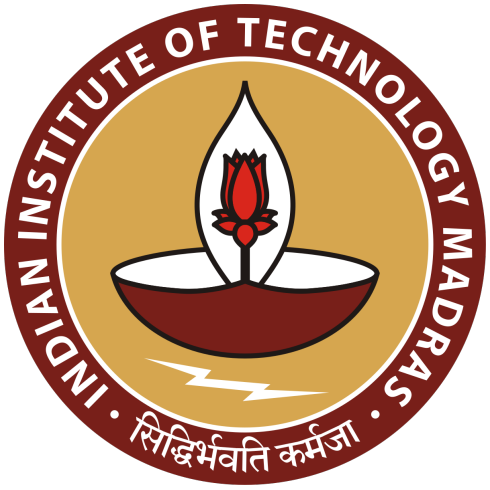
Assignment 2

CS5691: Pattern Recognition and Machine Learning



Nikhil Anand
BE20B022
Indian Institute of Technology Madras
April 2023

Contents

1 Question 1

1.1 (i)

We are firstly asked to obtain the least squares solution w_{ML} to the regression problem using the closed form solution.

The regression problem is stated as the following optimisation problem:

$$min_w||X^T w - y||^2$$

To minimise this expression, we may set the derivative to zero, and we end up with the following closed form solution:

$$w = (XX^T)^{-1}Xy$$

where X is a $d \times n$ matrix, and y is an $n \times 1$ matrix.

After loading the training data and extracting X and y from it, we then run the following command to compute the closed form solution:

```
w = np.dot(np.linalg.inv(np.dot(X, X.T)), np.dot(X, y))
```

Thus, w is a 100×1 vector which acts as the optimum value at which the linear regression has the least square error.

After finding the value of w, we compute the objective value at that w:

```
train_error = 0.03968644186272515
```

Also, if we see how w performs on the tests data set, we get the following test error:

```
test_error = 0.37072731116978985
```

Thus, it seems that the w that best fits the train data doesn't fit the test data as well. Thus, it kind of overfits the data. This also simply means that the specific linear pattern followed by the train data is not replicated by the test data.

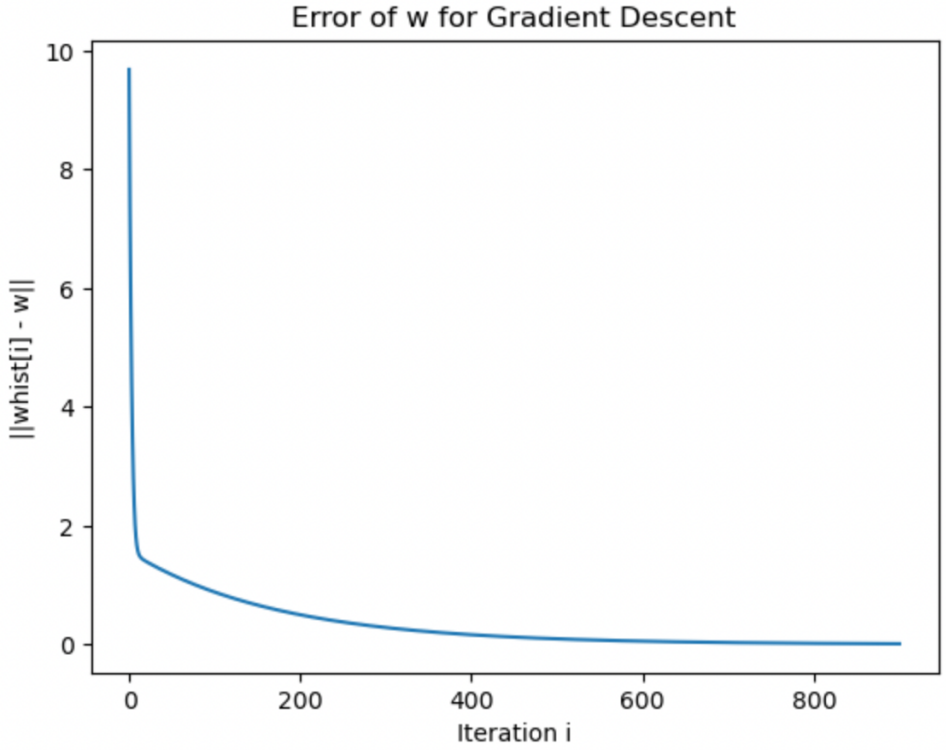
The least square errors are computed as:

```
test_error = ((np.dot(Xtest.T, w) - ytest)**2).mean(axis=0)
train_error = ((np.dot(X.T, w) - y)**2).mean(axis=0)
```

1.2 (ii)

We are asked here to write the code for the gradient descent (steepest descent) algorithm for computing w. This can be done for computational considerations, since calculating the inverse of XX^T can be computationally expensive.

We create an array *whist* to store the history of updates of w, and in every iteration, we add to the list an updated w based on the gradient descent. Here, we use a constant step size of 0.000007 during the gradient update as follows:




```
whist.append(whist[-1] - 0.000007*(np.dot(X, X.T)@whist[-1] - np.dot(X, y)))
```

This is simply iterated 900 times in the hope of converging to our true w value.

If we plot the true w value (as per closed form), we can get the error between the gradient estimate and the true minimum at each iteration i. After many iterations, this number converges to 0, showing that the gradient descent eventually converges to the true minimum.

Error is given by:

$$Error = ||w - w_{ML}||$$

Now we can check the test error based on the formula mentioned in part (i) of Question 1.

```
testerr = ((np.dot(Xtest.T, w) - ytest)**2).mean(axis=0)
```

As expected, the test error is around the same as in part (i).

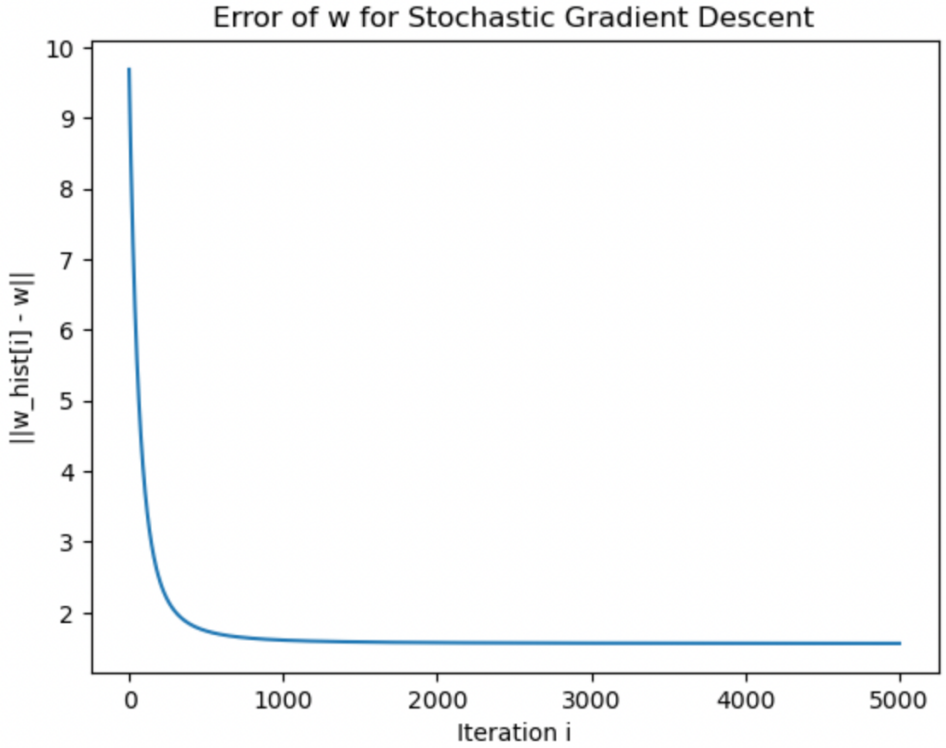
1.3 (iii)

Now we sample one hundred points at a time and perform a gradient update based on those points in our stochastic gradient descent algorithm. The gradient update is shown below. Again, we use an optimum step size that allows the algorithm to converge rather than diverge. We find such a step size by trial-and-error.

```
wc = wc - 0.00001*(np.dot(Xnew, Xnew.T)@wc - np.dot(Xnew, y))
```

The predicted w is taken to be the mean of all wc values computed till now. This predicted w is added to whist.

Then, the error difference of whist and w is plotted against iterations. We get the following plot:



This time we observe that the error in w does not converge to close to zero. It is hard for it to converge to zero, primarily because every time a new sample is drawn and the step direction of the gradient is calculated according to a different objective every time.

error = 1.5628686008579222

When calculating the error in the test dataset, we get -

error = 0.4298589508077529

The error is worse than the previous methods, but it turns out to be better than the training error. This may be because it provides additional stochasticity and may reduce some aspect of overfitting to some degree.

2 Question 2

2.1 (i)

Our first task is to code up the gradient descent algorithm for ridge regression. First, we state the objective that must be minimised for ridge regression:

$$min_w ||X^T w - y||^2 + \lambda ||w||^2$$

Now, when we compute the minimum of this, the closed form expression comes out to be:

$$w = (XX^T + \lambda I)^{-1} Xy$$

For gradient descent, the update rule that w would follow would be:

$$w^{t+1} = w^t - \eta \nabla_w E$$

Here, $\nabla_w E$ is given as:

$$\nabla_w E = (XX^T)w - Xy + \lambda w$$

Now to code the gradient descent algorithm for ridge regression, we use this update rule.

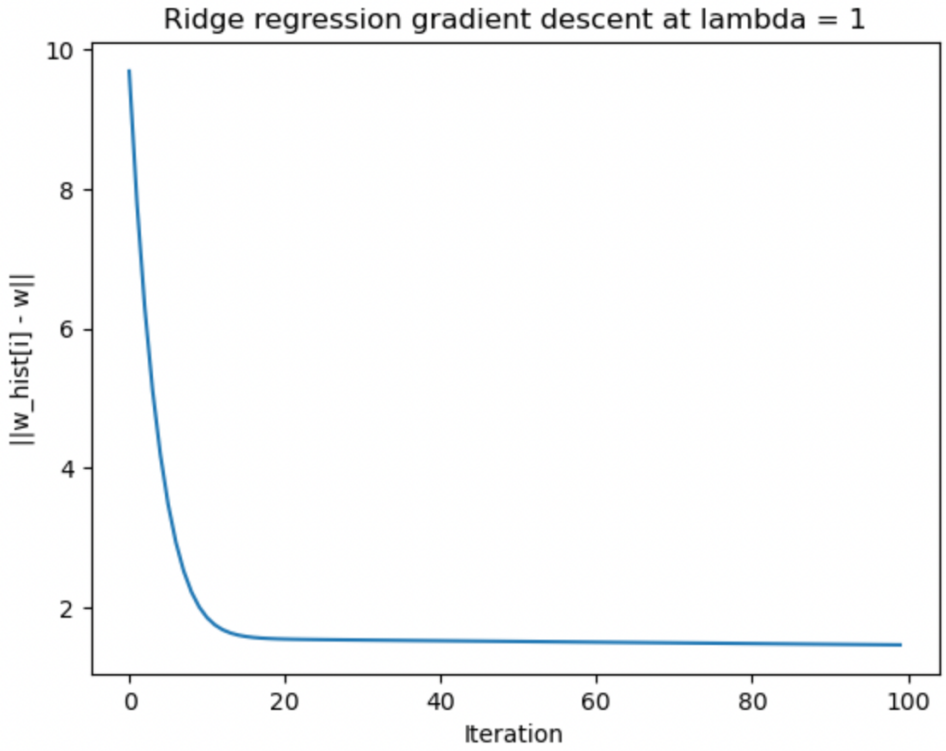
```
for i in range(100): # take lambda as 0.1
    whist.append(w)
    grad = np.dot(X, np.dot(X.T, w)) - np.dot(X,y) + 0.1*w
    w = w - 0.000001*grad
```

This gradient descent populates whist with the values of w during the gradient updates.

We then compute the true w from the closed form expression:

```
wtrue = np.dot(np.linalg.inv(np.dot(X,X.T)+0.1*np.eye(X.shape[0])),np.dot(X,y)
```

We then plot the history values of computed w's vs the true w, to see how it varies and tends to zero over time.



The final error in w is about 1.4437237240194059.

The test and train errors come out to be 0.37068490482025923 and 0.03968644484384218 respectively.

2.2 (ii)

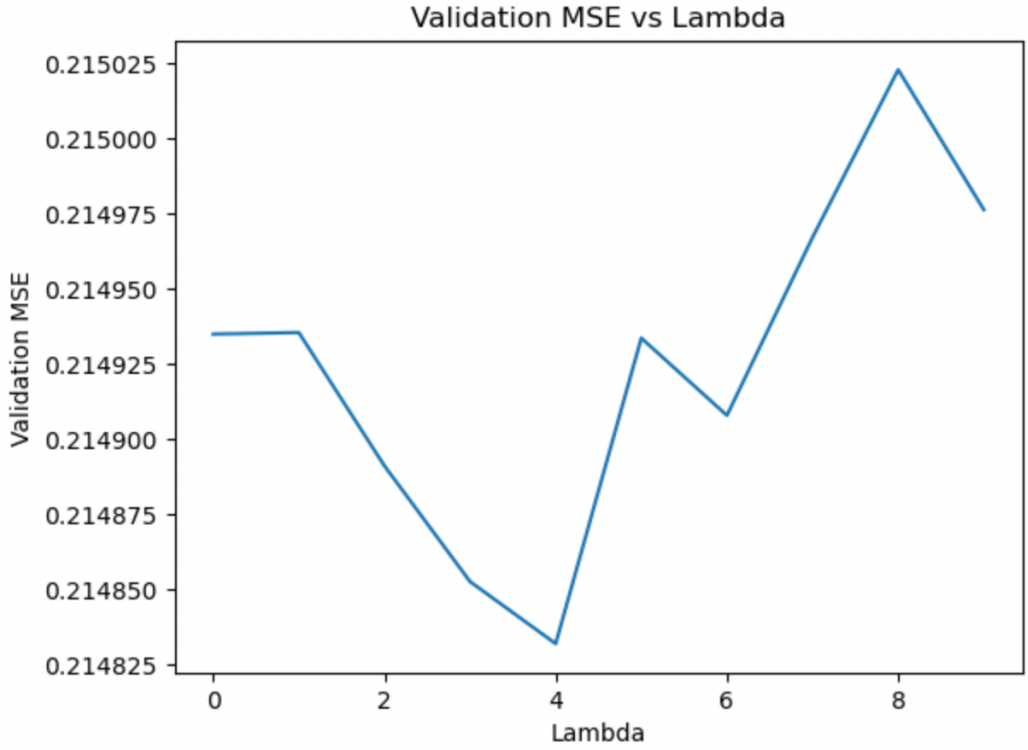
Now, we must cross-validate for various choices of λ and plot the error in the validation set as a function of λ.

We first write the splitK(k) function to perform a K-fold split on the given train dataset.

Then we vary the value of l from 0 to 9 (where λ is l*0.01). Thus, we are varying λ from 0 to 0.09.

For each λ we split the dataset into 8 possible training-validation splits, and for each split, we compute the ridge regression w on the training set, and calculate the validation error on the validation set. Finally, the mean of the entries in mse_splits is added to the array of MSEs.

We now plot the validation errors vs the corresponding values of lambda. The plot seems to vary from time to time, due to the stochastic nature of selection of the training and validation splits. However, we come to notice that the value of λ at which the plot is minimum is around 0.04 most of the time.



As we observe, the test error of the ridge regression estimate w_R and the maximum likelihood estimate w_{ML} come out to be:

wR = 0.3707103466753794
wML = 0.37072731116978985

It appears that the two are very similar. wR is slightly better. This could be due to the regularisation term of ridge regression which may have helped avoid some form of overfitting in the ML estimator.