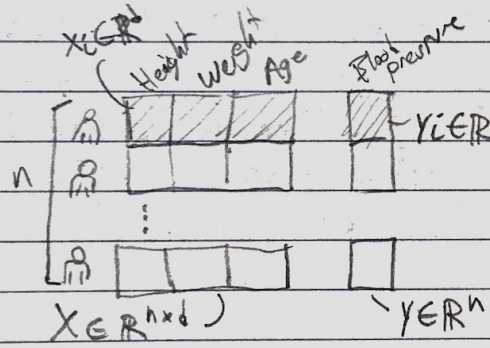
Homework 3

Q 1

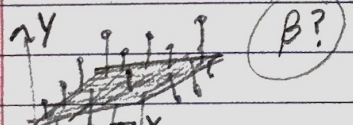
Suppose that we have a dataset $D = \{(x_i, y_i)\}_{i=1}^n$ of n samples, each having input features $x_i \in \mathbb{R}^d$ and output lables $y_i \in \mathbb{R}$. Suppose that there exists a vector $\beta \in \mathbb{R}^d$ (aka. the model) such that

$$y_i = x_i^\top \beta + \varepsilon_i, \text{ for } i \in \{1, \dots, n\},$$

where $\{\varepsilon_i\}_{i=1}^n$ are independent and identically distributed Gaussian random variables with mean 0 and variance $\sigma^2 > 0$. Denote by $X \equiv [x_i^\top]_{i=1}^n \in \mathbb{R}^{n \times d}$ the matrix comprising sample features in its rows and $y \equiv [y_i]_{i=1}^n \in \mathbb{R}^n$ the vector of all labels.



$$D = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$$
$$y_i = \beta^T x_i + \varepsilon_i, \qquad i = 1, \dots, n$$
$$\varepsilon_i \sim N(0, \sigma^2) \text{ i.i.d.}$$



(1.1) Prove that the maximum likelihood estimator (MLE) equals the least-squares estimator (LSE), i.e.:

$$\hat{\beta}^{ML} \equiv \arg\max_{\beta \in \mathbb{R}^d} P(D \mid \beta) = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n (y_i - \beta^T x_i)^2$$

Do we need to know σ^2 to compute $\hat{\beta}^{ML}$?

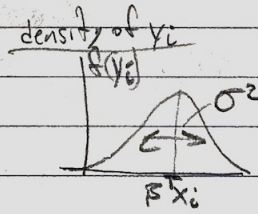
Say $y_i = f(x_i) + \varepsilon_i$ you are estimating

Computing least squares error is like saying data with some gaussian noise in i.i.d.

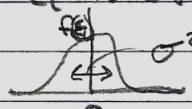
(1.1 continued) $\hat{\beta}^{ML} \equiv \arg\max_{\beta \in \mathbb{R}^d} P(\mathcal{D}|\beta) = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$

$P(\mathcal{D}|\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \left[e^{-(y_i - \beta^T x_i)^2 / 2\sigma^2} \right]$, this is a lot to do out so we can take the log since $\arg\max_{\beta \in \mathbb{R}^d} \log P(\mathcal{D}|\beta) = \arg\max_{\beta \in \mathbb{R}^d} P(\mathcal{D}|\beta)$

Also, we know this to be the equation for $P(\mathcal{D}|\beta)$ because the noise/error $\{\varepsilon_i\}_{i=1}^n$ are i.i.d. gaussian random variables so y_i also is of the same form. The probability of the dataset is given by the product of a bunch of gaussians.



if $\beta^T x_i = f(x_i)$ we have $y_i = f(x_i) + \varepsilon_i$ so if ε_i is distributed like:



Then y_i will have the same shape just shifted by $\beta^T x_i$ i.e. $y_i \sim N(\beta^T x_i, \sigma^2)$

Now we know why we use the gaussian distribution for the equation for $P(\mathcal{D}|\beta)$.

We can take the log as said above but similarly we can minimize the negative log, $\arg\max_\beta P(\mathcal{D}|\beta) = \arg\min_\beta -\log P(\mathcal{D}|\beta)$

$$-\log P(\mathcal{D}|\beta) = -\log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_i - \beta^T x_i)^2 / 2\sigma^2}$$

$$= -\log \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \prod_{i=1}^{n} e^{-(y_i - \beta^T x_i)^2 / 2\sigma^2}$$

This term becomes a constant we can call $-C$

$$= -\sum_{i=1}^{n} \log e^{\frac{-(y_i - \beta^T x_i)^2}{2\sigma^2}} - C$$

$$= \sum_{i=1}^{n} \frac{y_i - \beta^T x_i)^2}{2\sigma^2} - C = \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 - C$$

When we find the arg min it does not matter if we subtract a constant C or multiplying by a scalar $\frac{1}{2\sigma^2}$

Homework 3

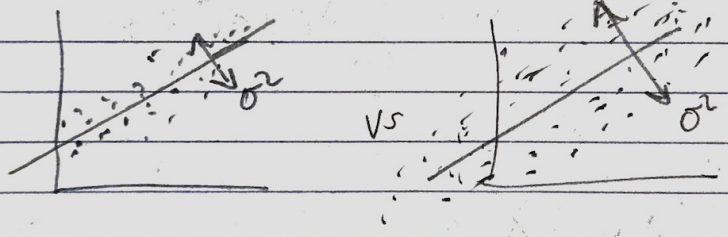
(1.1 cont) so if $\arg\min_{\beta}$ is the same then we can say

$$\arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$$

so we arrive at the minimum (or least) of the error $(y_i - \beta^T x_i)$ squared.

So we finally see, $\hat{\beta}^{ML} = \arg\max_{\beta \in \mathbb{R}^d} P(\mathcal{D}|\beta) = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$

we also see that we do not need σ^2 to compute $\hat{\beta}^{ML}$! Since it is a scalar it did not impact our argument. also intuitively the noise around our "best fit/ least squares error" line does not impact what the line should be. ex:



(1.2) Show that $\mathbb{E}[\hat{\beta}^{ML}] = \beta$ and $\text{cov}(\hat{\beta}^{ML}) = \sigma^2 (X^T X)^{-1}$

$y_i = \beta^T x_i + \varepsilon_i$, ε_i are i.i.d., $\mathbb{E}[\varepsilon_i] = 0$, $\mathbb{E}[\varepsilon_i^2] = \sigma^2$
$i = 1, \dots, n$ ↳ mean of the noise ↳ variance of the noise

$y = X\beta + \varepsilon$

$$\hat{\beta}^{ML} = \arg\max_{\beta \in \mathbb{R}^d} P(\mathcal{D}|\beta) = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$$

So, To find the expectation of the ML estimate of β we can also look at the square error loss $f(\beta) = \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$. setting its gradient equal to zero will help us find the $\arg\min_{\beta}$ and we will use that to find $\mathbb{E}[\hat{\beta}^{ML}]$.

(1.2 continued)

$\nabla f(\beta) = \sum_{i=1}^{n} \nabla f_i(\beta)$, where $f_i(\beta) = (y_i - \beta^T x_i)^2$

$$\nabla f_i(\beta) = \begin{bmatrix} \frac{\partial f_i(\beta)}{\partial \beta_1} \\ \vdots \\ \frac{\partial f_i(\beta)}{\partial \beta_n} \end{bmatrix} \in \mathbb{R}^d \quad \text{and} \quad \frac{\partial f_i(\beta)}{\partial \beta_k} = \frac{\partial}{\partial \beta_k} (y_i - \underbrace{\beta_1 x_{i1} - \beta_2 x_{i2} - \dots - \beta_d x_{id}}_{\beta^T x_i})^2$$

$$= \frac{\partial}{\partial \beta_k} (y_i - \beta_k x_{ik} - \sum_{j \neq k} \beta_j x_{ij})^2$$

$$= -2(y_i - \beta^T x_i) x_{ik}$$

and so, $\nabla f_i(\beta) = -2(\underbrace{y_i - \beta^T x_i}_{\in \mathbb{R}}) \underbrace{x_i}_{\in \mathbb{R}^d} \in \mathbb{R}^d$

also, $\nabla f(\beta) = 2 \sum_{i=1}^{n} (\beta^T x_i - y_i) x_i = 2 \sum_{i=1}^{n} x_i (x_i^T \beta - y_i)$

$$= 2 \sum_{i=1}^{n} x_i x_i^T \beta - 2 \sum_{i=1}^{n} x_i y_i$$

$$= 2 X^T X \beta - 2 X^T y$$

$$X = \begin{bmatrix} x_1 \cdots \\ x_2 \cdots \\ x_n \cdots \end{bmatrix} \in \mathbb{R}^{n \times d}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$$

$\arg\min_\beta$ where $\nabla f(\beta) = 0$ so, $0 = 2X^T X\beta - 2X^T y$

$$2X^T y = 2X^T X \beta$$

$$\beta = (X^T X)^{-1} X^T y$$

$\hat{\beta} = (X^T X)^{-1} X^T y$, $y = X\beta + \varepsilon$

$\hat{\beta} = (X^T X)^{-1} X^T (X\beta + \varepsilon)$

$= \underbrace{(X^T X)^{-1} X^T X}_{I} \beta + \underbrace{(X^T X)^{-1} X^T}_{G} \varepsilon$

$\hat{\beta} = I\beta + G\varepsilon$

$\mathbb{E}[\hat{\beta}] = \beta + \mathbb{E}[G \cdot \varepsilon] = \beta + G \cdot \mathbb{E}[\varepsilon] = \beta + G \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

$\mathbb{E}[\hat{\beta}] = \beta$

sum of the expectations is the expectation of the sums

Homework 3

(1.2 cont.) $\text{cov}(\hat{\beta}^{ML}) = \sigma^2 (X^T X)^{-1}$

$\text{cov}(\hat{\beta}^{ML}) = \mathbb{E}\left[(\hat{\beta} - \mathbb{E}[\hat{\beta}])(\hat{\beta} - \mathbb{E}[\hat{\beta}])^T\right]$, outer product $\Rightarrow$ matrix $\in \mathbb{R}^{d \times d}$

$= \mathbb{E}\left[(\hat{\beta} - \beta)(\hat{\beta} - \beta)^T\right]$, we have shown $\hat{\beta} = \beta + G\varepsilon$ where $G = (X^T X)^{-1} X^T$

[so, $\hat{\beta} - \beta = G\varepsilon$]

$= \mathbb{E}\left[(G \cdot \varepsilon)(G \cdot \varepsilon)^T\right]$, where ε is the vector containing all of the noise

$= \mathbb{E}[G \varepsilon \varepsilon^T G^T]$, since G is a constant then,

$= G\, \mathbb{E}[\varepsilon \cdot \varepsilon^T] G^T$

$\mathbb{E}[\varepsilon \cdot \varepsilon^T] = \begin{bmatrix} \mathbb{E}[\varepsilon_1^2] & \mathbb{E}[\varepsilon_1 \varepsilon_2] & \cdots \\ & \ddots & \\ & & \mathbb{E}[\varepsilon_n^2] \end{bmatrix} = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \sigma^2 \end{bmatrix}$, $\mathbb{E}[\varepsilon_i \varepsilon_j] = \mathbb{E}[\varepsilon_i]\,\mathbb{E}[\varepsilon_j]$ ↳ independent so, $= 0$

$= \sigma^2 I$

$= G \sigma^2 I G^T$

$= \sigma^2 G G^T$

$= \sigma^2 (X^T X)^{-1} X^T \left[(X^T X)^{-1} X^T\right]^T$

$= \sigma^2 (X^T X)^{-1} X^T X (X^T X)^{-1}$

$= \sigma^2 (X^T X)^{-1} \cdot I$

$\text{cov}(\hat{\beta}^{ML}) = \sigma^2 (X^T X)^{-1}$

Note; if the noise is i.i.d. gaussian then our $\hat{\beta} \sim N(\beta, \sigma^2 (X^T X)^{-1})$

unbiased estimate is variable w/ σ^2.

The bigger σ^2, the bigger the noise of ys, the bigger the covariance of $\hat{\beta}$. The larger the noise the more uncertain we are of β. our estimate becomes worse.

(1.3) Consider now a new sample $(x_0, y_0) \in \mathbb{R}^d \times \mathbb{R}$ outside the data set, for which: $y_0 = x_0^T \beta + \varepsilon_0$
where $\varepsilon_0 \sim N(0, \sigma^2)$, and is independent of all the noise variables $\{\varepsilon_i\}_{i=1}^n$ in D. Show that the expectation prediction error (EPE) of estimate $\hat{y}_0 = x_0^T \hat{\beta}^{ML}$ is given by: $\mathbb{E}[(y_0 - \hat{y}_0)^2] = \sigma^2 + x_0^T \text{cov}(\hat{\beta}^{ML}) x_0$

EPE of $\hat{y}_0 = x_0^T \hat{\beta}^{ML}$ comes from our noise ε_0 and we dont know β so we estimate it, $\hat{\beta}^{ML}$. ε_0 is irreducible error $\hat{\beta}^{ML}$ has estimation error.

$x_0 \in \mathbb{R}^d$, $y_0 = x_0^T \beta + \varepsilon_0$, $\mathbb{E}[\varepsilon_0] = 0$ $\mathbb{E}[\varepsilon_0^2] = \sigma^2$, ε_i are i.i.d
$\hat{y}_0 = x_0 \hat{\beta}^{ML}$

So, $\mathbb{E}[(y_0 - \hat{y}_0)^2] = \mathbb{E}[(y_0 - x_0^T \hat{\beta}^{ML} + x_0^T \hat{\beta}^{ML} - \hat{y}_0)^2]$

$= \mathbb{E}[\underbrace{(y_0 - x_0^T \beta)^2}_{A} + \underbrace{(x_0 \beta^T - \hat{y}_0)^2}_{B} + \underbrace{2(y_0 - x_0^T \beta) \cdot (x_0^T \beta - \hat{y}_0)}_{C}]$

$A = (y_0 - x_0^T \beta)^2 = \varepsilon_0^2$
$\mathbb{E}[A] = \mathbb{E}[\varepsilon_0^2] = \sigma^2$

$C = (y_0 - x_0^T \beta)(x_0^T \beta - \hat{y}_0) = \varepsilon_0 (x_0^T \beta - \hat{y}_0)$
$= \varepsilon_0 (x_0^T \beta - x_0^T \hat{\beta}^{ML})$, $\hat{\beta}^{ML}$ is random since our data is random
$\mathbb{E}[C] = \mathbb{E}[\varepsilon_0] \mathbb{E}(x_0^T \beta - x_0^T \hat{\beta}^{ML}) = 0 \cdot 0 = 0$

for gati vectors $x^T y = y^T x$

$B = (x_0 \beta^T - \hat{y}_0)^2$
$\mathbb{E}[B] = \mathbb{E}[(x_0^T \beta - x_0^T \hat{\beta}^{ML})(\beta^T x_0 - \hat{\beta}^{ML T} x_0)]$
$= \mathbb{E}[x_0^T (\beta \cdot \hat{\beta}^{ML})(\beta^T - \hat{\beta}^{ML}) x_0]$
$= x_0^T \mathbb{E}[(\beta - \hat{\beta}^{ML})(\beta - \hat{\beta}^{ML})^T] x_0$
$= x_0^T \mathbb{E}[(\hat{\beta}^{ML} - \beta)(\hat{\beta}^{ML} - \beta)^T] x_0$
$= x_0^T \text{cov}(\hat{\beta}^{ML}) x_0$

note: Estimator and Covariance is undefined if $\text{rank}(X) < d$

$\mathbb{E}[(y_0 - \hat{y}_0)^2] = \mathbb{E}[A + B + C] = \sigma^2 + x_0^T \text{cov}(\hat{\beta}^{ML}) x_0$

off by, error noise + estimation error

Homework 3

Q2

Consider the exact same setting as Q1, but suppose now that we further assume that $\beta \sim N(0, \frac{\sigma^2}{\lambda} I)$, i.e. that the model β is sampled from a Gaussian prior centered at zero, with a covariance that depends on $\lambda > 0$.

(2.1) What information does the prior capture / what does it tell us about our prior belief on β as $\lambda \to \infty$?

As $\lambda \to \infty$, the variance (σ^2/λ) becomes very small which indicates a high confidence in our mean (0). It tells us our prior belief on β is low; that there is low correlation between X and Y, (our features and classes).

(2.2) What information does the prior capture / what does it tell us about our prior belief on β as $\lambda \to 0$? What did we call such a prior in previous lectures?

As $\lambda \to 0$ the variance (σ^2/λ) becomes very large and approaches the uninformative prior. It would be uninformative if all of our coefficients in β were 1. But it indicates a high correlation between all of our features and the classifications.

(2.3) Show that the maximum a posteriori estimate corresponds to ridge regression i.e.:

$$\hat{\beta}^{MAP} \equiv \arg\max_{\beta \in \mathbb{R}^d} P(\beta|D) = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \lambda \|\beta\|_2^2$$

$$\arg\max_{\beta} P(\beta|D) = \arg\max_{\beta \in \mathbb{R}^d} P(D|\beta) P(\beta)$$

$$= \arg\min_{\beta \in \mathbb{R}^d} -\log P(D|\beta) - \log P(\beta)$$

$$= \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \lambda \|\beta\|_2^2$$

(2.4) Where does $\hat{\beta}^{MAP}$ converge to as $\lambda \to \infty$?

For a gaussian prior centered around zero, the variance would shrink very small so you would have a β super concentrated around zero.

(2.5) Where does $\hat{\beta}^{MAP}$ converge to as $\lambda \to 0$? Contrast this to what happens to the prior: do we see a phenomenon that we have also observed in other parameter estimation tasks?

$\hat{\beta}^{MAP}$ converges to the uninformative prior and then we also see that $\hat{\beta}^{MAP} \equiv \hat{\beta}^{ML}$.

this is also apparent when we did our proof before, and I saw,

$$\hat{\beta}^{ridge} = \hat{\beta}^{MAP} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \lambda \|\beta\|_2^2$$

as $\lambda \to 0$ we have

$$\arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + 0$$

which we recognize from before as the maximum likelihood estimate, $\hat{\beta}^{ML}$

So we can see ridge regression as an interpolation between map and MLE and λ as an interpolation between an uninformative prior and a prior that implements occams razor (one that implements the KISS principle → prefer it simple)

Homework 3

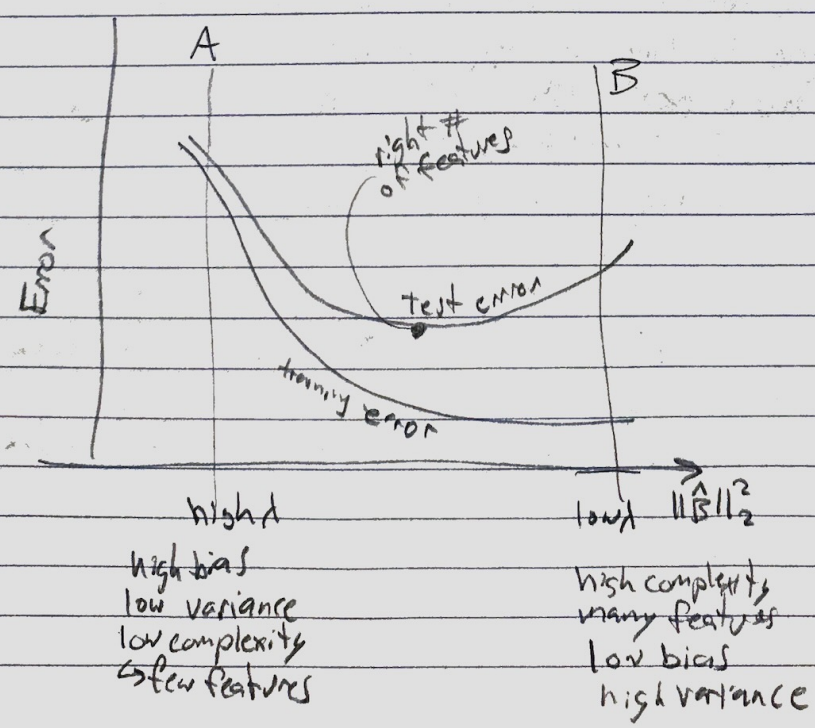
Q3

(3.1) For what value of λ is the trainy error minimized and why?

$\lambda = 0$, Trainy error is minimized at $\lambda = 0$ since it corresponds to the most complex model where the model is fitted to the training data set's noise. Also the error is $(y_i - \beta^T x_i)^2 + \lambda \|\beta\|_2^2$, so $\lambda = 0$ minimizes the error in that equation since $\lambda \|\beta\|_2^2$ would become zero.

(3.2) How does the norm of the estimated model $\|\hat{\beta}\|_2$ change as we increase λ and why?

As we increase λ, the norm decreases as seen on the graph. More features are zero so the model is less complex and the norm of the estimated model is smaller.



(3.3) Between points A and B, which one corresponds to a model trained with a higher value of λ? A

(3.4) Btwn A, B, which are we learning more complex model, more parameters away from 0? B

(3.5) Btwn A, B which has higher estimation error? B

(3.6) higher model error / bias? A

(3.7) Which is overfitting the training set? B

(3.8) If we increase the number of samples in the training set, would the test error curve change and how? Model would be better so test error minimum would decrease. More Samples will decrease the error more near B than A. Near A the model is simple, has high bias and more data will not fix that but near B, the model error is low and the model will be better when trained by more data. More data will inform the model when trained, which coefficients are zero such as Zipcode when considering blood pressure, so the test error will decrease.

(3.9) If we decrease the noise variance σ^2, would the test error curve change? and how? If you decrease the noise the model will be trained more accurately so the test error value decreases, and be more closely fitted to the training error.

Homework 3

Q4

(4.1) Provide the mathematical formula for function fun in generate_data.py. What coordinates of vector $x \in \mathbb{R}^5$ does its output depend on? Is it a linear function of $x \in \mathbb{R}^5$?

$$fun(x) = 1.3 + 2(X_0) - 1.1(X_1) + 0.7(X_2) + 1.2(X_3) + 0.4(X_0^2) - 1.5(X_1)(X_3) - 0.7(X_4^2)$$

The output of fun(x) depends on X_0, X_1, X_2, X_3, X_4

It is not a linear function since it has terms X_0^2, X_4^2

(4.2) Run code on generated data with $N = 1000$, $d = 5$, $\sigma = 0.01$. fr = 10% → 100% 10% increments. Plot train + test RMSE vs # of train samples, given to the model. Does increasing the number of samples significantly reduce either errors? Do you think that increasing the number of samples beyond 1000 would help in significantly lowering either error? If not, why?

Increasing the number of samples does not significantly reduce either error, so no increasing the number of samples beyond 1000 would not help to significantly lower either error. The model is linear, but the generated data is from a polynomial, so the model error is going to always outweigh the noise of the data, so more data will not fix the model error.

☆ see plot

4.2


Training and Test RMSE vs Training Set Size
Train RMSE
Test RMSE
RMSE
2.1
2.0
1.9
1.8
1.7
100
200
300
400
500
600
700
Training Set Size: number of training samples

(4.3) Run code on a dataset generated by generate_data.py with N=1000, d=40, σ=0.01. Note, increasing input vector adds variables that do not affect labels; often called nuisance or redundant variables. Plot train and test RSME vs training sample size. Do you see any differences from Q4.2? Explain what you think is happening to cause this behavior. Do you think that increasing the number of samples beyond 1000 would significantly reduce the test error?

The main difference from Q4.2 plot is the test error increased, also the plots are less scattered and settle quickly to an error/RMSE. Since the model is trained on more categories $X_5 - X_{39}$ but the generated data has not changed the function or variables the function depends on $(X_0 - X_4)$ then the model is taking into account random data that does not actually matter. So this introduces more noise. $\beta_5 - \beta_{39}$ does decrease with more samples but the model is still trying to fit linear to a polynomial so the bulk of the error is still there and will not be fixed with more samples, so beyond 1000 would not significantly reduce test error.

☆ 4.3 see plot

4.3


Training and Test RMSE vs Training Set Size (Data from _N_1000_d_40_sig_0_01)
Train RMSE
Test RMSE
RMSE
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
100
200
300
400
500
600
700
Training Set Size: number of training samples

Homework 3

(4.4) see code for def lift(x_initial)
which expands or 'lifts' x to x' (vectors)
by appending x_i and $x_i \cdot x_j$ for all $i \geq j$ to x'

(4.5) Show that function: fun() can be written as a linear function of such a lifted vector x', i.e. a function of the form: $f(x') = \beta^T x' + c$
for appropriately defined vector $\beta \in \mathbb{R}^{d'}$ and constant $c \in \mathbb{R}$, and provide β and c.

$$fun(x) = 1.3 + 2(x_0) - 1.1(x_1) + 0.7(x_2) + 1.2(x_3) + 0.4(x_0^2) - 1.5(x_1)(x_3) - 0.7(x_4^2)$$

$$x' = [x_0, x_1, x_2, x_3, x_4,$$
$$x_0^2, x_1x_0, x_1^2, x_2x_0, x_2x_1,$$
$$x_2^2, x_3x_0, x_3x_1, x_3x_2, x_3^2,$$
$$x_4x_0, x_4x_1, x_4x_2, x_4x_3, x_4^2]$$

$$f(x') = 2(x'_0) - 1.1(x'_1) + 0.7(x'_2) + 1.2(x'_3) + 0.4(x'_5) - 1.5(x'_{13}) - 0.7(x'_{20}) + 1.3$$

$f(x') = \beta^T x' + c$ where $c = 1.3$ and,

$$\beta^T = [2, -1.1, 0.7, 1.2, 0,$$
$$0.4, 0, 0, 0, 0,$$
$$0, 0, -1.5, 0, 0,$$
$$0, 0, 0, 0, -0.7]$$

(4.6) My code for lift(x_initial) is actually a step ahead and does what liftDataset should do.

(4.7) Lift the X's in the data set by adding function to LinearRegressionSweepN w/ N=1000, d=5, σ=0.01. How does this plot differ from previous plots you generated? Why? Obtain the parameters (coefficients and intercept) of your finally trained model. Did you manage to learn function fun?

This plot converges to RMSE≈0.01 with test RMSE slightly higher. this is a much better result than before since with the lifted parameters we can now have an accurate model and decrease the model error expeadly with larger data sets.

The script was able to learn the function fun with the coefficients only ± 0.001 off of the actual values.

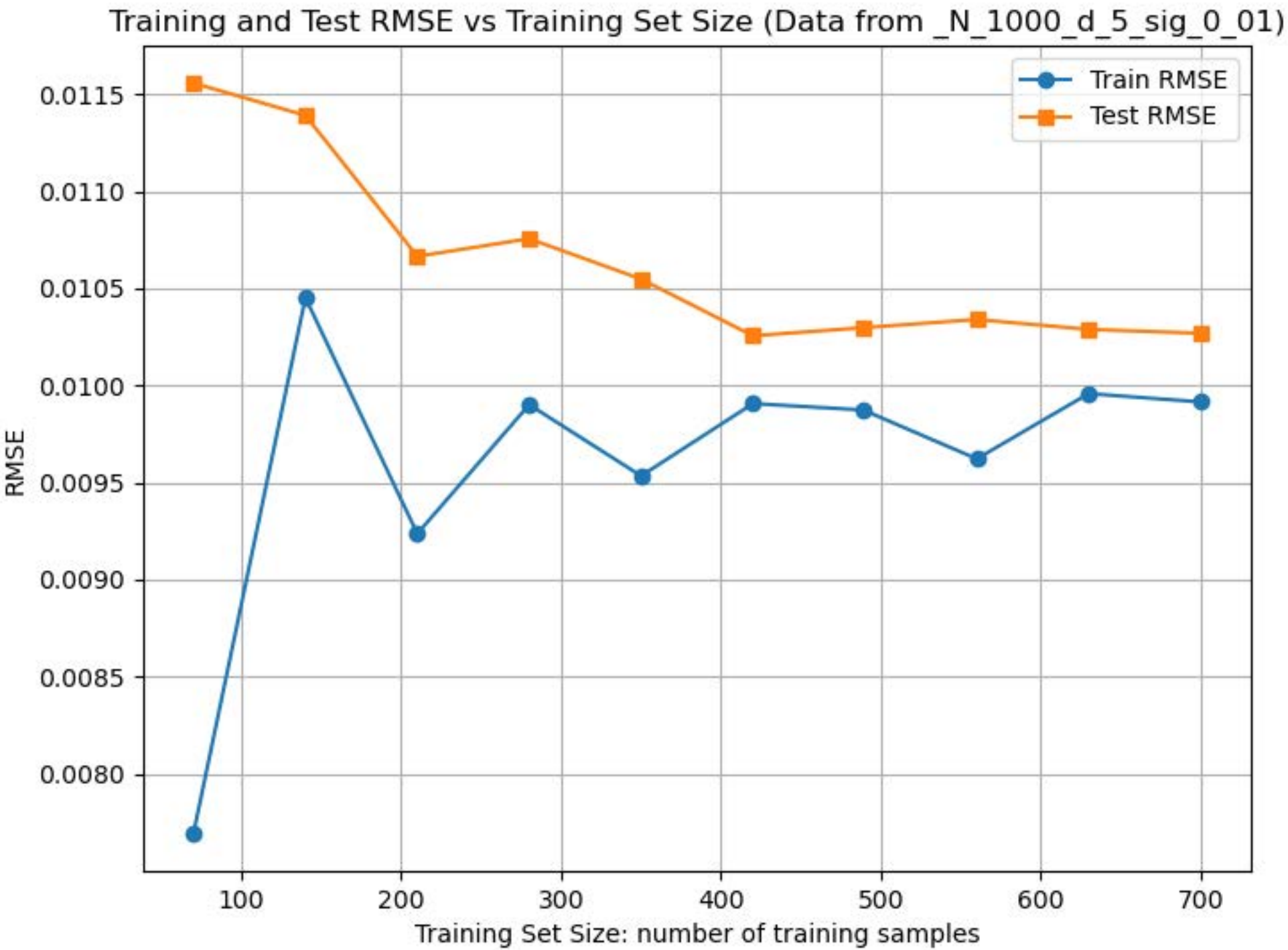
cx $\beta_0 = 1.99994 \approx 2.0$ ✡ seeplot

(4.8) Repeat but with d=40. How does the training and test error change as you increase the number of training samples? Obtain the parameters (coefficients and intercept) of your finally trained model. Did you manage to learn function fun? Do you think that increasing the number of samples beyond 1000 would improve the test error / the proximity of your learned model to the true fun fun?

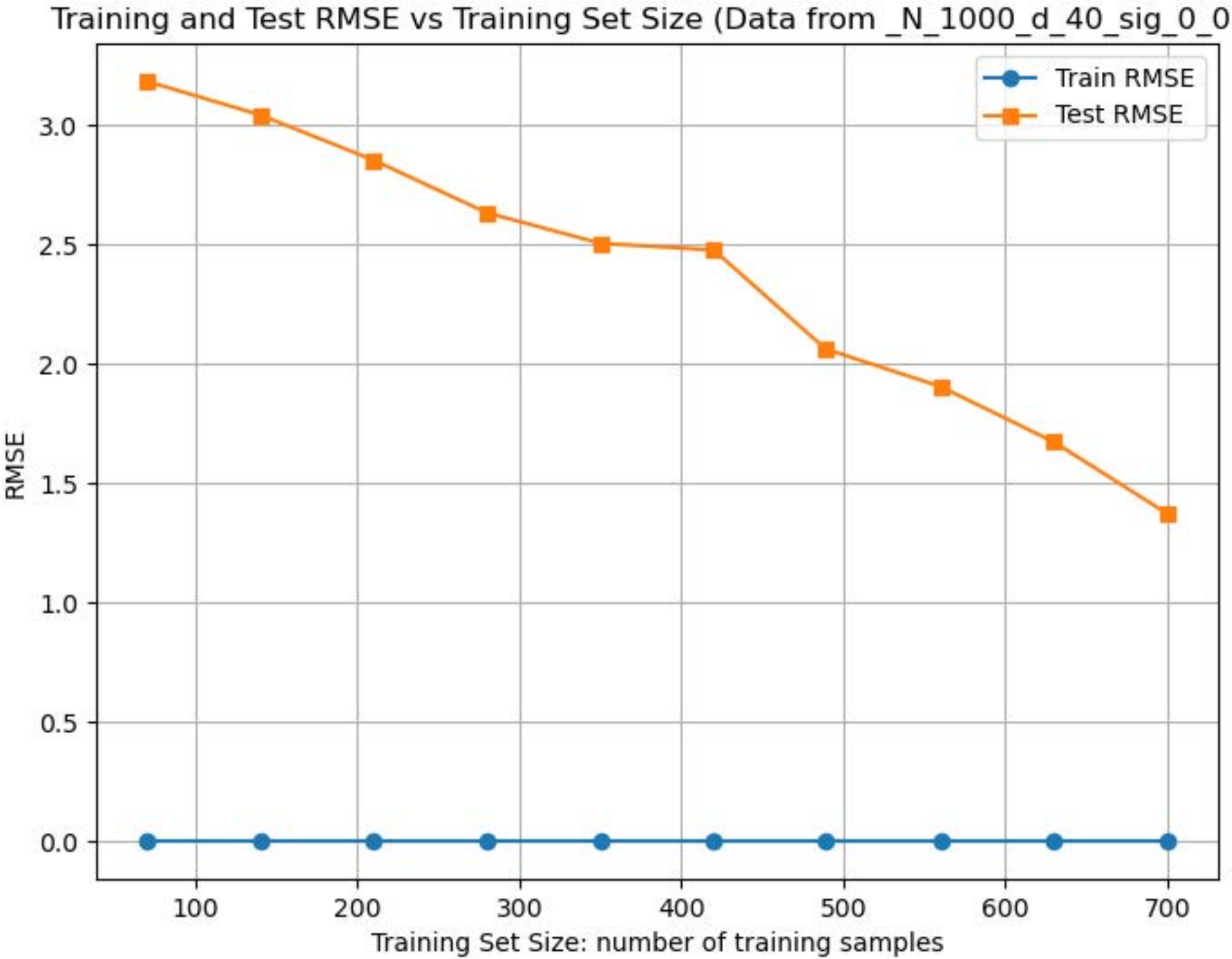
The training error increases with increasing training samples but, The training error is very low compared to the test error. The test error decreases as you increase the number of training samples. The script got close to learning the function fun, off by about ± 0.3 for the coefficients, with the zero coefficients also off, not quite zero. Yes, increasing the number of samples beyond 1000 would improve the test error as seen in the trend already, and made possible by better matching the true function fun, so the proximity of the learned model would improve. ✡ see plots

$\epsilon^3 = 1.7 \approx 2.0$

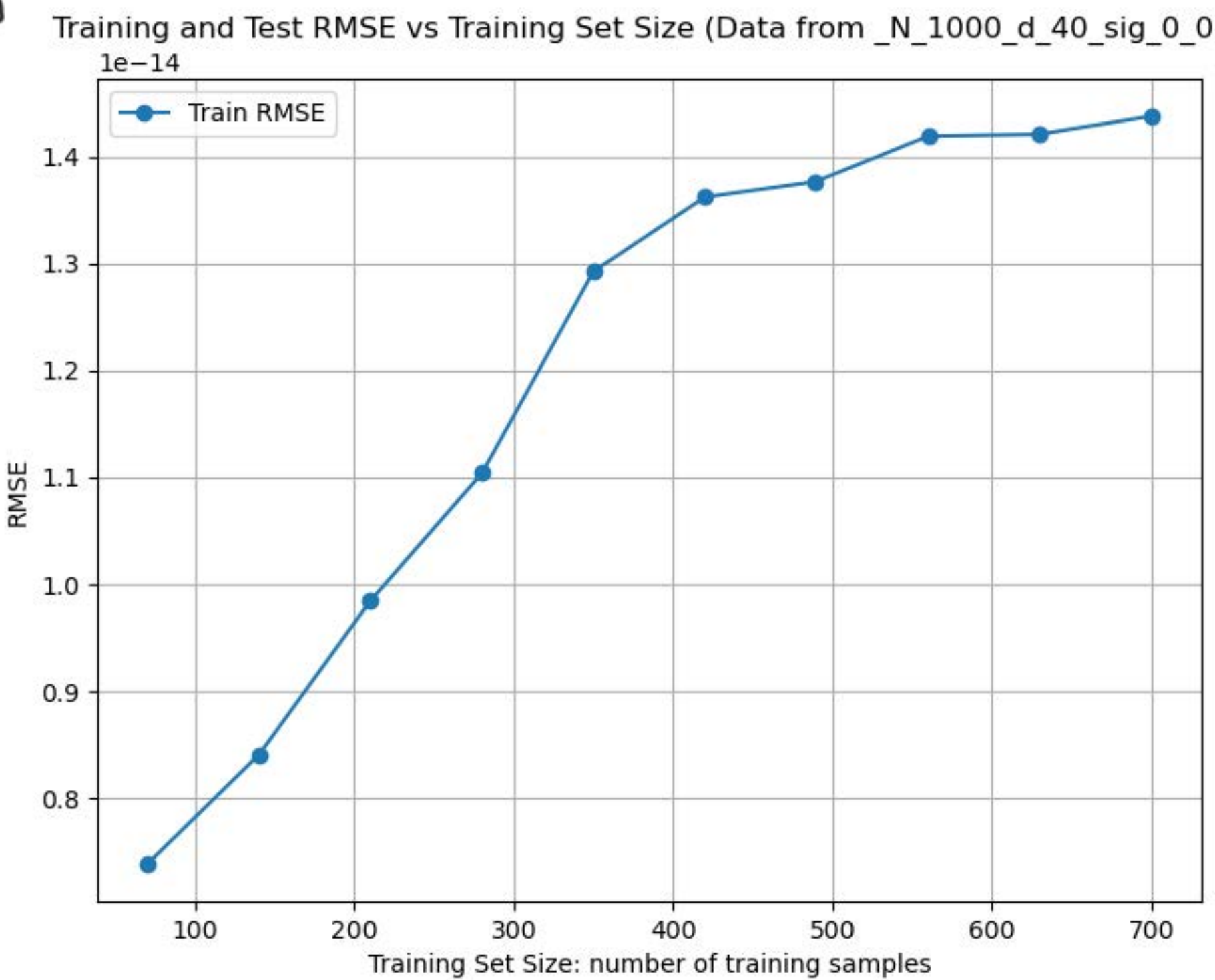
4.7


Training and Test RMSE vs Training Set Size (Data from _N_1000_d_5_sig_0_01)
Train RMSE
Test RMSE
RMSE
0.0115
0.0110
0.0105
0.0100
0.0095
0.0090
0.0085
0.0080
100
200
300
400
500
600
700
Training Set Size: number of training samples

4.8


Training and Test RMSE vs Training Set Size (Data from _N_1000_d_40_sig_0_01)
Train RMSE
Test RMSE
RMSE
3.0
2.5
2.0
1.5
1.0
0.5
0.0
100
200
300
400
500
600
700
Training Set Size: number of training samples

4.8


Training and Test RMSE vs Training Set Size (Data from _N_1000_d_40_sig_0_01)
1e−14
Train RMSE
RMSE
1.4
1.3
1.2
1.1
1.0
0.9
0.8
100
200
300
400
500
600
700
Training Set Size: number of training samples

Bonus Question

, LassoCV.py?

1. Read through the code in file Lassop, and explain what it does.
It loads the X, y data the same, and splits it into test/train the same
assigns alpha = 0.1 , It also imports Lasso, from sklearn model

Creates a lasso model w/ alpha, whose alpha acts as our λ on L_1 norm
assigns cv to a KFold with 5 splits, and shuffle as true

creates a score variable using cross_val_score function with variables / inputs, model, X_train, y_train. and sets cv to cv, and scoring to the negative RMSE.

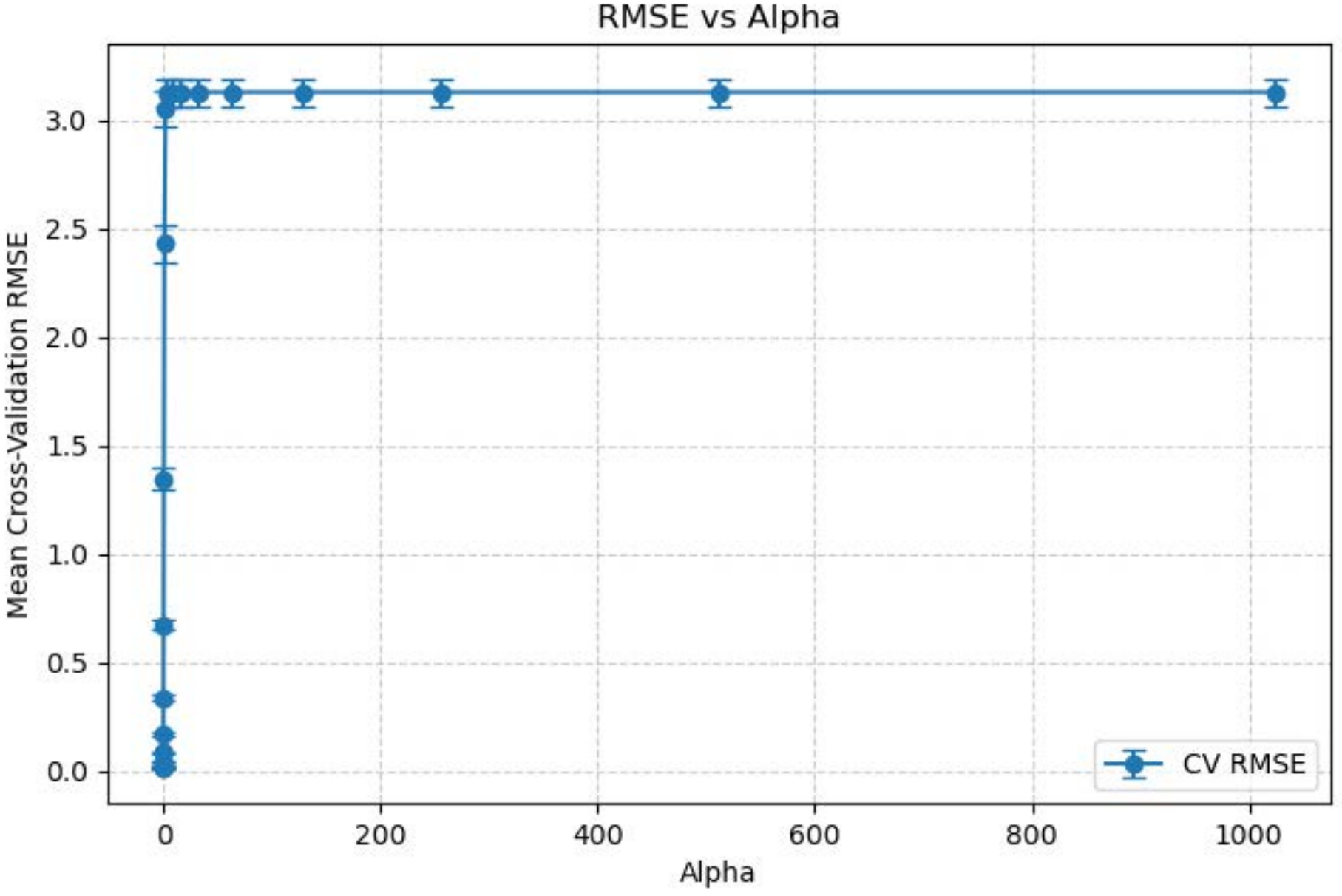
prints, cross-validation RMSE for α = alpha value and then the negative of the average of the scores as well as the standard deviation of the scores.
tells you when its done fitting the linear model.
and fits the X training and y training data to the lasso model

Finally computes the RMSE for the train and test classifications with inputs of y and the models predictions of ys to see how far off the model is.

and Finally finally prints the training and testing RSME

Also Cross_val_score splits the dataset X and y into cross validation folds cv = n folds, the estimator is trained and the score is recorded. After doing it cv times, cross_val_score returns an array of scores one for each fold. Then we can calculate the mean and std to get an overall estimate of the models performance.

bonus


RMSE vs Alpha
Mean Cross-Validation RMSE
Alpha
CV RMSE
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000

(Bonus 3) Train RMSE = 0.009983
Test RMSE = 0.00188
best α = 0.000977
β_0 = 1.99912 β_1 = 0.39983 all close to fun

Yes! it was better since kfold and sweeping was able to determine which variables (coeffs) should be zero.

see plot

Git repo @ barker_ch
IML
HW3

```python
import numpy as np

def fun(x):
    """
        Compute a simple quadratic function f:R^5 -> R.

    """

    return 1.3 + 2.*x[0] - 1.1*x[1] + 0.7*x[2]  + 1.2*x[3] + \
            0.4* (x[0]**2) -1.5 *x[1]*x[3] - 0.7 * (x[4]**2)

def postfix(N,d,sigma):
    """Converts parameters into a handy string, to be appended to file names."""
    return "_N_%d_d_%d" % (N,d) \
         + "_sig_%s" % str(sigma).replace(".","_")


if __name__ == "__main__":
    # Set random seed for reproducibility
    np.random.seed(42)


    # Number of samples
    N = 1000

    # Noise variance
    sigma = 0.01


    # Feature dimension
    d = 40

    print("Generating dataset with N = %d, σ = %f, d = %d..." % (N,sigma,d), end="")

    # Generate random features
    X = np.random.randn(N, d)

    # Generate pure labels
    y = []

    for i in range(N):
        y.append(
                    fun(X[i,:])
```

```
                    )
    y = np.array(y)

    # Add noise to labels
    err = np.random.normal(scale = sigma, size = N)
    y = y + err

    print(" done")

    psfx = postfix(N,d,sigma)

    print("Saving X and y... ",end="")
    np.save("X" + psfx,X) #saves X with name, X N d sigma
    np.save("y" + psfx,y) #saves y with name, y N d sigma
    print(" done")
```

```python
import numpy as np
from mpmath.matrices.eigen_symmetric import c_he_tridiag_0
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import root_mean_squared_error as rmse
from data_generator import postfix



# Number of samples
N = 1000

# Noise variance
sigma = 0.01

# Feature dimension
d = 40


psfx = postfix(N,d,sigma)

# function lift = liftDataset
def lift(x_initial):
    x_prime = []

    xi = len(x_initial)
    for row in range(xi):
        x_expand = []
        x_0 = x_initial[row]

        l = len(x_0)
        for i in range(l):
            x_expand.append(x_0[i])
            for j in range(i,l):
                x_expand.append(x_0[j]*x_0[i])

        x_prime.append(x_expand)

    return np.array(x_prime)
# test lift function
# x_initials = [[1,2,3,4,5],
#               [2,4,6,8,10]]
# new_x_i =lift(x_initials)
# print(new_x_i)
```

```
46
47
48
49 X = np.load("X"+psfx+".npy")
50 y = np.load("y"+psfx+".npy")
51
52
53
54 print("Dataset has n=%d samples, each with d=%d features," % X.
   shape,"as well as %d labels." % y.shape[0])
55 frac = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
56 rmse_train_array = []
57 rmse_test_array = []
58 training_length_array = []
59 X = lift(X)
60
61 for i, fr in enumerate(frac):
62     # ... use both i and frac
63     print(f"Training with {fr*100}% of the training data")
64     #Split data: test set made up of fraction of N data (ex 0.30*
   1000 = 300)
65     testSize = 0.30
66     X_train, X_test, y_train, y_test = train_test_split(
67         X, y, test_size=testSize, random_state=42)
68
69     print("Randomly split dataset to %d training and %d test
   samples" % (X_train.shape[0],X_test.shape[0]))
70     print("Training with number of n samples = %d" % int(len(
   X_train) * fr))
71
72     ### select only a fraction (ex:10%) of the test data (X_train
    y_train)
73     #fr = 0.1 # fraction of test set (ex 10%)
74     # Calculate the number of elements to select (ex:10% of 1000)
75     num_elements_to_select = int(len(X_train) * fr)
76
77     # Generate random indices
78     random_indices = np.random.choice(len(X_train),
   num_elements_to_select, replace=False)
79
80     #print("Original X array:", X_train)
81     #print("Original y array:", y_train)
82
83     # Select the elements using the random indices
84     X_train = X_train[random_indices]
```

```
 85      y_train = y_train[random_indices]
 86
 87      #print(f"Selected {fr} of X data:", X_train)
 88      #print(f"Selected {fr} of y data:", y_train)
 89
 90      ###
 91
 92
 93      model = LinearRegression()
 94
 95      print("Fitting linear model...",end="")
 96      model.fit(X_train, y_train)
 97      print(" done")
 98
 99
100      # Compute RMSE on train and test sets
101      rmse_train = rmse(y_train,model.predict(X_train))
102      rmse_test = rmse(y_test,model.predict(X_test))
103      rmse_train_array.append(rmse_train)
104      rmse_test_array.append(rmse_test)
105      training_length_array.append(len(y_train))
106
107      print("Train RMSE = %f, Test RMSE = %f" % (rmse_train,
     rmse_test))
108
109
110      print("Model parameters:")
111      print("\t Intercept: %3.5f" % model.intercept_,end="")
112      for i,val in enumerate(model.coef_):
113          print(", β%d: %3.5f" % (i,val), end="")
114      print("\n")
115      #print(f'x1',X_test)
116
117  #Plot the train and test RMSE as a function of the number of
     training samples given to model
118
119  plt.figure(figsize=(8, 6))
120  plt.plot(training_length_array, rmse_train_array, marker='o',
     label='Train RMSE')
121  #plt.plot(training_length_array, rmse_test_array, marker='s',
     label='Test RMSE')
122  plt.xlabel("Training Set Size: number of training samples")
123  plt.ylabel("RMSE")
124  plt.title(f"Training and Test RMSE vs Training Set Size (Data
     from {psfx})")
```

```
125 plt.legend()
126 plt.grid(True)
127 plt.show()
128
129
130
131
```

```
 1 import numpy as np
 2 from sklearn.model_selection import KFold,train_test_split,
   cross_val_score
 3 from sklearn.linear_model import Lasso
 4 from sklearn.metrics import root_mean_squared_error as rmse
 5 import matplotlib.pyplot as plt
 6 from data_generator import postfix
 7
 8
 9 # Number of samples
10 N = 1000
11
12 # Noise variance
13 sigma = 0.01
14
15
16 # Feature dimension
17 d = 40
18
19 psfx = postfix(N,d,sigma)
20
21 # function lift = liftDataset
22 def lift(x_initial):
23     x_prime = []
24
25     xi = len(x_initial)
26     for row in range(xi):
27         x_expand = []
28         x_0 = x_initial[row]
29
30         l = len(x_0)
31         for i in range(l):
32             x_expand.append(x_0[i])
33             for j in range(i,l):
34                 x_expand.append(x_0[j]*x_0[i])
35
36         x_prime.append(x_expand)
37
38     return np.array(x_prime)
39
40
41 X = np.load("X"+psfx+".npy")
42 y = np.load("y"+psfx+".npy")
43 X = lift(X)
44
```

```
45 print("Dataset has n=%d samples, each with d=%d features," % X.
   shape,"as well as %d labels." % y.shape[0])
46
47 X_train, X_test, y_train, y_test = train_test_split(
48     X, y, test_size=0.30, random_state=42)
49
50 print("Randomly split dataset to %d training and %d test samples"
    % (X_train.shape[0],X_test.shape[0]))
51
52
53 # alpha = 0.1
54 #
55 # model = Lasso(alpha = alpha)
56 #
57 # cv = KFold(
58 #         n_splits=5,
59 #         random_state=42,
60 #         shuffle=True
61 #         )
62 #
63 #
64 #
65 # scores = cross_val_score(
66 #         model, X_train, y_train, cv=cv,scoring="
   neg_root_mean_squared_error")
67 #
68 #
69 # print("Cross-validation RMSE for α=%f : %f ± %f" % (alpha,-np.
   mean(scores),np.std(scores)) )
70
71 # === New code: scan alpha values and pick the best one ===
72
73 alphas = 2.0 ** np.arange(-10, 11)  # from 2^-10 to 2^10
74 cv = KFold(n_splits=5, random_state=42, shuffle=True)
75
76 mean_rmse = []
77 std_rmse = []
78
79 print("Performing 5-fold CV for multiple α values...")
80
81 for alpha in alphas:
82     model = Lasso(alpha=alpha, max_iter=10000, random_state=42)
83     scores = cross_val_score(
84         model, X_train, y_train,
85         cv=cv,
```

```
 86         scoring="neg_root_mean_squared_error"
 87     )
 88
 89     mean_rmse.append(-np.mean(scores))
 90     std_rmse.append(np.std(scores))
 91     print("α = %8.5f -> CV RMSE = %.6f ± %.6f" % (alpha, -np.
    mean(scores), np.std(scores)))
 92
 93 # Find best alpha (lowest mean RMSE)
 94 best_idx = np.argmin(mean_rmse)
 95 best_alpha = alphas[best_idx]
 96
 97 print("\nBest α found: %f (CV RMSE = %.6f)" % (best_alpha,
    mean_rmse[best_idx]))
 98
 99 model = Lasso(alpha=best_alpha, max_iter=10000, random_state=42)
100 print("Fitting linear model over entire training set...",end="")
101 model.fit(X_train, y_train)
102 print(" done")
103
104
105 # Compute RMSE
106 rmse_train = rmse(y_train,model.predict(X_train))
107 rmse_test = rmse(y_test,model.predict(X_test))
108
109 print("Train RMSE = %f, Test RMSE = %f" % (rmse_train,rmse_test
    ))
110
111
112
113 # Plot CV mean RMSE as a function of alpha with error bars
114 plt.figure(figsize=(8, 5))
115 plt.errorbar(alphas, mean_rmse, yerr=std_rmse, fmt='-o', capsize
    =4, label='CV RMSE')
116 plt.xlabel('Alpha')
117 plt.ylabel('Mean Cross-Validation RMSE')
118 plt.title('RMSE vs Alpha')
119 plt.grid(True, which='both', linestyle='--', alpha=0.6)
120 plt.legend()
121 plt.show()
122
123
124
125 print("Model parameters:")
126 print("\t Intercept: %3.5f" % model.intercept_,end="")
```

```
127 for i,val in enumerate(model.coef_):
128     print(", β%d: %3.5f" % (i,val), end="")
129 print("\n")
130
131
132
```