Lab 7,8 and 9 Report

Introduction

Supervised algorithms are an effective way to classify and decode data to make predictions on new information. With these algorithms neural signals can be taken to anticipate how hands move, if someone is having a seizure, or what letter someone wants to type. Classifiers deliver fast bit rates and categorize data into several categories. Continuous decoders normally use a variant of linear regression to predict a wider range of values. In this lab we implement Naive Bayes and linear discriminant analysis algorithms to forecast what direction a monkey would move its hand in based on training data and neural firing rates. Additionally, linear regression, ridge regression, LASSO, and Kalman filters were trained and used to predict the x and y position, and x and y velocity using training data.

Methods

The first step in this lab consisted of implementing the Naive Bayes algorithm. The "firingrate.mat" data given in class is used which consists of firing rates from 95 neurons in 182 trials for 8 different reaching positions for a monkey. Naive Bayes is a generative classifier algorithm. This means that there is a probabilistic model of the distributions that the data is being drawn from and that it is only able to organize data into a set number of categories. Naive Bayes algorithm works by utilizing Bayes Rules, Equation 1, which gives the probability of an event based on past information. Equation 2 describes how the Naive Bayes classifier works, which looks for the maximum likelihood target given the spike data. In this classifier, a poisson distribution was assumed which presumes a distribution of integer counts oriented in bell curves with a greater variance for higher means. This is a fair assumption as neurons usually act this way. Naive Bayes also assumes that the different features of sample x are independent from one another for a given class. This is usually not true but has been shown to not have a great effect on decoder performance. To implement the algorithm the firingrate.mat data was imported into MATLAB and split into training and testing data. The training data consisted of the first 91 trials including all of the neurons and possible directions while the testing data consisted of the last 91 trials including all of the neurons and possible directions. The classifier was then trained on the training data. The mean for each feature-class (neuron-direction) pair of the training data was recorded. A doubly nested for loop was then created with the outer iterating through the 91 trails and the inner iterating through the 8 targets. The feature was saved for all of the neurons in each iteration for the trail and target of the testing data. For each feature the probability density of observing that feature in that direction was found. This was done by inserting another for loop going through the 8 target guesses and a final fourth embedded loop that iterates through the 95 neurons. The probability was found using Equation 3. Normally the probabilities are multiplied like in Equation 2, however since they can be very small numbers, the log of the probabilities was taken and summed to avoid floating point errors. MATLAB's poisspdf() function was also utilized to find the poisson density given the feature of the neuron and mean for each neuron at its target guess. All of the probabilities were recorded for each guess and the guess that gave the

maximum probability was chosen as that prediction. The accuracy of the predictions was evaluated by going through each prediction and recording how many were correct (target 1 equaled 1, target 2 equaled 2 and so on). The accuracy was determined as the number of correct predictions divided by the total number of predictions. To ensure that the algorithm was working correctly, a spoofed data set was created and tested with. This was done by taking the mean of firingrate.mat across the number of trials and across the 8 predictions leaving a 95 value vector. An empty matrix of 95x128x8 values was created. The poissrnd() function was used to generate random numbers from the poisson distribution and neuron mean for all the trials and targets of each neuron.

The next step looked at implementing a linear discriminant analysis algorithm. Linear discriminant analysis (LDA) conceptually makes a hyperplane separating two clusters of data. LDA is also a classifying algorithm meaning it can only estimate into a discrete number of classes. A w matrix is chosen that when multiplied by the data gives the best single number for differentiating between two classes. Equation 4 is what is used to determine what class the data belongs to. Equation 5 determines the location of the centroid of clusters. If the data is independent the centroids can be subtracted to find the optimal w. If the clusters are not independent, Equations 6, 7 , and 8 must be used to find the maximum ratio of between class covariance to within class covariance and it results that the optimal w comes from the largest S_w^{-1} $(m_1 - m_2)$ where S_w is within class covariance and m_1 and m_2 are the centroids. Linear discriminants put all the data together and get one covariance matrix. This assumes that the data is legitimately correlated and makes the decision boundary a line. Quadratic discriminant will calculate a separate covariance matrix for each class. This will usually do a worse job estimating covariance since there is less data to understand it, potentially may cause more overfitting, and might make the decision boundary quadratic. LDA assumes that the data is normally distributed meaning each feature would show a bell shaped curve when plotted.LDA also assumes linearity, independence, and homoscedasticity(variance among groups variable are the same across predictors), LDA was implemented in MATLAB using most of its functions. A linear discriminant was also used in this lab meaning each of the classes is assumed to have identical covariance matrices. The data from “ecogclassifydata.mat” given in class was loaded into MATLAB which contained average power values from 0.5 sec prior to movement to 1.5 sec after movement in 60 -120 Hx bands for 38 trails with data recorded from 27 electrode pairs. The group variable indicates which finger was squeezing (1 is rest, 2 is thumb, 3 is index, 4 is middle, 5 is ring and pinkie). MATALB’s classify function was used and given the test data, training data, training classes and ‘linear’ parameter to ensure a linear discriminant was used. The test data and training data were changed so that each trial was able to be the test data while the rest was the training data. If the classify function predicted the correct data this was counted as a correct prediction. The correct predictions divided by the total number of predictions was done and determined to be the accuracy. To determine where the errors occurred a Figure 1 was made to show what percentage of the time a finger is classified as itself and what percentage of time it was classified as another finger.

A linear regression algorithm was then implemented. The basic equation for linear regression is given by equation 9. Linear regression is a linear model that assumes a linear relationship between the input and output variables. Linear regression also assumes that the variance of the errors is constant for all independent variables, there is little or no multicollinearity, and the observations are independent. In this lab testing data is used to find a decoder, B, that can be multiplied by the input data Y, to find the predicted output variable X. The objective function for linear regression is given by Equation 10. The data used was from the "contdata95.mat" file given in class and has an X variable where there are columns of X position, Y position, X velocity, and Y velocity at 31413 time points. There is also a Y variable that has firing rates for 95 recorded units. The data is currently in 100 ms bins but a delay was added to the data to result in 950 total features. To do this, each column was duplicated to the right 9 times and in the first duplicated column the values were rotated down with the top value being 0. In the second duplicated column the values were rotated down 2 positions and the top two values were 0, so on and so forth. Then a column of 1s was added to the front for computations. Inserting this time lag is only helpful for "offline" testing (not testing done on a person live) because it will cause a person's previous firing rate to affect the action now and may not be as helpful. The data was then split into training and testing data where the training data contained the first 15707 time points and the testing data contained the time points form 15708 until the end. Equation 11 was used with the training data to find the decoder matrix, B. Equation 9 was then used with the testing data and B matrix to find the predicted values. Figures 2 and 3, show the plots of the predicted X positions, Y positions, X velocities, and Y velocities respectively along with the real values. The mean squared error for each position was then calculated with MATLAB's immse() function with the prediction and real values as parameters. Additionally, MATLAB's corr2() function was used to find the mean square error of all the predicted values.

Linear regression was extended to use ridge regression. Ridge regression is very similar to linear regression but it introduces a regularization term that should counter overfitting. Ridge regression has similar assumptions as linear regression: linearity, constant variances, and independence. Ridge regression does work better than linear regression with mutlicolinearity The objective function for linear regression is described by Equation 13 now and Equation 14 describes how to find the B term with ridge regression. The same data with the time lag that was used in the linear regression was used for this part and the first half of data was set as the training data and the second half as the testing data as well. 15 lambda values equally spaced from 0 to 0.1 were evaluated to see the effects on correlation and mean squared error. Each lambda value was looped through and the B matrix was calculated using Equation 14 with an N of 15706, training data, and an identity matrix with sides of 951. The mean squared error for each position was then calculated with MATLAB's immse() function with the prediction and real values as parameters. Additionally, MATLAB's corr2() function was used to find the mean square error of all the predicted values. Plots of the predicted data with the real data of 2 lambda values were

plotted in Figures 5 and 6 and the lambda value that gave the most accurate results was determined.

A modified version of ridge regression was used called least absolute shrinkage and selection operator (LASSO). LASSO does variable selection and regularization in an attempt to make more accurate predictions. It uses the L_1 norm instead of the L_2 norm. Equation 14 describes the objective function for LASSO. The assumptions for LASSO are the same as ridge regression: linearity, constant variances, and independence. The same lagged time data used for linear and ridge regression was used for this part. The first 15707 data points were set as the training data, the second half were used as testing data, and the best lambda value found in the ridge regression was used. MATLAB's lasso() function was used with the training data and the best-found lambda value to get the B matrix for each prediction (X position, Y position, X velocity, Y velocity). The mean squared error (MSE) was found using the immse() function and the correlation was found using the corr2() function.

A Kalman filter was used to incorporate neural data, physics data, and some noise to predict motion. The model assumes that the current state is a linear function of the previous state and the noise is normally distributed. The same data with the time lag that was used in the linear regression was used for this part and the first half of data was set as the training data and the second half as the testing data as well. Xt was created with transposing all but the last element in the first half of the positions and velocities of the training data and x_{t-1} was made by taking everything but the first term for the positions and velocities of the training data. Then the A matrix, the relationship between the previous x and next, was calculated with equation 15. C, the relationship between the neural data and x data, was calculated with equation 16 with the training data. W, the noise term with physics and Q the noise term for the neural data were created in equations 17 and 18. A loop through all time points of the testing data was done. In each iteration Equation 19 was used to update the posterior state estimate of x at time t, Equation 20 to update the posterior error covariance of x, Equation 21 to update the Kalman gain (best proportion of the neural guess to include compared to the best physics guess), Equation 22 to update x, and Equation 23 to update the error covariance. The MSE and correlation coefficients were calculated for each paramter and they were plotted in a graph along with the real variable.

Results

In the first part of this lab a Naive Bayes algorithm was used to classify what direction a monkey would put his hand using training data and neural firing rates. This resulted in a 97.39% accuracy when the model was used on the training data. This shows that the model was working well classifying almost all of the testing data correctly. To ensure that the model is working properly and does not have a deceptively high percentage, it was also tested on a spoof data set that randomly generated numbers for each neuron based on a poisson distribution using the mean of all the trails and directions for each neuron.This resulted in a 12.43% accuracy rate which makes sense as there are 8 possible classifications random data should result in a ⅛ accuracy. Naive Bayes is a fast program that is easy to implement and as shown can predict data really well

sometimes. However, one disadvantage is that it assumes that the features are independent which works in this case but may not always work. Additionally, if it sees a categorical variable that was not shown in the dataset it assigns it a zero probability. Lastly, since it is a classifier, it can only categorize the data into a set number of groups and won't work as well for continuous data as other methods.

LDA was also used to determine which finger was squeezing given power values from electrode pairs. The accuracy of the predictions was found to be 71.79%. This is less accurate than the Naive Bayes algorithm but it also had less data to train with. A confusion matrix was also created in Figure 1 that showed the percentage of time a finger is classified as itself and what percentage it was classified as another finger. Figure 1 shows that group 3 had the lowest percentage of classifying itself only doing so about 40% of the time. Group 1 had the best percentage of classifying itself doing so 100% of the time. Groups 2,4, and 5 classify themselves about 85% of the time. LDA is a quick and simple classifier although it has some limitations. LDA assumes a normal distribution, which may not always be the case. If the other assumptions mentioned above are not met it can decrease the accuracy of the predictor. LDA does not work great with small sample sizes. LDA is also a classifying algorithm so it has a set number of groups to categorize the data.

Linear regression was implemented to predict the position and velocity of a hand based on training data and firing rates. Figure 2 shows the predicted X position and real X position of the hand over 250 ms and Figure 3 shows the same for Y position. In Figure 2 the line for predicted X value is pretty close to the real X values. The predicted X values is much more noisy especially in between movements it does not stay flat like the real values but whenever there is a spike the predicted values spike as well and to a similar magnitude. There are similar observations for Figure 3. The predicted Y values follow the real Y values pretty closely. There is also a lot more noise as it goes up and down when the arm is not reaching but the predicted values also line up with the spikes and to a similar magnitude. mm/s? The MSE and the correlation coefficient of all the parameters are reported in Table 1. The X and Y positions MSEs are 129.3625 and 135.9947 while the X and Y velocities are 3002.4 and 2935.6 respectively. The X and Y position correlations are .9141 and .8859 while the X and Y velocities are .8696 and .8387 respectively. MSE tells the average squared difference between the predicted points and the real points. The MSE of the positions are about the same and the velocities are about the same but the positions are much lower. This means that the predictions for position were on average closer to the real values than the predictions for velocities. Correlation says how related two variables are. The correlations for position are a little higher than the correlations for velocity meaning that the positions are more closely related to the actual values than the velocities. Even though the positions had better predictions than the velocity, this does not mean the velocities had bad predictions. The velocities were still able to come close to the real values. Linear regression is a great tool that is easy to implement to forecast continuous data and the output measurements are easy to interpret. However, linear regression assumes a linear

relationship between dependent and independent variables and that there is independence. Outliers can also have large effects on the results and linear regression is prone to underfitting.

Ridge regression, an extension of linear regression, was implemented on the same data as before for the same purpose. Neuron firing rates were used to predict X and Y position and velocity. Different regularization terms were tested to see what would give the best prediction. Figure 4 shows the MSE of different lambda values and Figure 5 shows the correlation of different lambdas for the X posittion. Although the differences in MSE and correlation are small, we see that a lambda value of about .0286 gives the lowest MSE and highest correlation which means that it made the most accurate predictions. Using a lambda of 0.0286 the MSE of X position, Y position, X velocity, and Y velocity was found to be 129.7188, 136.2481, 3033.1734, 2945.3164 respectively while the correlations were found to be .9139, 0.88562, .86811, 0.83812 (these values are also found in table 2). The regularization term reduces overfitting by penalizing the B matrix for having large numbers. Ridge regression and linear regression are very similar algorithms. They use almost identical equations but ridge regression introduces the regularization term to get better predictions. We see here that ridge regression gives slightly lower MSEs which means there is less error, and slightly higher correlations, meaning that the predictions are more correlated to the actual values, but not by a significant amount.

Another extension of linear regression, LASSO, was used to calculate the same values on the same data as the previous two regression methods. Neuron firing rates were used to predict X and Y positions and velocities. LASSO incorporates regularization and variable selection to improve the accuracy of the linear regression. Lasso was done with the same best regularization term as the ridge regression, 0.0286, for each of the parameters. The MSE of X position, Y position, X velocity, and Y velocity was found to be 138.1741, 210.1580, 3183.1, and 3056.9 respectively while the correlations were found to be .9153, .8870, .8703, and .8395 (these values are also found in table 3). The MSEs are higher than in the linear and ridge regression however the correlation is also higher. This shows that LASSO might be better or worse depending on the use but it does provide a closer correlation to the actual data than in linear or ridge regression. The terms in the B matrix in LASSO often look smaller than the terms in the B matrix in ridge regression. This is likely from the regularization that is done in LASSO and may help to have the outliers not have a significant impact on the predictions.

Finally, Kalman filter was also implemented

Appendix

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	129.3543	135.9860	3002.2	2935.4
Correlation	.9141	.8859	.8696	.8387

Table 1. Mean squared error and correlation coefficient of linear regression parameters.

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	128.8856	135.7369	2992.8	2927.7
Correlation	.9144	.8860	.8699	.8391

Table 2. Mean squared error and correlation coefficient of ridge regression parameters with a best found lambda of .0284.

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	138.1741	210.1580	3183.1	3056.9
Correlation	.9153	.8870	.8703	.8395

Table 2. Mean squared error and correlation coefficient of LASSO.

Figures

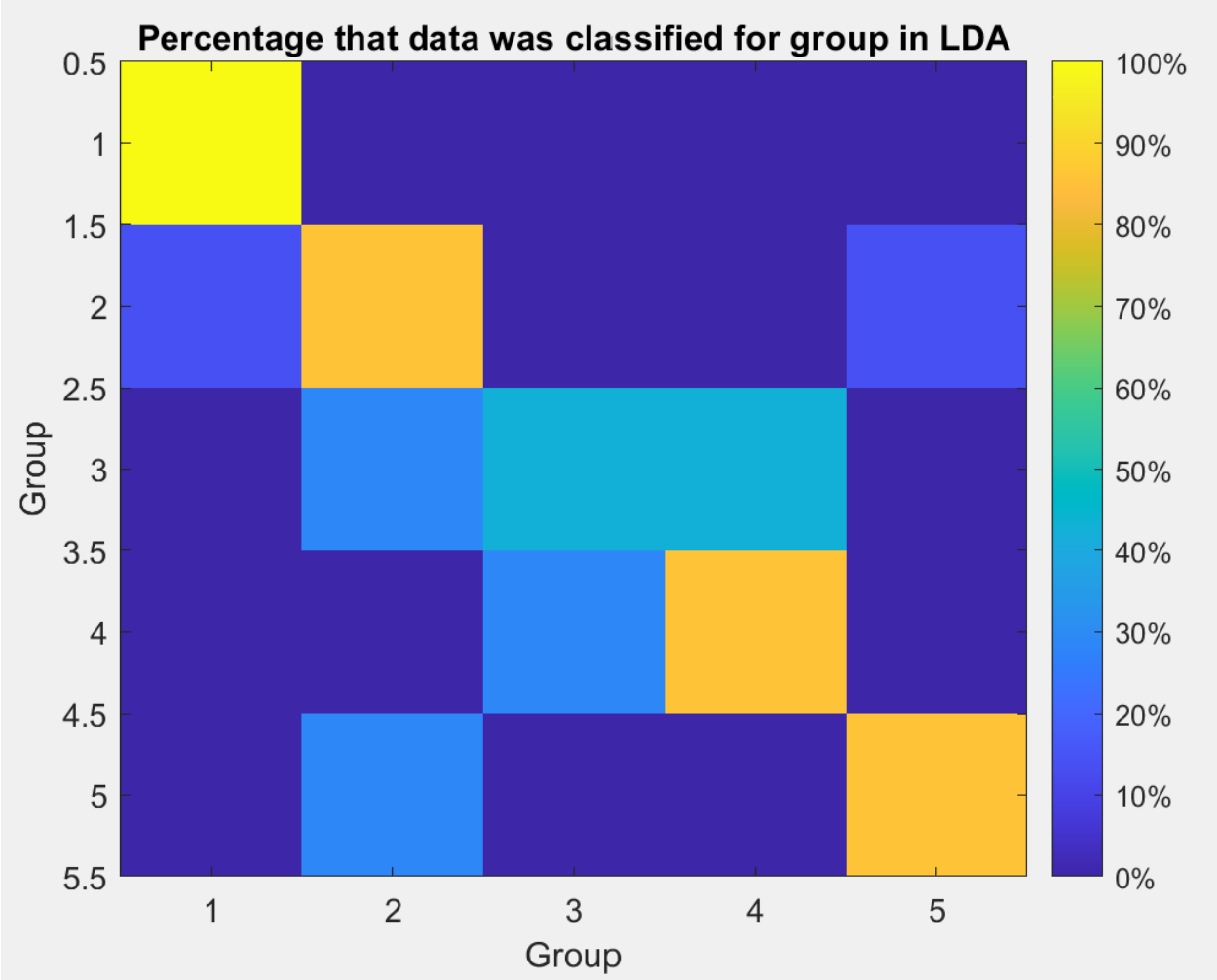

Figure 1. Percentage that data was classified for groups in LDA.

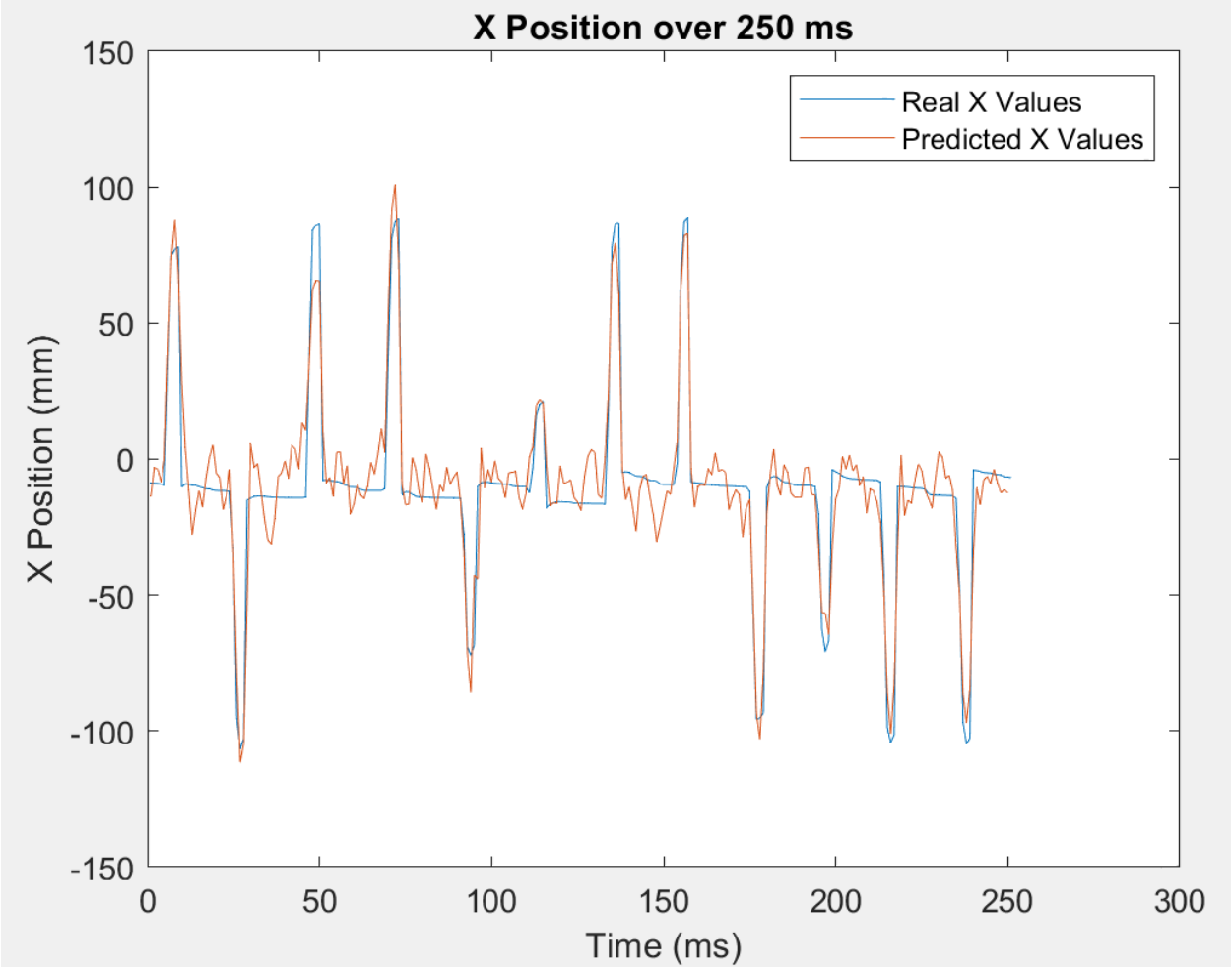

Figure 2. Predicted X position and real X position over 250 ms using linear regression.

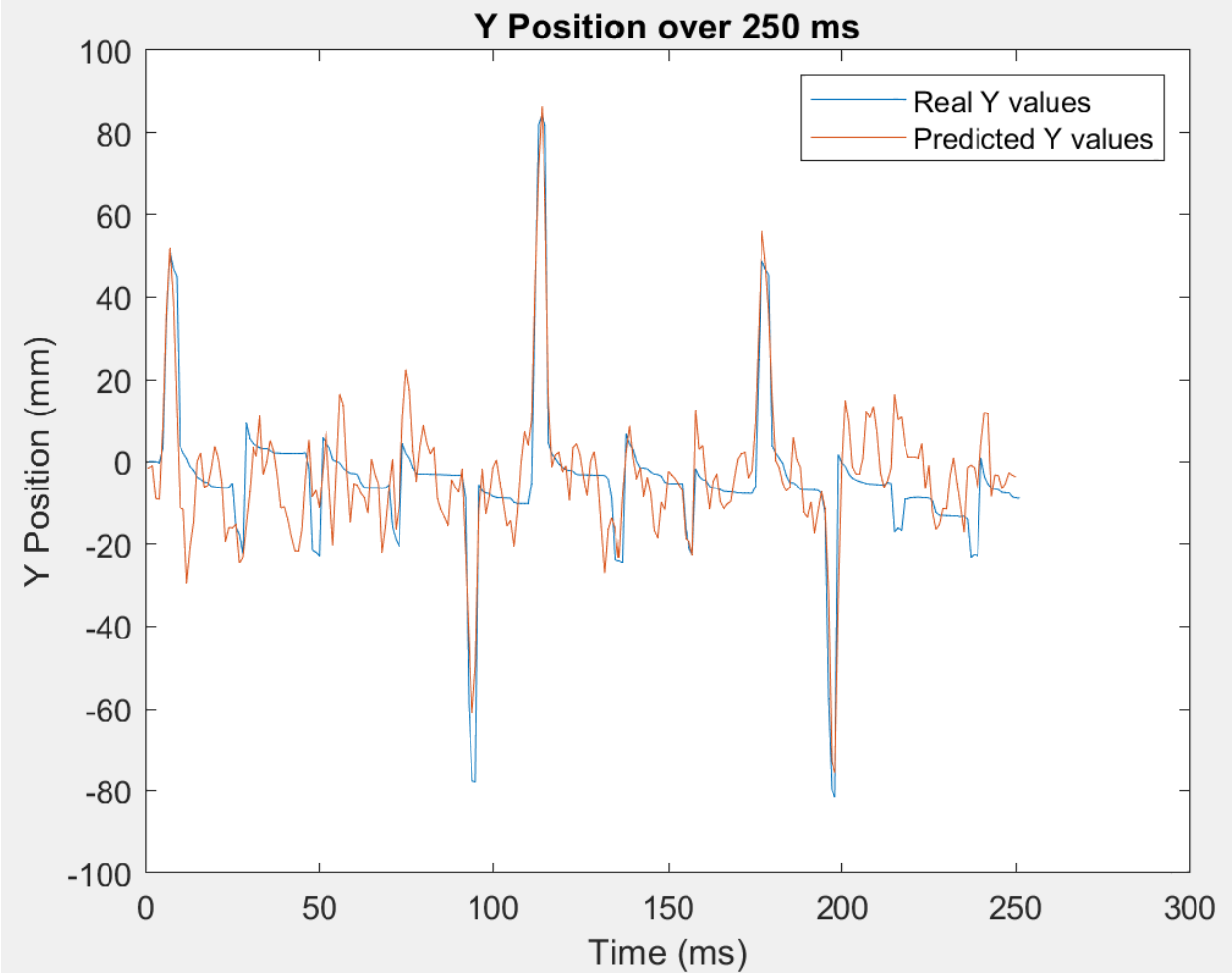

Figure 3. Predicted Y position and real Y position over 250 ms using linear regression.

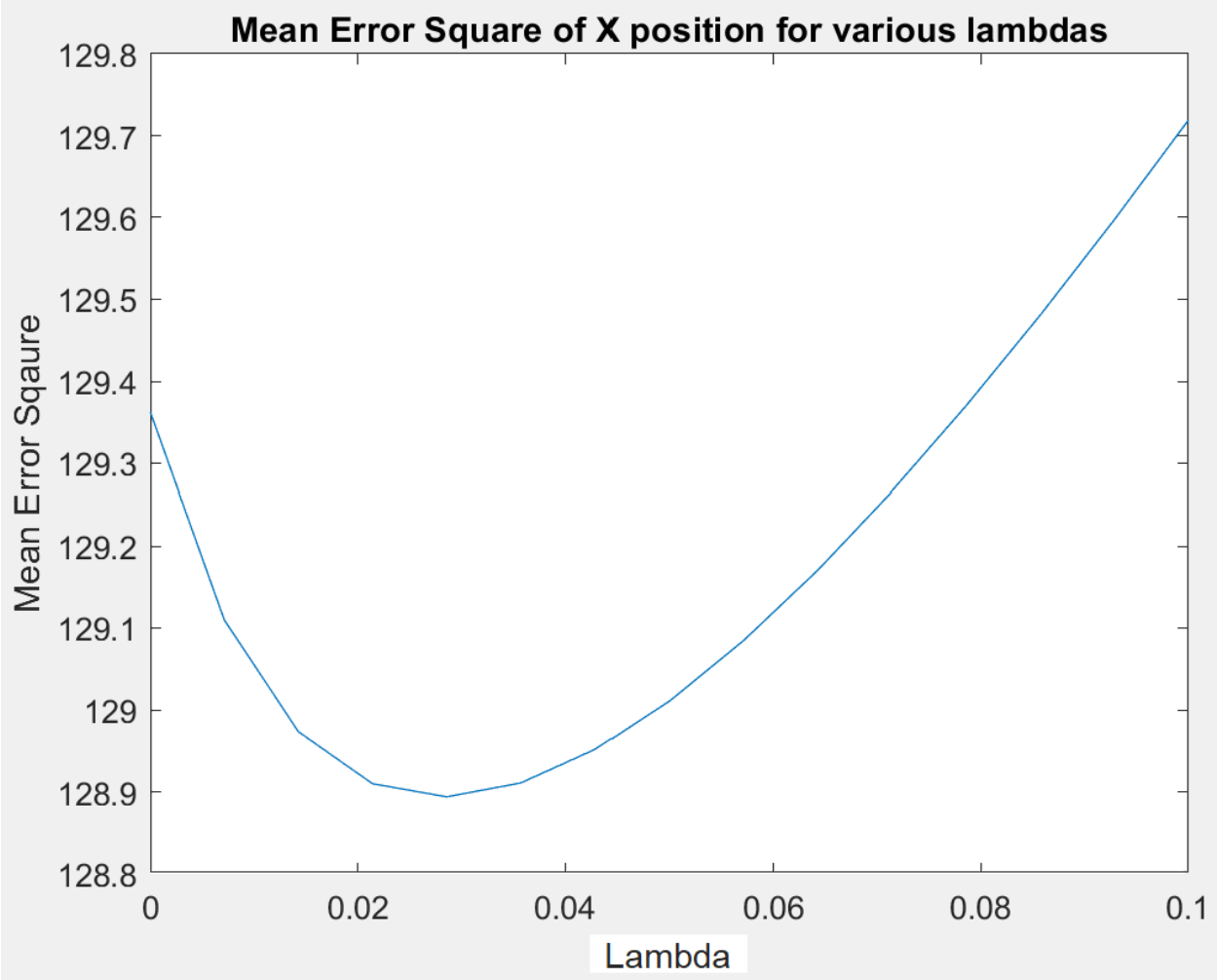

Figure 4. Sweep of mean square error for different lambdas in ridge regression.

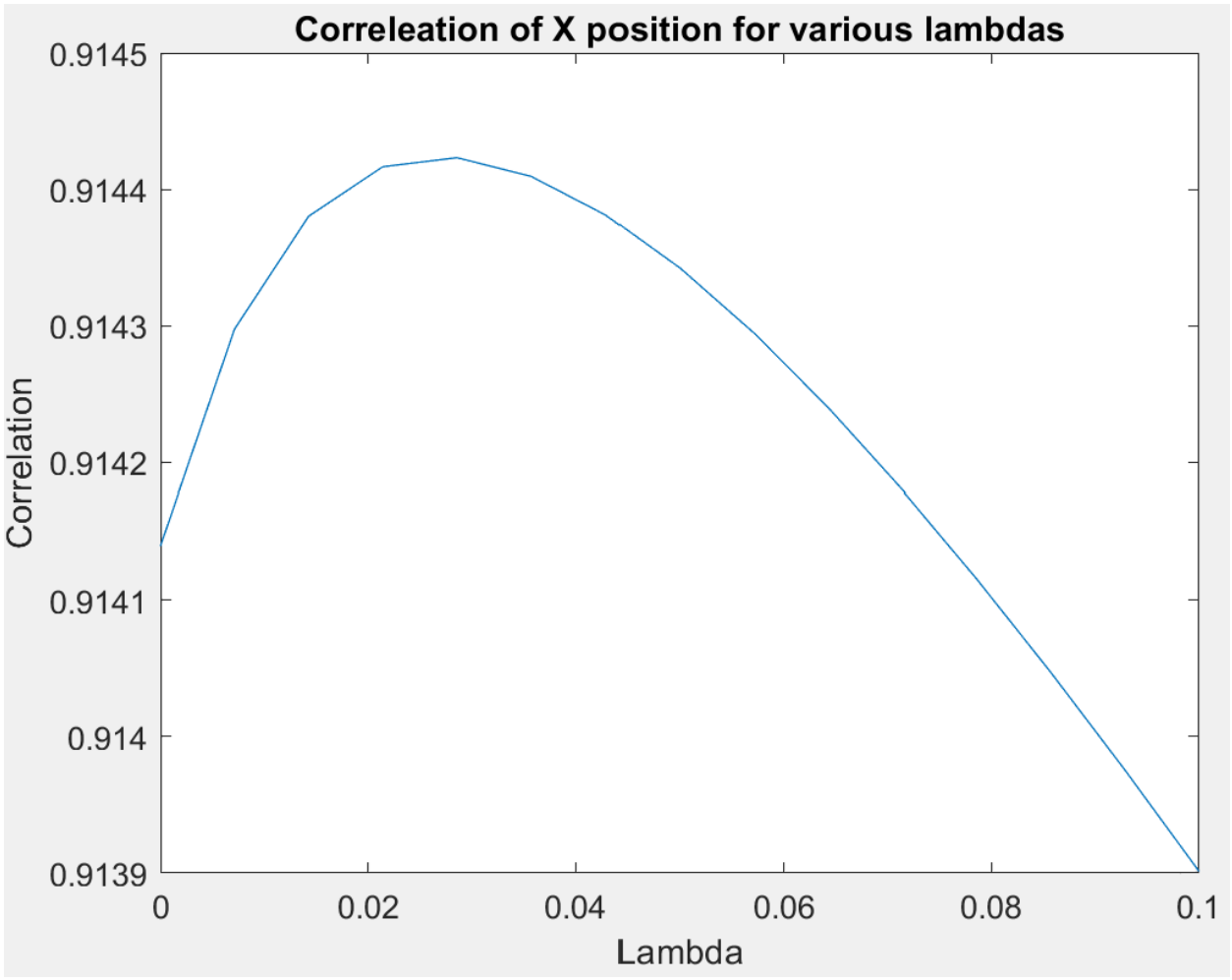

Figure 5. Sweep of correlations for different lambda values in ridge regression.

Equations

Equation 1

$$p(\vec{x} \mid \vec{y}) = \frac{p(\vec{y} \mid \vec{x})p(\vec{x})}{p(\vec{y})}$$

Equation 1. Bayes Rule used to calculate probability given prior information.

$$\hat{f}(x) = \arg\max_k \left[\prod_{j=1}^{D} \hat{g}_k^{(j)}(x) \right]$$

Equation 2. X is a vector of features, D is the number of features in x, and $\hat{g}_k^{(j)}$ is the estimated probability distribution of features j for class k.

$$\sum_{j=1}^{D} \log \hat{g}_{k=1}^{(j)}(X_i)$$

Equation 3. Total probability using logs because the numbers can be small.

$$y(\vec{x}) = \vec{w}^T \vec{x} + w_0$$

Equation. 4. Calculates discriminate in LDA. y(x) says what x belongs to, x is the data to be classified, w^T is what is fit during training, and w0 is the bias term to determine if y(x) is > or < 0.

$$\vec{m}_k = \frac{1}{N_k} \sum_{n \in k} x_n$$

Equation 5. Centroid of cluster of data.

$$J(\vec{w}) = \frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$$

Equation 6. Ration of between class covariance and within class covariance.

$$S_B = (\vec{m}_1 - \vec{m}_2)(\vec{m}_1 - \vec{m}_2)^T$$

Equation 7. Between class covariance.

$$S_W = \sum_{n \in class1} (\vec{x}_n - \vec{m}_1)(\vec{x}_n - \vec{m}_1)^T + \sum_{n \in class2} (\vec{x}_n - \vec{m}_2)(\vec{x}_n - \vec{m}_2)^T$$

Equation 8. Within class covariance.

$$\vec{x} = \vec{y}B$$

Equation 9. Linear regression equation.

$$\min_{\vec{w},b} \left[\frac{1}{n} \sum_{i=1}^{n} (\vec{w}^T \vec{x}_i + b - y_i)^2 \right]$$

Equation 10. Objective function for linear regression. x_i is a vector of features for observation i, y_i is the measured outcome of observation i, and w_i and b are estimated parameters of the linear model.

$$B = \left(\vec{y}^T \vec{y} \right)^{-1} \vec{y}^T \vec{x}$$

Equation 11. Equation to find B matrix with testing data for linear regression.

$$\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i + b - y_i)^2 + \lambda\left|\left|\vec{w}\right|\right|_2^2\right]$$

Equation 12. Objective function for linear regression.where $||w||_2$ is the L_2 norm of w.

$$\beta = (X^T X + N\lambda I)X^T Y$$

Equation 13. Equation to find B matrix with testing data for ridge regression. Lambda is the regularization term, N is the total number of training data points, and I is the identity matrix.

$$\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i + b - y_i)^2 + \lambda\left|\left|\vec{w}\right|\right|_1^2\right]$$

Equation 14. Objective function for LASSO.

$$A = X_t X_{t-1}^T (X_{t-1} X_{t-1}^T)^{-1}$$

Equation 15. Kalman filter A equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$C = Y X_t^T (X_t X_t^T)^{-1}$$

Equation 16. Kalman filter C equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$W = \frac{1}{\#\,pts - 1}(X_t - AX_{t-1})(X_t - AX_{t-1})^T$$

Equation 17. Kalman filter W equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$Q = \frac{1}{\#\,pts}(Y_t - CX_t)(Y_t - CX_t)^T$$

Equation 18. Kalman filter Q equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1}$$

Equation 19. Kalman filter previous posterior state estimate of x at time t update (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$P_{t|t-1} = AP_tA^T + W$$

Equation 20. Kalman filter previous posteriori error covariance of x (Kalman, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$K_t = P_{t|t-1}C^T(CP_{t|t-1}C^T + Q)^{-1}$$

Equation 21. Kalman gain equation (Kalman, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$\hat{x}_t = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t-1})$$

Equation 22. Kalman filter x update equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$P_t = (I - K_t C)P_{t|t-1}$$

Equation 23. Kalman filter error covariance equation update. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)