Home Work – 4

Saiprakash nalubolu

B01037579

Q1. LFD Exercise:4.3

(a) Given H is fixed:

If H is simpler than f before increasing f: If we increase the complexity of f, deterministic noise will generally increase because H isn't complex enough to capture the new intricacies of f. This mismatch means H will struggle to fit f well, leading to higher noise and a higher chance of overfitting, since H tries to stretch beyond its limits to match the data.

If H is more complex than f before increasing f: As we increase the complexity of f, initially, the deterministic noise might decrease because H has the capacity to handle a bit more complexity. This could lead to less overfitting as the models in H have the necessary flexibility. However, if f becomes more complex than H, deterministic noise will start to increase again, as H no longer can keep up, which can again increase the risk of overfitting.

(b) Given f is fixed:

If H is simpler than f before decreasing H: Decreasing the complexity of H further, will increase deterministic noise because it moves H even further from being able to represent f. This increased gap will likely lead to models that are too generalized and may underfit, which means they won't capture the necessary patterns in the data and won't overfit since they don't have the capacity to capture much detail, including noise.

If H is more complex than f before decreasing H: Reducing the complexity of H initially lowers deterministic noise because it cuts out the excess capacity that might model random noise instead of the underlying pattern in f. This reduces overfitting since the model is less likely to capture the noise. But if H is simplified too much, passing the point where it's just complex enough to model f, deterministic noise starts to rise again because now H is too simplistic. This increase in noise occurs even as the model's potential to overfit decreases because the model is becoming increasingly unable to capture detail, tilting more towards underfitting.

In simpler terms, if the model's toolset H doesn't match up well with what's happening in real life, which is f, you can end up with errors either by having a toolset that's too basic or by having one that's excessively complicated. Finding the sweet spot for H relative to f is crucial—it should be complex enough to model the reality accurately without being so complex that it starts to mistake random fluctuations as meaningful patterns.

Q2. LFD Exercise 4.6

In the context of a perceptron model, which is fundamentally a linear classifier, there are two types of constraints that can be considered when training the model: hard-order and soft-order.

When we apply a **hard-order constraint**, we are typically limiting the model to a very specific and strict decision boundary. This boundary is determined by the weights of the model w, and the input features x, through the equation $\omega^T x$ = 0. If the model has fewer parameters, it means that the VC dimension, which is a measure of the model's complexity is reduced. A lower VC dimension implies that the model has a limited ability to classify points. It's structured to classify a set number of points accurately, and adding more points or more variability to the dataset might lead to misclassifications. Hence, with fewer parameters and a hard-order constraint, the perceptron is more robust against overfitting but may not classify as many points correctly.

On the other hand, if we apply **a soft-order constraint** to the perceptron, we're giving the model more flexibility. The soft-order constraint does not force the model to maintain a strict adherence to the hard margins defined by the weights and the decision boundary. Instead, it allows for some flexibility, meaning that the sign of the classification decision sign($\omega^T x$) remains the same even when the magnitude of the weights w, is varied (scaled by a factor a>0). This flexibility can be beneficial in scenarios where the data is not perfectly linearly separable, or where there's noise in the data. The soft-order constraint allows the perceptron to still classify points correctly despite these challenges.

Therefore, in practice, the soft-order constraint is often more useful for binary classification using the perceptron model, because it provides a balance between strictness and flexibility, allowing the model to handle a variety of data distributions more effectively.

Q3. LFD Exercise: 4.8

For each model denoted as Hm, the corresponding g_m^- is derived independently from the validation dataset. When evaluating expected out of sample error with respect to data set g_m^- is constant. Hence the expectation only depends on xn.
From the derivation in the LFD book Equation 4.8,

$$\begin{aligned}\mathbb{E}_{\mathcal{D}_{\text{val}}}\left[E_{\text{val}}(g^-)\right] &= \frac{1}{K}\sum_{\mathbf{x}_n\in\mathcal{D}_{\text{val}}}\mathbb{E}_{\mathcal{D}_{\text{val}}}\left[\mathsf{e}\left(g^-(\mathbf{x}_n),y_n\right)\right],\\ &= \frac{1}{K}\sum_{\mathbf{x}_n\in\mathcal{D}_{\text{val}}}E_{\text{out}}(g^-),\\ &= E_{\text{out}}(g^-). \qquad (4.8)\end{aligned}$$

$$E_m = E_{\text{val}}(g_m^-); \quad m = 1,\ldots,M.$$

Hence it is reasonable to consider Em as the unbiased estimate of out of sample error Eout(g_m^-).

Q4. LFD Exercise 4.11

This fluctuation in the experiment is expected because cross-validation is an estimation of Eout based on the training data alone, and it can vary depending on the specific samples chosen for the validation set in each run of the experiment.

When we repeat the experiment many times and average the results, we can expect the following behavior:

1. The average Ecv is expected to be a good estimate of Eout, especially with the k-fold cross-validation procedure.
2. Bias and Variance of Ecv; Ecv is subject to two sources of error, **bias** due to it being an estimate and **variance** due to the randomness in which data points end up in the validation set. However, as we average over many experiments, the variance component diminishes.
3. Convergence of Averages; With many repetitions, the average Ecv would converge to the expected value of the cross-validation error, which should closely track the average Eout if the model and validation procedure are consistent.

Therefore, if we averaged the black and red curves over many repetitions of the experiment, we would expect:

The average Ecv to generally lie slightly above the average Eout if there is a slight optimistic bias in cross-validation, which is often the case because the model is indirectly exposed to the validation data during model selection. However, if the cross-validation procedure is well-tuned and the model is not too complex relative to the amount of data, the average Ecv may lie very close to the average Eout, potentially even below it if the training data is very representative of the overall population from which the test data are drawn.

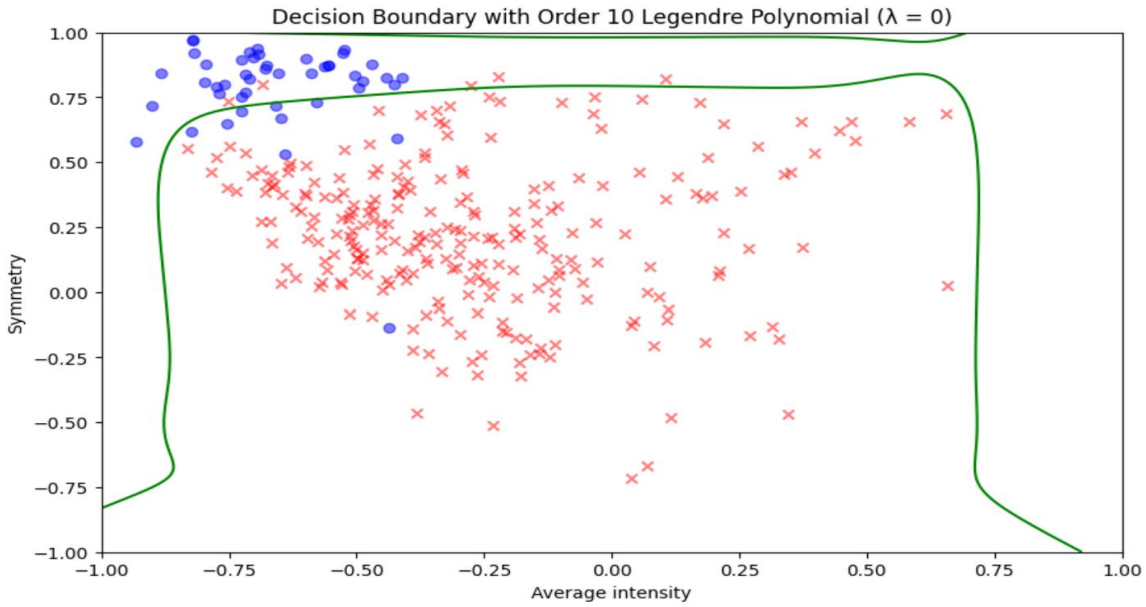
In summary, without overfitting and with a robust cross-validation setup, the average Ecv would lie close to, but often slightly above, the average Eout due to the inherent optimism in the cross-validation error estimate. However, if the training data is especially representative, it's possible for the average Ecv to be even slightly below the average Eout.

Q5.

Task-1:

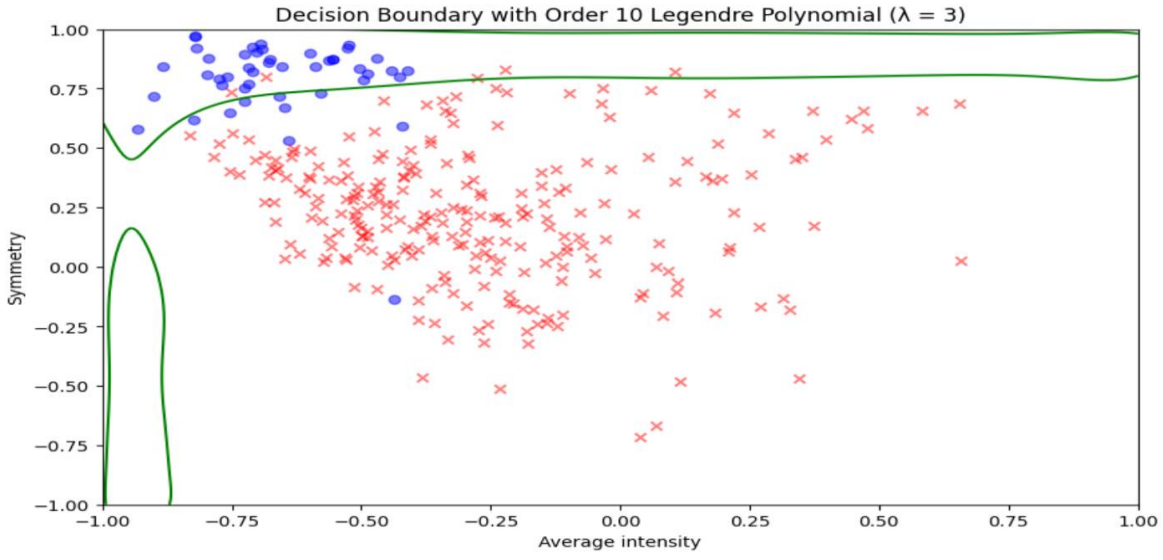
```
The dimensions of Z_train after applying the 10th-order Legendre transform are: (300, 22)
The dimensions of Z_test after applying the 10th-order Legendre transform are: (8998, 22)
```

Task-2:



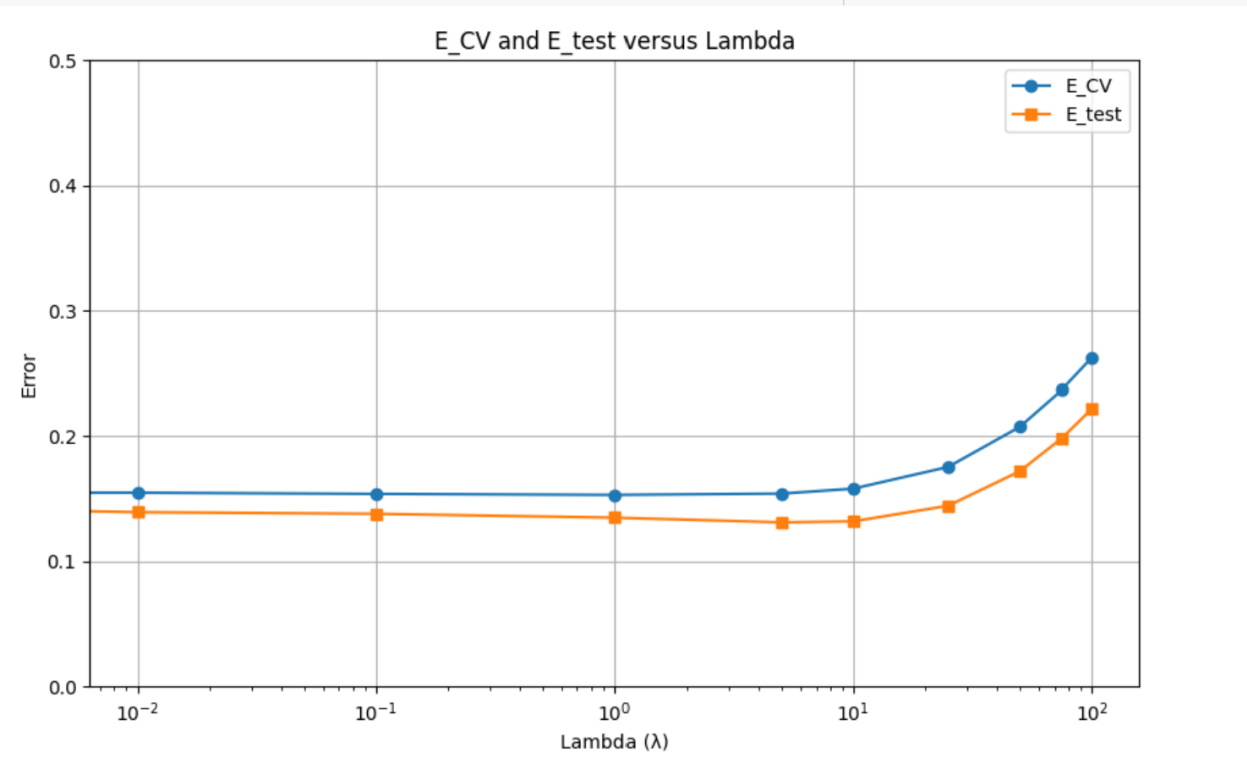
The Decision boundary seems to be Overfitting the data. And as per theory when Lambda is 0 It generally overfits, and as we increase lambda gradually it fits the data and then it proceeds to underfitting.

Task-3:



When Lambda=3, the decision boundary fits the data better when compared to lambda=0, but still it is over fitting the data as seen from graph.

Task-4:



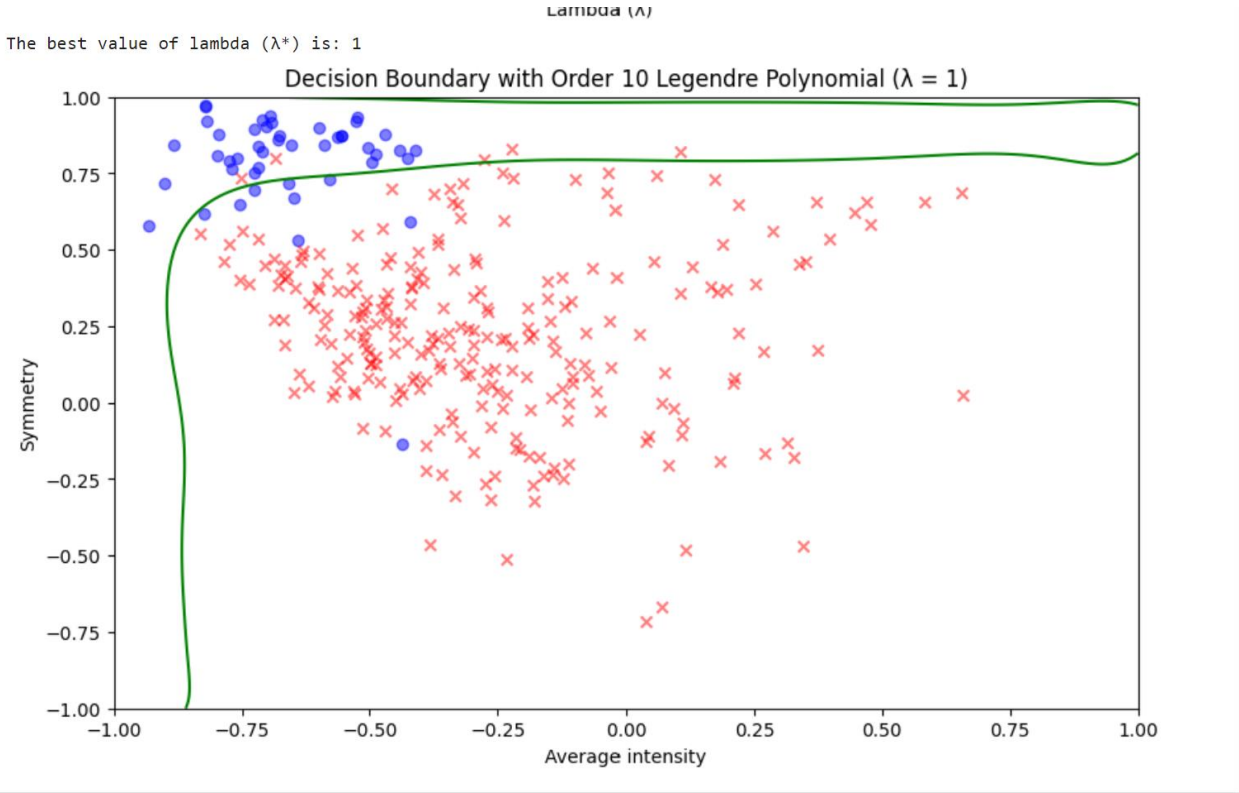
From the graph, as λ increases from 0 to a certain point, both E_CV (the cross-validation error) and E_test (the test error) seem to be relatively stable. This indicates that small amounts of regularization do not significantly impact the performance of the model on both the cross-validation set and the test set.

As λ continues to increase past the threshold, E_CV, E_test increase. The increasing trend in errors as λ becomes large suggests that the model starts to underfit: the regularization term is dominating the error minimization process, which causes the model to become too simple. It fails to capture the underlying pattern in the data, thereby increasing the prediction errors.

This is a typical behavior where too little regularization (very low λ) doesn't penalize complexity enough, which might lead to overfitting, and too much regularization (very high λ) adds too much bias into the model, leading to underfitting.

It is important to find a balance for λ that minimizes both E_CV and E_test, which often corresponds to the "sweet spot" where the model has the right level of complexity to generalize well to unseen data. In practice, this value of λ could be found through techniques like grid search or randomized search over a range of λ values, and using cross-validation to determine which one yields the best validation performance.

Task-5:



Task-6:

```
E_out (classification error): 0.04000889086463658
99% Confidence interval for E_out: [0.03468712275665169, 0.04533065897262148]
```

Task-7:

The cross-validation error ECV(λ ∗) is an estimate of the out-of-sample error for the model trained with the regularization parameter λ ∗. The goal of ECV is to provide an unbiased estimate of Eout by using a part of the training data to validate the model. However, there are several factors that can affect the bias of ECV:

Estimation Bias: If the training set is small, ECV can be a biased estimator because the training sets in each fold are smaller than the full training set. This can lead to overestimation of the error due to higher variance in the smaller training sets.

Model Bias: ECV is also dependent on the specific data points that are left out during each fold of the cross-validation. If the data has high variance or if the left-out points are not representative of the overall distribution, the estimate can be biased.

Mismatch in Error Metrics: If ECV is computed using a different error metric than Etest, it can lead to bias. For example, if ECV is computed using the mean squared error for regression, but Etest is considered as a classification error, they are not directly comparable.

Lambda Selection: $\lambda*$ is chosen based on minimizing ECV, which might not always correspond to minimizing Etest. This can happen if the model overfits the validation set used in cross-validation but not the test set, or vice versa.

ECV($\lambda*$) is a reasonably good and relatively unbiased estimator of Etest. This is particularly true if the data is large and randomized, as is often the case with datasets like MNIST.

However, ECV($\lambda*$) cannot be guaranteed to be an entirely unbiased estimator because it inherently uses the data to both train and validate the model, and the optimal λ is selected based on its own estimation of error, which can lead to a selection bias. To make a more concrete assessment we can also consider the variance of the error estimates. If they are close and the variance is low, it indicates that ECV is a good estimator for Etest.

Task-8:

The test error Etest(wlin($\lambda*$)) is an estimate of the out-of-sample error Eout(wlin($\lambda*$)) based on a specific subset of the data that has not been used during the training process. In the context of unbiased estimation, we can consider the following:

Data Distribution: If the test set is a random, representative sample of the overall data distribution, and if it has not been used in any way to make decisions during the learning process (such as in the selection of λ), then Etest is likely to be an unbiased estimator of Eout.

Model Generalization: An unbiased estimator should generalize well to unseen data. Etest can be considered unbiased if the test set can reliably represent new data that the model may encounter.

Moreover we have used Leave one out cross validation to ensure that the test set is completely held out and not used in any way during the training or validation process. Hence, Etest(wlin($\lambda*$)) is an unbiased estimator of Eout(wlin($\lambda*$)).

https://colab.research.google.com/drive/1ltJ4AUFvl0veVP7YKx_wAYOLcbuBVvc5#scrollTo=ZGSDbUdfVCct