

# CS436/536: Introduction to Machine Learning

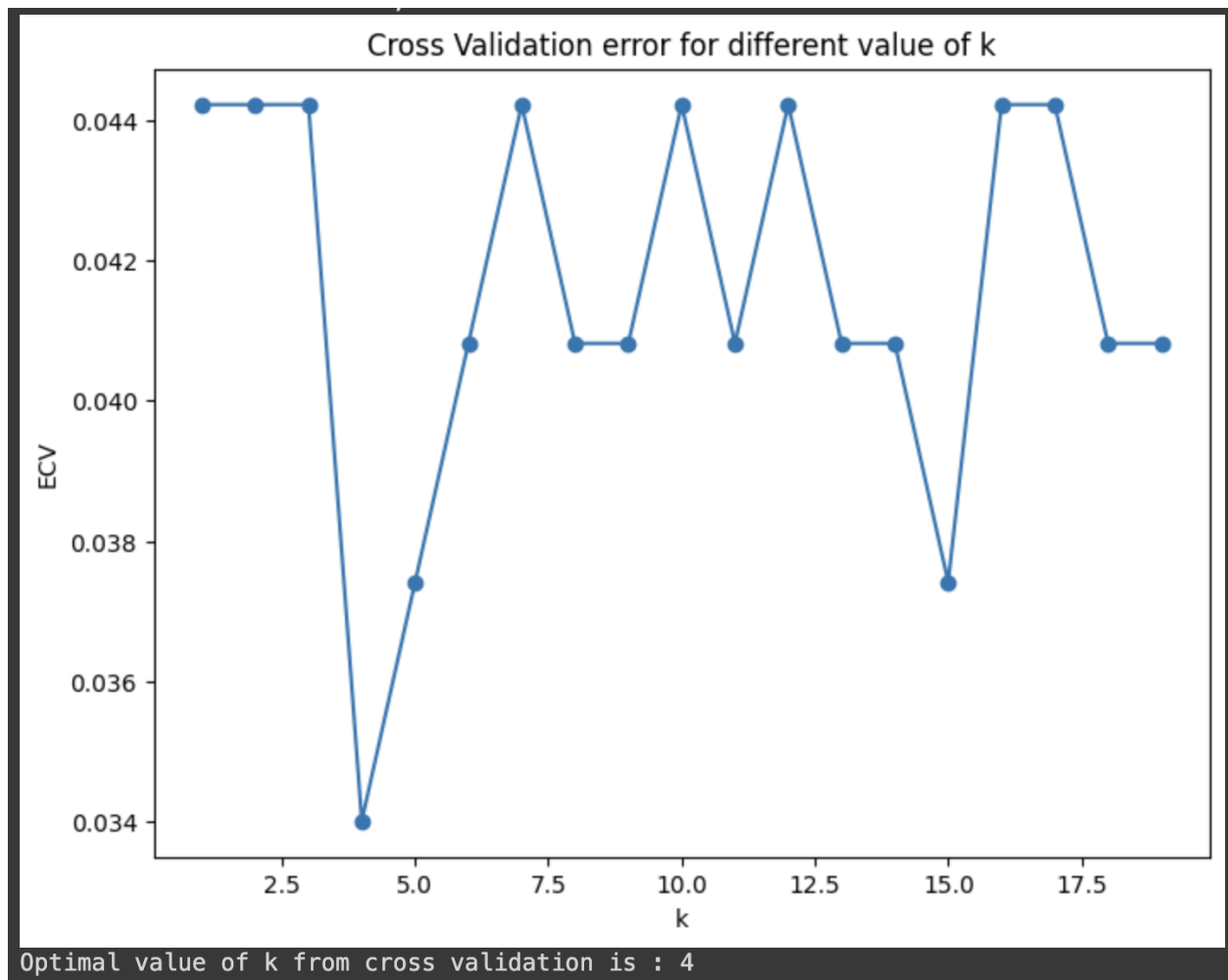
## Homework 4

### 1. [400 points] The k-NN rule.

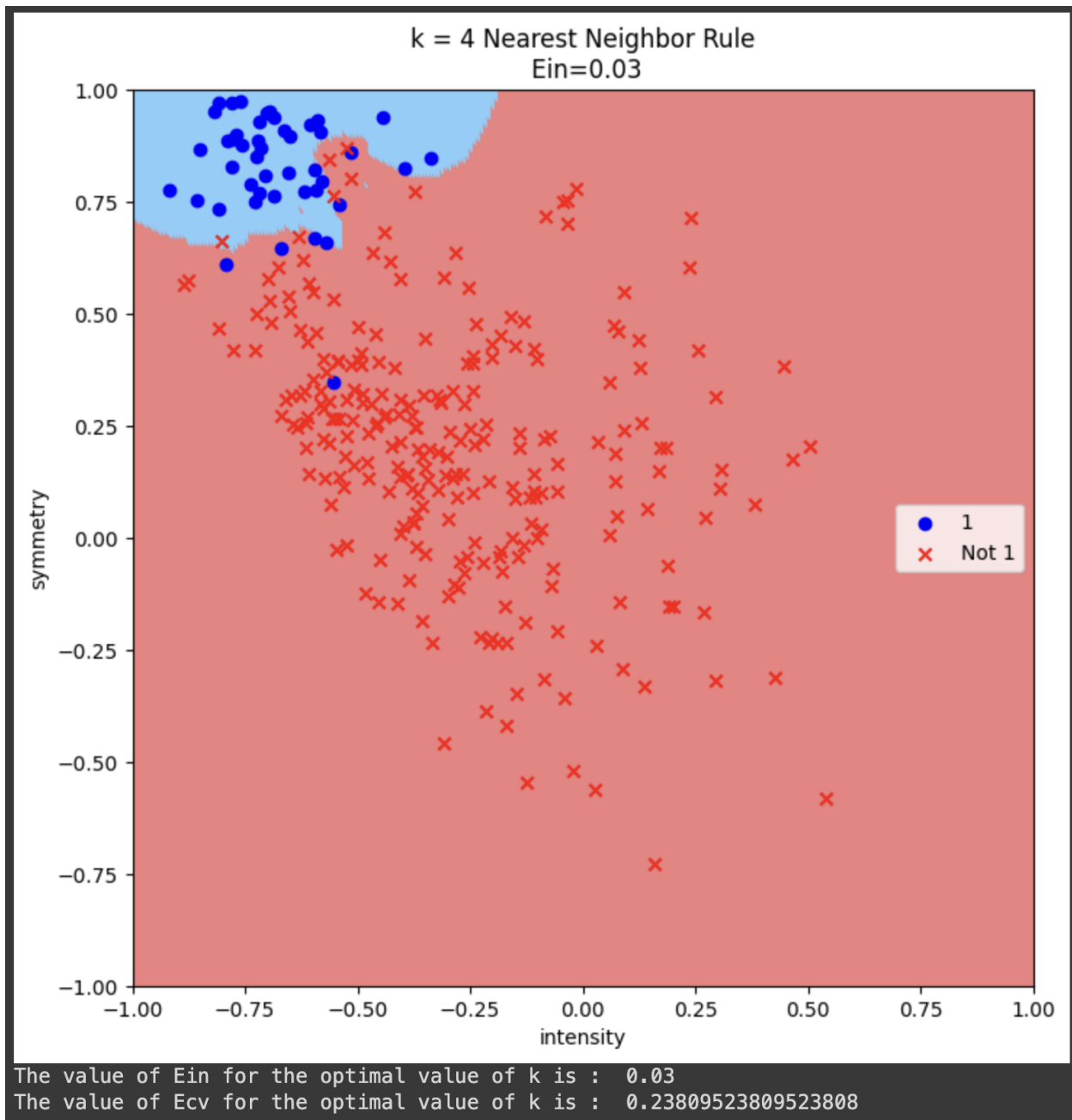
- Use cross validation with D to select the optimal value of k for the k-NN rule. Show a plot of ECV versus k and indicate the value of k you choose.
- For the chosen value of k, plot the decision boundary. Also compute and report its  $E_{in}$  and ECV.
- Report  $E_{test}$  for the k-NN rule corresponding to the chosen value of k.

### Solution:

a: The optimal values are found between the range of 1 to 20. The optimal value obtained is 4.



b: Plotting the decision boundary for the optimal solution of  $k$ , i.e. for  $k = 4$ . The calculated value of  $E_{in}$  is 0.03 and the calculated value of  $E_{cv}$  is 0.2380



C: Reporting  $E_{test}$  for the  $k$ -NN rule corresponding to the chosen value of  $k$ .

The value of  $E_{test}$  for the optimal value of  $k = 4$  is : 0.04223160702378306

**(2) [400 points] Neural Networks and Backpropagation.**

Implement neural networks and *stochastic* gradient descent using backpropagation for a neural network architecture specified by a vector  $[d^{(0)}, d^{(1)}, \dots, d^{(L)}]$ , where  $d^{(L)} = 1$ . Use  $\theta(s) = \tanh(s)$  as the transformation function for every node in a hidden layer. For the output layer, allow the user to specify whether to use  $\theta(s) = s$ ,  $\theta(s) = \tanh(s)$  or  $\theta(s) = \text{sign}(s)$ . Set the architecture to  $[2, m, 1]$ , i.e. 2 inputs or  $d^{(0)} = 2$ ,  $m$  hidden units, i.e.  $d^{(1)} = m$ , and 1 output node, i.e.  $d^{(L)} = 1$  and  $L = 2$ . Implement *stochastic* gradient descent on the squared error  $E_{\text{in}}(\mathbf{w}) = \frac{1}{4N} \sum_{n=1}^N (h(x_n; \mathbf{w}) - y)^2$ , and check your gradient computation as follows:

- (a) Use a network with  $m = 2$ . Set all weights to 0.25 and consider a dataset with 1 data point:  $x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ ;  $y_1 =$
1. For the output node using either  $\theta(s) = s$  or  $\theta(s) = \tanh(s)$ , obtain the gradient of  $E_{\text{in}}(\mathbf{w})$  using the backpropagation algorithm. Report this result and check its dimensionality, there must be as many numbers as parameters in this network.

Solution:

```
layer 1
[[0.25 0.25]
 [0.25 0.25]
 [0.25 0.25]]
gradient using backpropagation G[1] / ||G[1]||
[[-0.28867513 -0.28867513]
 [-0.28867513 -0.28867513]
 [-0.57735027 -0.57735027]]
layer 2
[[0.25]
 [0.25]
 [0.25]]
gradient using backpropagation G[2] / ||G[2]||
[[-0.68040574]
 [-0.51819303]
 [-0.51819303]]
```

- (b) Now, obtain the gradient numerically by perturbing each weight, one at a time, by 0.0001. Report this result. If the result is not similar to the previous result, there is likely something wrong with your backpropagation gradient computation.

Solution:

```
layer 1
[[0.25 0.25]
 [0.25 0.25]
 [0.25 0.25]]
gradient using numerical G[1] / ||G[1]||
[[-0.28867513 -0.28867513]
 [-0.28867513 -0.28867513]
 [-0.57735027 -0.57735027]]
layer 2
[[0.25]
 [0.25]
 [0.25]]
gradient using numerical G[2] / ||G[2]||
[[-0.68040574]
 [-0.51819303]
 [-0.51819303]]
```

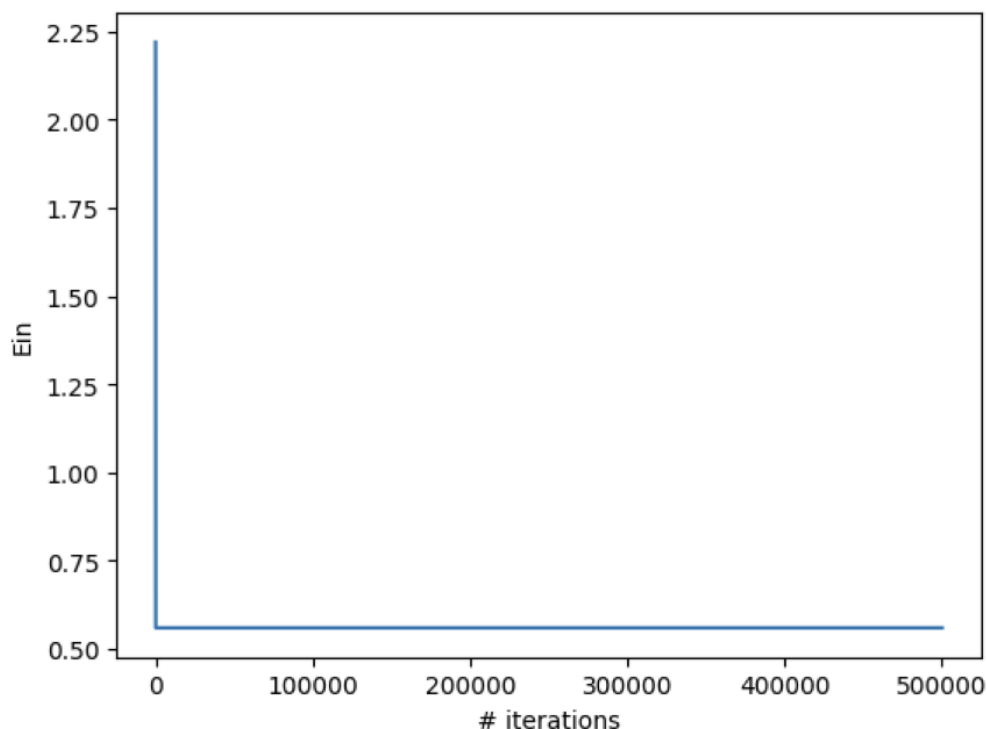
**(3) [600 points] Neural Networks for MNIST Digits Dataset.** Use your neural networks implementation with  $m = 10$  hidden units to build a classifier for classifying 1s vs. Not 1s. Use the two features and data you generated in previous homeworks and the 300 input data points you selected as the training set.

Randomly initialize the weights to small values. Then, use  $\theta(s) = \tanh(s)$  as the transformation function for the hidden units. During *training*, use the  $\theta(s) = s$  linear transformation function for the *output layer* only, i.e., use the regression for classification paradigm. Once the *training phase is complete*, switch the transformation function used in the *output layer* to the sign function, i.e., set  $\theta(s) = \text{sign}(s)$  to classify data points.

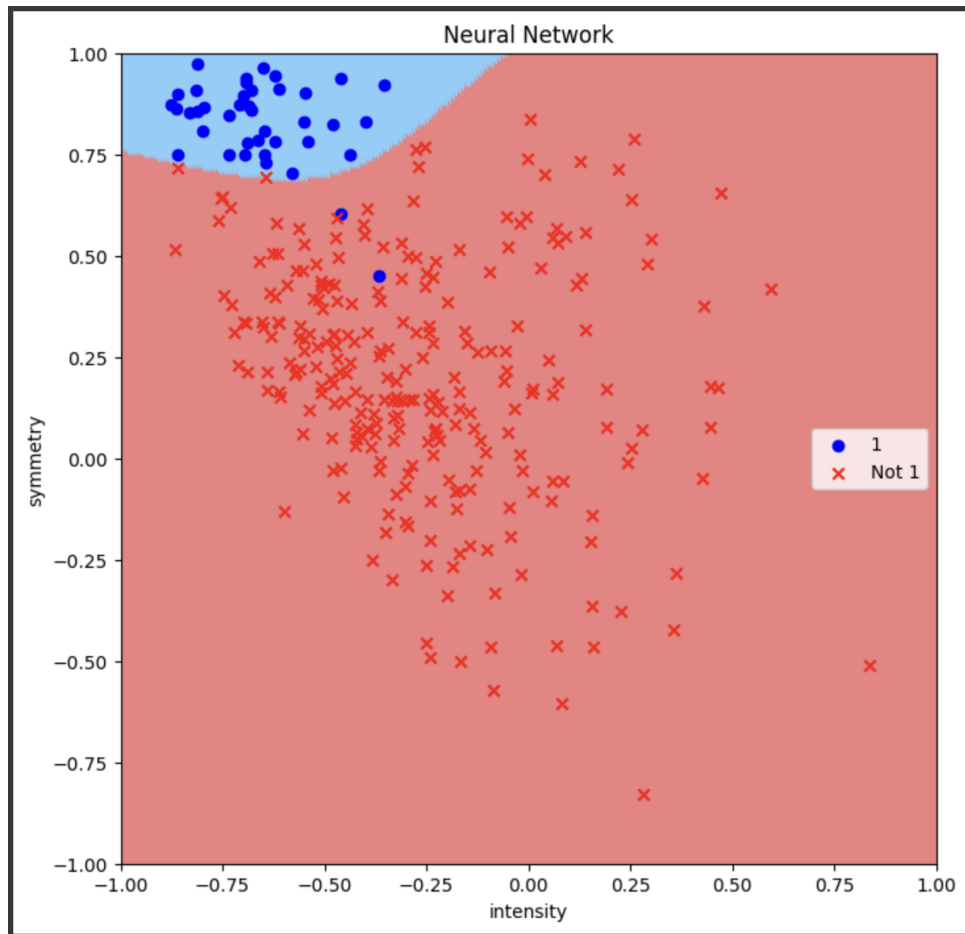
- Plot  $E_{\text{in}}(w)$  vs. iterations for the variable learning rate *stochastic* gradient descent heuristic and  $2 \times 10^6$  iterations. Plot the decision boundary for the resulting classifier.
- Now, use weight decay with  $\lambda = 0.01/N$  and use variable learning rate *stochastic* gradient descent to minimize the augmented error ( $E_{\text{aug}}$ ). Plot the decision boundary for the resulting classifier.
- Now, use early stopping with a validation set of size 50 and a training set of size 250. Plot the decision boundary for the resulting classifier that had minimum validation error ( $E_{\text{val}}$ ).

Solution:

a)



b)



**(4) [300 points] Compare Methods.** Now that we have implemented several algorithms to learn from data, it is time to reflect upon them. Compare the final test error from your attempts to solve the 1s vs. Not 1s classification problem on the MNIST Digits Dataset, namely:

- (i) Regularized Linear Model with 10-th order polynomial transform, where the regularization parameter  $\lambda$  is selected using cross validation (in HW3).
- (ii) The  $k$  Nearest Neighbors rule where  $k$  is selected using cross validation.
- (iii) Neural Networks with early stopping.

Report and summarize all of your results and make some intelligent observations about the results you obtained.

Solution:

Please refer below for the values:

Regularized Linear Model:

- $E_{test} = 0.1109$
- $E_{cv} = 0.1204$

k-Nearest Neighbors : ( $k = 4$ )

- $E_{test} = 0.0422$
- $E_{cv} = 0.2380$


Neural Networks:

- After seeing the result from the calculations done we can observe that we got the lowest test error in case of k-Nearest Neighbors while the cross validation error came out a bit high for the same. This can be possible due to the data selection for  $E_{test}$  came in our favor and gave the best result
- Whereas in cross validation error we create different combinations of the data to get the best decision boundary so the output of  $E_{cv}$  is more reliable as it gives us a tight bound to the error rate whereas the results obtained from  $E_{test}$  does not assure that every time the error will be close to that range
- In the case of the Regularized Linear Model we got both the  $E_{test}$  and  $E_{cv}$  low and close to each other. This shows that the samples created using cross validation were close to our initial test samples and that is why the error rate is similar
- The overfitting of the data won't be a concern over here because we have regularized the data for the Linear Model and because of this it is more likely that this model will provide lower out of sample error

- If considering both the values we can say that the Regularized Linear Model gave us the best accuracy and lowest error rate for both Etest and Ecv
- In case of Neural networks with early stopping we would get a higher error rate as compared to both of the earlier models because neural networks are more prone to overfitting the data
- So their performance on out of sample data will not be as efficient as the earlier models
- We can prefer Neural Networks where we have large data or where overfitting is not a concern, but for limited data is more preferable to select the Regularized Linear model or KNN model over the Neural Networks
- We can prevent overfitting in case of Neural Networks by carefully tuning the hyperparameters of the model



## Access Link

 [HarsimranSinghDhillon\\_HW\\_4.ipynb](#)