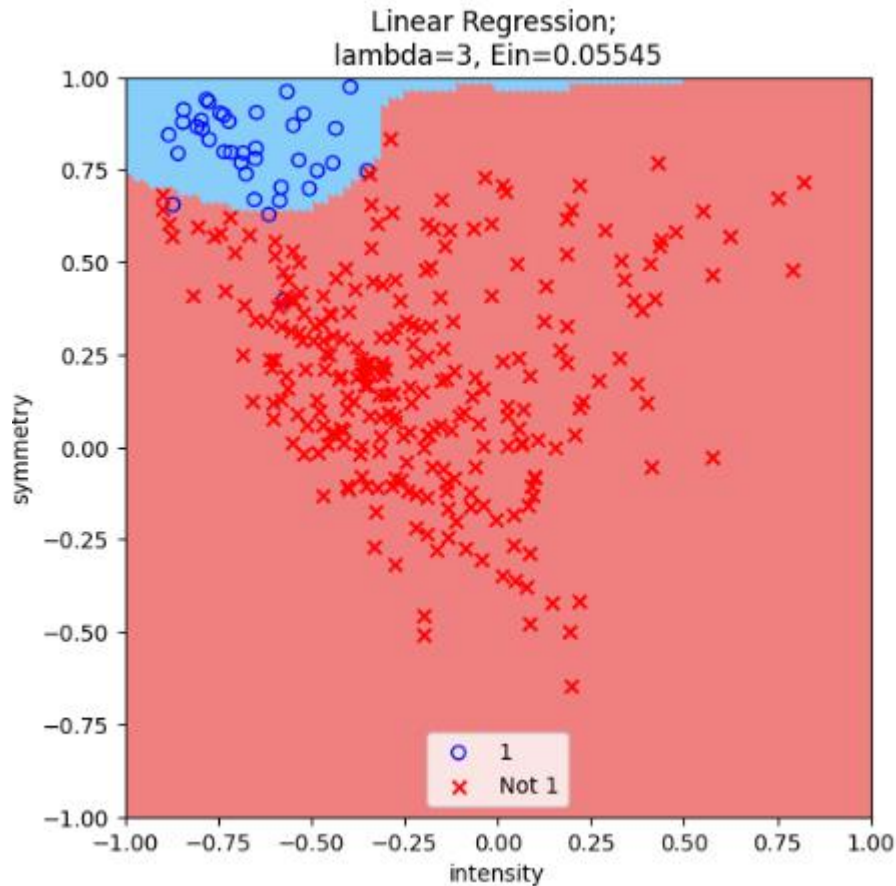
```
↔ Z shape (300, 66)
   Ztest shape (8998, 66)
```


(Task 2) [100 points] **Overfitting.** Plot the decision boundary of the output of the regularized linear regression algorithm without any regularization ($\lambda = 0$). What do you observe, overfitting or underfitting?



I observe **Overfitting** because the hypothesis is trying to fit the data very closely and this is very complex. We can also see it in the above plot.

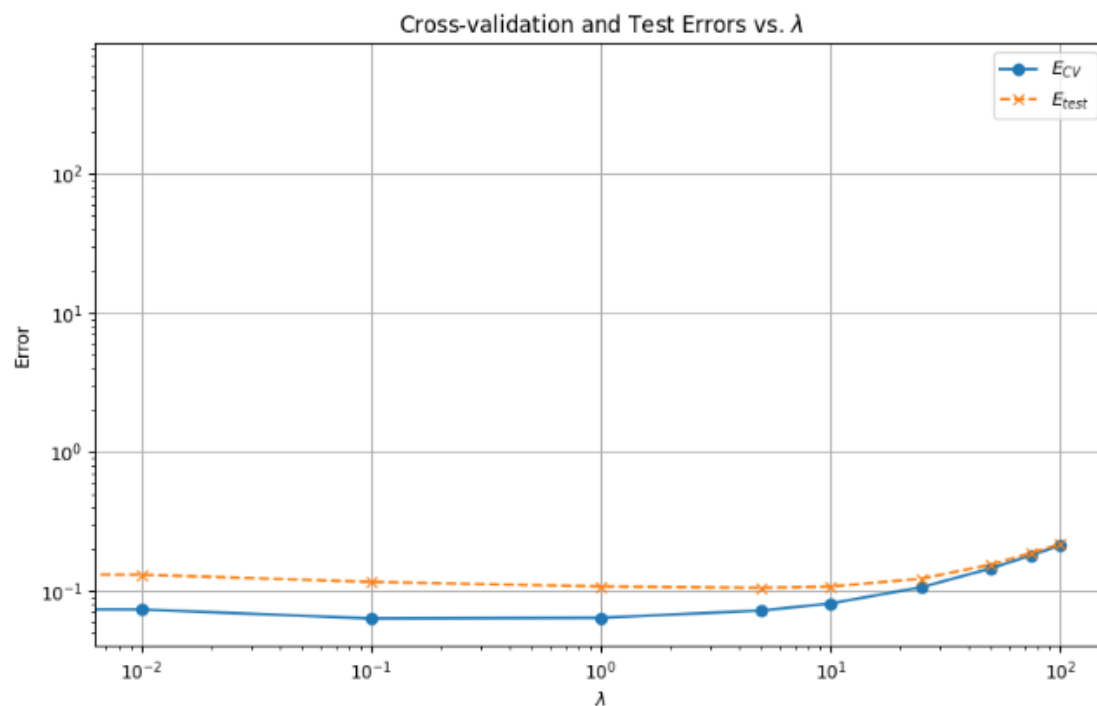
(Task 3) [100 points] **Regularization.** Plot the decision boundary of the output of the regularized linear regression algorithm with $\lambda = 3$. Do you observe overfitting or underfitting?



As regularized linear regression model with $\lambda=3$, shows that it classifies most of the points correctly. As we increase λ to 3, we can see that it reduces the overfitting and move towards the best fitting case, and further increase to that will lead to underfitting. Since here, the boundary is simple and fails to capture some finer patterns, this indicates **underfitting** rather than overfitting.

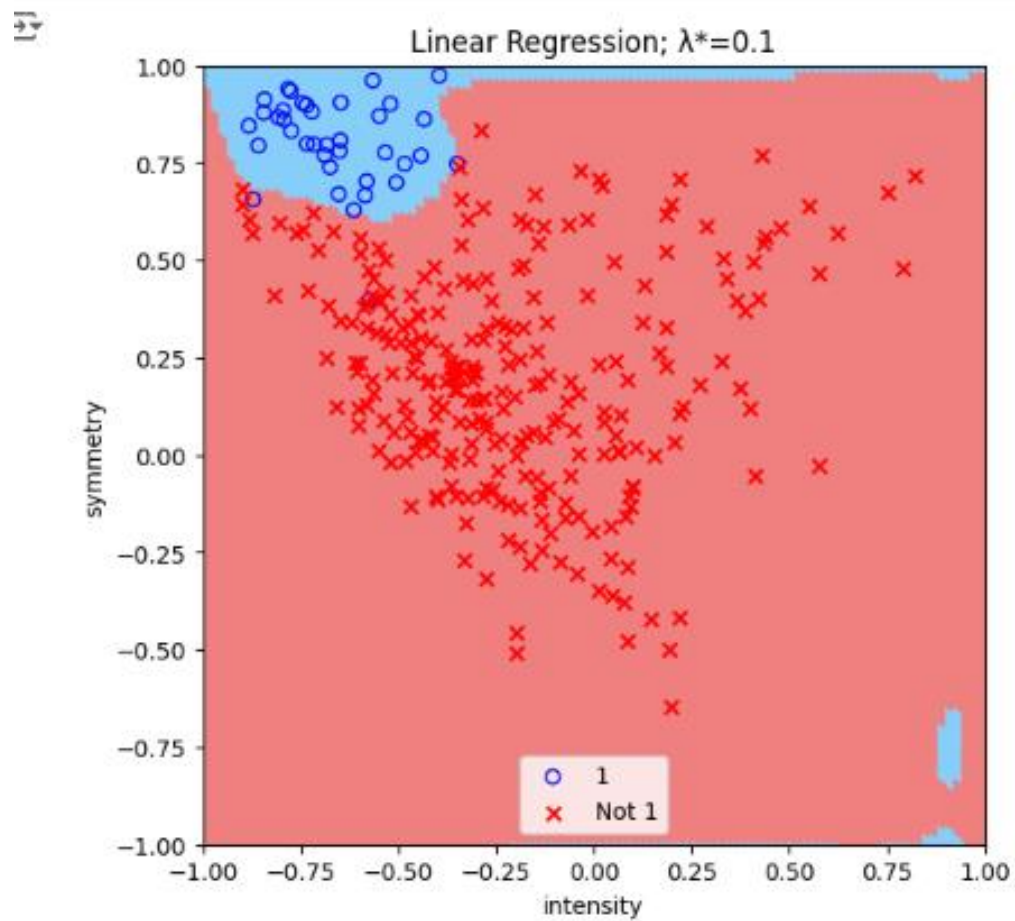
(Task 4) **[200 points] Cross Validation.** Use leave-one-out cross validation to estimate $E_{CV}(\lambda)$ for $\lambda \in \{0, 0.01, 0.1, 1, 5, 10, 25, 50, 75, 100\}$. Plot E_{CV} versus λ and $E_{\text{test}}(w_{\text{lin}}(\lambda))$ versus λ on the same plot. Comment on the behavior of E_{CV} and E_{test} versus λ . Here, E_{CV} and E_{test} are the regression, sum of squared errors.

```
Linear Regression with Regularization with lambda=0, Ecv = 59.12535988130602, Etest = 554.7471824050991
Linear Regression with Regularization with lambda=0.01, Ecv = 0.07351194328903848, Etest = 0.1307966605866893
Linear Regression with Regularization with lambda=0.1, Ecv = 0.06329776358431723, Etest = 0.1160955579859954
Linear Regression with Regularization with lambda=1, Ecv = 0.06403541718764742, Etest = 0.10770073704450285
Linear Regression with Regularization with lambda=5, Ecv = 0.07224899559937102, Etest = 0.10524670386226478
Linear Regression with Regularization with lambda=10, Ecv = 0.08118061382713532, Etest = 0.10747046360175373
Linear Regression with Regularization with lambda=25, Ecv = 0.10646785920962025, Etest = 0.12253696291903517
Linear Regression with Regularization with lambda=50, Ecv = 0.14534812040475287, Etest = 0.15486472120304343
Linear Regression with Regularization with lambda=75, Ecv = 0.18051993348290166, Etest = 0.187482127915091
Linear Regression with Regularization with lambda=100, Ecv = 0.21247859321759816, Etest = 0.21825643584552326
```



As we can see, Initially, as the regularization parameter (λ) increases, the cross-validation error (ECV) decreases, indicating a reduction in overfitting. However, beyond a certain threshold, the ECV begins to increase, mainly due to excessive regularization introducing bias into the model. As λ further increases, test error values initially increase gradually, but then escalate rapidly. This indicates that excessive regularization is detrimentally affecting model performance. Eventually, both λ and ECV increase, indicating that regularization has a negative impact on the model, leading to an increase in bias.

(Task 5) [100 points] Pick λ . Use the cross validation errors from the previous step to pick the best value of λ , and call it λ^* . Plot the decision boundary corresponding to the weights $w_{\text{lin}}(\lambda^*)$.



Best lambda (λ^*): 0.1

(Task 6) [100 points] **Estimate Classification Error.** Use $w_{\text{lin}}(\lambda^*)$ for classification and estimate the *classification* out-of-sample error $E_{\text{out}}(w_{\text{lin}}(\lambda^*))$ for your final hypothesis g . Estimate $E_{\text{out}}(g)$ to distinguishing between digits that are 1s and not 1s (give the 99% error bar).

`Eout(wlin(λ^*)): 0.1160955579859954`

`99% Confidence Interval for E_out: [0.09893699188410862, 0.13325412408788218]`



```
# Calculate classification error rates
error_rate_ones = misclassified_ones / total_ones
error_rate_non_ones = misclassified_non_ones / total_non_ones

# Calculate overall classification error (Eout(g))
total_misclassified = misclassified_ones + misclassified_non_ones
total_samples = len(y_pred)
overall_error_rate = total_misclassified / total_samples

# Calculate 99% error bars
z_score = 2.576 # Z-score for 99% confidence interval
one_std_dev = np.sqrt(overall_error_rate * (1 - overall_error_rate) / total_samples)
error_bar = z_score * one_std_dev

print(f"Classification Error for 1s: {error_rate_ones:.4f}")
print(f"Classification Error for non-1s: {error_rate_non_ones:.4f}")
print(f"Overall Classification Error (Eout(g)): {overall_error_rate:.4f}")
print(f"99% Confidence Interval Error Bar: ±{error_bar:.4f}")
```



```
Classification Error for 1s: 0.1420
Classification Error for non-1s: 0.0164
Overall Classification Error (Eout(g)): 0.0336
99% Confidence Interval Error Bar: ±0.0049
```


(Task 7) [100 points] Is E_{CV} biased? Comment on whether $E_{CV}(\lambda^*)$ is an unbiased estimator of $E_{\text{test}}(w_{\text{lin}}(\lambda^*))$ (treated as regression error). Why or why not?

```
# Compute Etest using the best lambda and the separate test dataset
Etest = squaredError(w_reg_best, Ztest, ytest)

# Compute ECV using cross-validation, leave-one-out
Ecv = linearRegressionCVError(Z, y, best_lambda)

# Print both ECV and Etest
print(f'ECV( $\lambda^*$ ): {Ecv}')
print(f'Etest( $w_{\text{lin}}(\lambda^*)$ ): {Etest}')
```

```
ECV( $\lambda^*$ ): 0.06329776358431723
Etest( $w_{\text{lin}}(\lambda^*)$ ): 0.1160955579859954
```

Q 5 task 7

In general E_{cv} can be biased estimator of E_{test} (the test error) for a given model. The bias arises from the fact that E_{cv} is computed on the same dataset used for model selection.

Whereas, E_{test} is computed on different a separate, independent test dataset.

E_{cv} typically underestimates the true error because the model has already seen the data points that were left out in E_{cv} .

Specifically for the best d^* selected based on E_{cv} , $E_{cv}(d)$ is not an unbiased estimator of $E_{test}(w_{in}(d))$ treated as a regression error. This is because $E_{cv}(d^*)$ is calculated on the same dataset used for model selection (the dataset with one data points left out at a time for cross validation) and the model has already seen these data points during training. Therefore, it does not fully capture the model's performance on unseen data, it tends to be optimistic (biased low) in estimating the true test error.

(Task 8) [200 points] **Data snooping.** $E_{\text{test}}(w_{\text{lin}}(\lambda^*))$ an unbiased estimator of $E_{\text{out}}(w_{\text{lin}}(\lambda^*))$ (treat them as classification errors)? Why or why not? If not, what could we do differently to fix things, so that it is? Explain.

Question 5) task 8

$E_{\text{test}}(w_{\text{lin}}(\lambda))$ is not an unbiased estimator of $E_{\text{out}}(w_{\text{lin}}(\lambda))$ when treating them as classification error, and this is due to data snooping or data leakage.

to fix this thing we can do -

- 1) Split the data \rightarrow Divide dataset into 3 subsets - a training, a validation set and test set. The training set is used for model training, the validation set is used for hyperparameter tuning (e.g. selecting the best λ^*) and the test set is kept entirely separate and untouched until the final evaluation.
- 2) Model selection \rightarrow By using validation sets to select best λ^* and train the model using only the training sets, this will ensure that the model selection is independent of test data.
- 3) Evaluation on test dataset \rightarrow After choosing the best λ^* and training the model on training data, E_{test} computed on test data will give the unbiased estimation of $E_{\text{out}}(w_{\text{lin}}(\lambda))$ as it is based on the data the model has never seen during model selection or training.

Link for google colab:

https://colab.research.google.com/drive/1X8vYwvtz_4j_TOyWjttq15iTtj4q8XaG?usp=drive_link