THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Group Project

ST443 Machine Learning and Data Mining

Group 9

By
Magdalena Dunicz 202236527
Tweshaa Dewan 202210055
Maciej Kuczek 201919998
Reuben Mathew 202322381
Byungho Jeong 202341465

Question 1: Machine Learning on Real Data

1.1 Introduction

This report builds upon the "Machine Learning-based Occupancy Estimation Using Multivariate Sensor Nodes" paper by Singh et al. (2018)[2]. Using data from sensors measuring CO2, temperature, light, motion, and sound, we employ a selection of classification algorithms to estimate the number of occupants in a room. We share the belief of the authors that robust Machine Learning Algorithms can enhance the efficiency of heating and air-conditioning systems and help cut costs.

1.2 Data

The data set was obtained from **UC Irvine Machine Learning Repository** and is the same as the one used in the aforementioned paper. This data set contains data collected from non-intrusive sensors measuring CO_2, temperature, illumination/light, sound, and motion. The data was collected for a period of 4 days in a controlled manner with the occupancy in the room varying between 0 and 3 people. The ground truth of the occupancy count in the room was noted manually. The final data set has 10129 observations and 16 features.

1.2.1 Feature Description

The variables included in the data set are:

- date on which the data was recorded `Date`
- time at which the data was recorded `Time`
- four different sensors in the room which recorded the temperature `S1_Temp`, `S2_Temp`, `S3_Temp`, `S4_Temp`
- the illumination in the room `(S1_Light, S2_Light, S3_Light, S4_Light)`
- the sound in the room `(S1_Sound, S2_Sound, S3_Sound, S4_Sound)`
- sensor which recorded the CO_2 in the room (`S5CO_2`)
- slope of the line which was calculated by fitting a linear regression in the values of CO_2 `(S5_CO_2_Slope)`
- two sensors deployed on ceiling ledges for motion detection using passive infrared `(S6_PIR, S7_PIR)`
- the occupancy count of the room `(Room_Occupancy_Count)`

It is worth noting that for over 80% of the observations, the room is empty `(Room_Occupancy_Count = 0)`, which means that the data set is heavily imbalanced. As this can cause accuracy-motivated model selection to be biased towards the majority class, we have implemented also ROC AUC score and F1 score, which are more robust to this issue. We checked the completeness of data and found no observations with missing values. Figure 17 on the Appendix shows the summary statistics of the features.

1.2.2 Pre-Processing

The data was split 80/20 into training/validation sets (with stratification). Given that some machine learning algorithms may be sensitive to the magnitude of variables, we used `sklearn`'s `StandardScaler` to generate a separate array of features with mean zero and standard deviation of one. The standard score of a given feature is calculated as:

$$Z = (x - u)/s$$

where u is the sample average and s is the sample standard deviation. Crucially, we fit the scaler using training data only, to prevent any data leakage.

1.2.3 Visualisation via PCA

In order to visualise the data, Principal Component Analysis with n_components=3 was applied to the scaled data. The three-dimensional scatterplot is shown as figure 20 in Appendix.
The three components explain 72.4% of the variance. Upon examining the three-dimensional scatter plot, it is evident that points representing an empty room are predominantly grouped together, with not much overlap observed with points representing a room with 1-3 occupants. However, the points representing different levels of occupancy are not as easily separable. This suggests that classifying the exact number of occupants is a more difficult problem than predicting if the room is occupied at all. To address this challenge, we employ nine different classification algorithms.

1.3 Review of Approaches Tried

1.3.1 Multinomal Logistic Regression

Logistic Regression models the probabilities of possible outcomes using a logistic function (note that `scikit`'s implentation applies L2 penalty by default). We applied Grid Search Cross-Validation to select the optimal hyperparameters from the grid including: 1) `C` for regularization control, with smaller values implying stronger regularization; 2) `class_weight` to address class imbalances, either as 'none' for equal weights or 'balanced' for automatic adjustment based on class frequencies; and 3) `multi_class` defining the strategy for multiclass problems, with options of one-vs-rest or multinomial classes. Afterwards, the optimal model was fit using the training data, resulting in an accuracy of 99.36%.

1.3.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) leverages Bayes Rule and conditional densities to establish a linear decision boundary. It operates under the assumption of equal covariance matrices for all classes. With no hyperparameters to tune, Grid Searches were unnecessary. The model demonstrated a high accuracy of 98.57%.

1.3.3 Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) relaxes the equal covariance matrix assumption of Linear Discriminant Analysis (LDA). QDA can outperform LDA in scenarios where covariance matrices vary and if there are sufficient observations for accurate estimation. Indeed, an accuracy score of 99.26% is a significant improvement over its linear counterpart.

1.3.4 Naïve Bayes

Naive Bayes is a straightforward and efficient supervised learning algorithm, known for its speed and accuracy, particularly excelling on large datasets. The Gaussian Naive Bayes variant, tailored for continuous data and real-valued features modelled using a normal distribution, proves advantageous for our multiclass classification task. The reported accuracy of 96.69% is the lowest among the models considered, which suggests that the "naive" assumption of conditional independence between pairs of features is violated.

1.3.5 K Nearest Neighbours (KNN)

In K-Nearest Neighbors (KNN), the class label of a new data point is determined by the majority class among its k nearest neighbors. Hyperparameter tuning is conducted, exploring different values for k (`n_neighbors`), distance weighting (weights with options 'uniform' or 'distance'), distance metric (p for L1 or L2), and metric type (metric for 'euclidean' or 'manhattan'). The accuracy of 99.56% attests to the effectiveness of the selected hyperparameters, particularly the use of 'distance' for weights, in enhancing the model's predictive performance. We also graphically inspected the relationship between the error rate and the number of neighbours and the resulting graph is available as figure 21 in the appendix.

1.3.6 Decision Tree Classifier

The Decision Tree classifier utilizes a hierarchical tree structure to recursively partition the dataset based on features, striving to optimize homogeneity within resulting subsets. Hyperparameter tuning, includes considerations for the splitting criterion (criterion), the strategy for selecting splits at nodes (splitter with options 'best' or 'random'), the maximum depth of the tree (`max_depth`), minimum samples required to split an internal node (`min_samples_split`), and minimum samples required at a leaf node (`min_samples_leaf`). This systematic grid search, yielding an impressive accuracy of 99.56%, underscores the discerning impact of hyperparameter choices on precision and model efficacy in decision tree classification.

1.3.7 Random Forest

The Random Forest algorithm, a potent tree ensemble method, elevates predictive accuracy by training each tree on a random subset of the dataset and selecting features randomly at each split. To optimize model performance, a grid search is employed, exploring hyperparameters such as the number of trees in the forest (`n_estimators`), splitting criterion (criterion with options 'gini' or 'entropy'), maximum tree depth (`max_depth`), minimum samples for internal node split (`min_samples_split`), minimum samples for leaf nodes (`min_samples_leaf`), and the number of features for optimal splits (`max_features` with choices 'sqrt', 'log2', or None). The tuned model achieves a remarkable accuracy of 99.75%, complemented by a robust AUC-ROC score of 99.84%. A visualisation showcasing the relationship between the number of trees and the error rate is provided as figure 22 in the appendix.

1.3.8 XG Boosting

Extreme Gradient Boosting is employed to iteratively enhance decision trees based on prior performance. Utilizing accuracy as the judging metric, a grid search is conducted to find optimal hyperparameters, including learning rate, boosting rounds (`n_estimators`), maximum tree depth (`max_depth`), observation fraction for each tree (subsample), and column fraction for each tree (`colsample_bytree`). The tuned model achieves an accuracy of 99.70%, with a remarkable AUC-ROC score of 99.98%. A visualisation showcasing the relationship between the number of trees and the error rate is provided as figure 23 in the appendix.

1.3.9 Support Vector Machine (SVM)

SVM classification is applied to the data, adept at handling multiple categorical variables by constructing a hyperplane in multidimensional space to separate classes. The iterative process of generating an optimal hyperplane minimizes errors, with the core objective of selecting a hyperplane with maximum margin between support vectors. This involves a Grid Search Cross Validation for tuning hyperparameters: 1) The choice of kernel functions, such as linear, polynomial, or radial basis function (RBF), for transforming input data; 2) The penalty parameter (`C`) for balancing misclassification and regularization, determining hyperplane margin size; and 3) `Gamma`, influencing the flexibility of the hyperplane and the risk of overfitting by regulating the impact of training data points. The tuned SVM model optimally navigates this parameter space, enhancing its classification accuracy to an exceptional value of 99.45%.
Note: The optimal parameters for all models are presented as figure 19 in the appendix. Plots of ROC curves are also provided in the appendix.

1.4 Results and Conclusions

Model	Accuracy Score	F1 Score	ROC AUC Score	Time Taken
Random Forest	99.75%	99.16%	99.97%	0.296 s
XGBoost	99.70%	98.99%	99.99%	0.548 s
KNN	99.56%	98.51%	99.97%	0.001 s
Decision Tree	99.56%	98.62%	99.55%	0.009 s
SVM	99.46%	98.20%	99.98%	0.405 s
Logistic Regression	99.36%	97.85%	99.99%	4.085 s
QDA	99.26%	97.62%	99.93%	0.016 s
LDA	98.57%	95.70%	99.97%	0.060 s
Naive Bayes	96.69%	90.53%	99.73%	0.003 s

Figure 1: Model Score Data

As we can see in the table above, the Random Forest model is the best algorithm to use for occupancy estimation, achieving a high accuracy score of 99.75%. This learning method aggregates predictions from 100 decision trees (`n_estimators`: 100), each growing without a maximum depth restriction (`max_depth`: **`None`**). The criterion is finely tuned to 'entropy' for enhanced information gain, and `max_features` is set to 'sqrt' to ensure diversity in feature consideration at each split. The absence of a maximum depth restriction allows the model to capture intricate patterns in the data effectively. The hyper-parameters tuned to include `min_samples_leaf` (set to 1) and `min_samples_split` (set to 2), enhances the overall accuracy and effectiveness of the Random Forest model. The predictive importance of each feature is summarised in the image below.

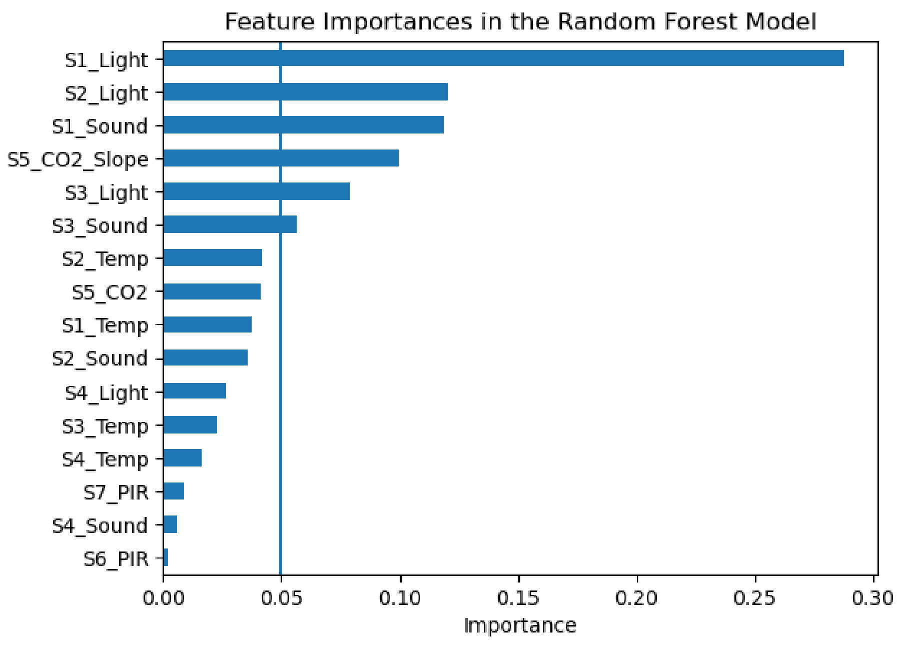


Figure 2: Feature Importance in the Random Forest Model

In conclusion, the Random Forest model, with hyper-parameters selected via Grid Search Cross Validation, stands as the best choice for accurate and interpretable prediction of room occupancy.

Question 2: Coordinate Descent Algorithm for Solving the Lasso Problem

The second part of the project is focused on generating the coordinate descent algorithm and implementing it to solve the Lasso problem and the Elastic Net. Our aim is to correctly implement a one-at-a-time coordinate descent algorithm on Lasso and Elastic Net, and construct a simulation to demonstrate dominance of Elastic Net over Lasso on prediction and feature selection.

We will first generate a data set with parameters given on the project guideline. Then we would construct the coordinate descent algorithm and implement the one-at-a-time approach on Lasso and Elastic Net. Next, we would run simulations with training / validation / test sets on Lasso and Elastic Net to compare the results. Lastly, we will run various simulations with different settings (change in n, p, pairwise correlation ρ, etc) to further prove the superiority of Elastic Net over Lasso.

2.1 Data Generating Process

We would first generate necessary data for simulation. The simulated data is a sample of multivariate normal distribution with the following mathematical equation as given on project guideline:

$$\mathbf{y} = \mathbf{X}\beta + \sigma\varepsilon, \quad \varepsilon \sim N(0, \mathbf{I}_n).$$

The initial parameters are followed as introduced in the project guidelines:

- number of observations n as 200
- number of predictors p as 8
- pairwise correlation $\rho^{|i-j|}$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ with $\rho = 0.5$
- error sigma σ as 3
- true beta β as a transpose of vector (3, 1.5, 0, 0, 2, 0, 0, 0)

We construct a function to produce the data with given parameters. The mathematical steps of the function are the following:

1. Create an 8 by 8 correlation matrix with pairwise correlation coefficient ρ between $\mathbf{x}_i$ and $\mathbf{x}_j$ set to $0.5^{|i-j|}$.

2. Generate 200 multivariate normal samples with mean 0 and standardised correlation matrix as the covariance matrix Sigma.

The resulting data is be a 200 by 8 matrix where each row represents an observation and each column a feature $\mathbf{x}_i$. The following code is executed to produce the data:

```r
generate_X_data <- function(n, p, pair_rho, has_intercept) {
  Sigma_X <- matrix(pair_rho, nrow = p, ncol = p)
  for(i in 1:p) {
    for(j in 1:p) {
      Sigma_X[i,j] <- 0.5^(abs(i-j))
    }
  }
  X <- mvrnorm(n, mu = rep(0, times = p),
               Sigma = Sigma_X,
               empirical = TRUE)
  if (has_intercept) {
    X <- cbind(1, X)
    p <- p + 1
  }
  return(X)
}

X <- generate_X_data(n, p, pair_rho, has_intercept)
error_term <- rnorm(n, mean = 0, sd = error_sigma)
y <- X %*% true_b + error_term
```

2.2 Algorithm Implementation

2.2.1 Introduction to Lasso

Lasso (Least Absolute Shrinkage and Selection Operator) regression is a type of linear regression technique that implements an additional penalty term on the original least-squares objective function in order to avoid certain cases such as over fitting. The mathematical expression for the Lasso problem from [1] can be expressed as the following:

$$f(\beta) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \gamma\sum_{j=1}^{p}|\beta_j|,$$

As it is an least-squares objective function, the objective is to search for beta coefficients β that minimises the function $f()$. As followed from the parameters on section 2.1,n is the number of observations, p is the number of predictors (features), y_i is the response variable for observation, β_j are the regression coefficients for predictors, x_{ij} is the value of predictor j for observation i, and λ is the regularization parameter that controls the strength of the penalty (denoted as γ in the formula above).

As mentioned, the Lasso regression adds the L1 penalty term, which is the sum of absolute values of β multiplied by λ. The resulting choice of beta coefficients would heavily depend on different values of λ.

We can observe the result of linear regression using Lasso on generated data from section 2.1 with existing library `glmnet`. We would later compare our own implementation of one-at-a-time coordinate descent algorithm on Lasso with the result from the `glmnet` library function. The following R code simply executes lasso regression on our data and visualise how betas decrease for different values of λ:

```
fit.lasso <-glmnet(X,y, alpha = 1)
plot(fit.lasso, xvar="lambda", label= TRUE)
```

As we can observe from figure 3, the log lambda values on the x-axis illustrates that betas β decrease as the log of λ increases. Moreover, this validates our data generation steps as the selected betas are in line with the simulation set-up.

2.2.2 Introduction to Elastic Net

The Elastic Net is also a type of linear regression technique like lasso, but with an additional L2 penalty term. It can be considered as a combination of Lasso and Ridge regression, offering the benefits of both techniques. The mathematical expression for Elastic Net from [1] can be expressed as the following:

$$\frac{1}{2n}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}\beta_j^2.$$

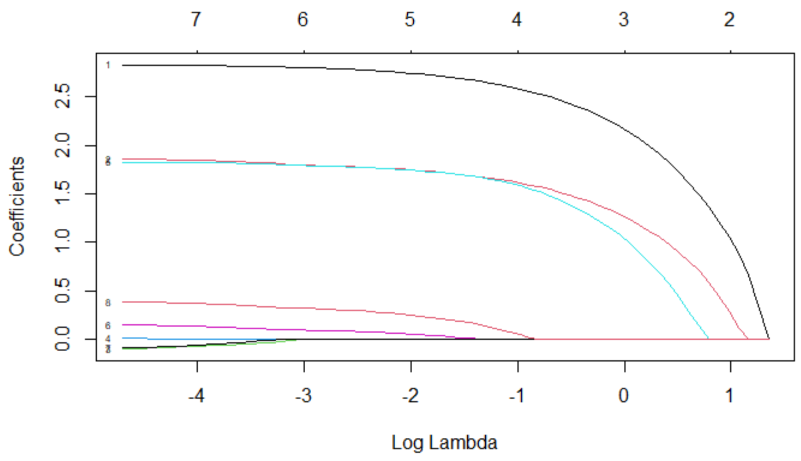


Figure 3: Log λ Beta selection plot of Lasso with `glmnet` function

The additional penalty term is calculated as the sum of squares of β coefficients multiplied by the λ_2 value. This term is also known as the penalty for Ridge regression.

As similar for Lasso, We can observe the result of Elastic Net on generated data from section 2.1 with existing library `glmnet`. The following R code simply executes Elastic Net on our data and visualise a beta convergence plot:

```
fit.lasso <-glmnet(X,y, alpha = 0.5)
plot(fit.lasso, xvar="lambda", label= TRUE)
```

Notice the difference in the mixing $alpha$ parameter compared to Lasso problem on section 2.2.1. Our elastic net implementation looks at λ_{L1} and λ_{L2} independently, hence the results may not be exactly comparable. Additionally, `glmnet` uses log of λ for regularisation purposes, whereas we look at the λ itself. We can observe the trend beta coefficients similar to Lasso but with different rate of converging to 0 as log of λ increases.

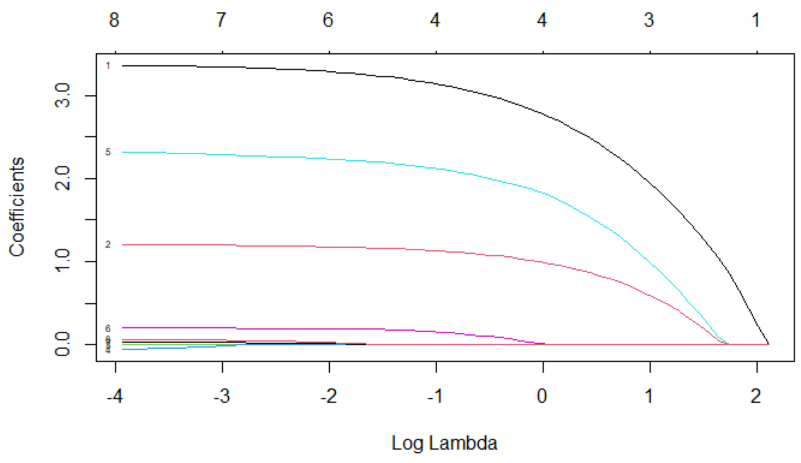


Figure 4: Log λ Beta selection plot of Elastic net with `glmnet` function

2.2.3 Introduction to Coordinate Descent algorithm

Coordinate Descent is an optimization algorithm used for minimizing the objective function in various machine learning models, particularly in the context of regularized linear regression problems like Lasso and Ridge. The algorithm iteratively optimizes each parameter while holding all others fixed, making it particularly well-suited for problems with a separable objective function.

The main advantage of Coordinate Descent is that each parameter update involves solving a univariate optimization problem, which can often have a closed-form solution or be solved efficiently. This makes the algorithm computationally attractive, especially in situations with a large number of features.

In this project we would construct the Coordinate Descent algorithm to solve the Lasso and Elastic Net. We can re-write the standard lasso formulation into the below equation from (Friedman, 2007)[1], which we will then minimise w.r.t each β_j, holding the remaining betas fixed.

$$f(\tilde{\beta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sum_{k\neq j} x_{ik}\tilde{\beta}_k - x_{ij}\beta_j\right)^2 + \gamma\sum_{k\neq j}|\tilde{\beta}_j| + \gamma|\beta_j|,$$

Which is a simple iterative algorithm that applies soft-thresholding with partial residual as a response variable. A step-by-step process of the Coordinate Descent algorithm would be the following:

1. Initialize the parameter vector (β coefficients) as 0 for each β_j, $j = 1, .., p$.
2. Repeat k times until until convergence is occurred, defined as a change between iterations for each β_j being lower than 1-e6, or a specified number of iterations is reached.
   - 2a) Compute the partial residual times $x_{i,j}$ for the j^{th} loop defined as $r_{i,j} = \beta_j + x_{i,j}\left(y_i - \sum_{l=1}^{p} x_{i,l}\beta_l\right)$. Please note, that this is a modification of the project guidelines given we multiply by $x_{i,j}$ directly in this step and add back β_j given that (Friedman, 2007)[1] define this approach as equivalent in their paper.
   - 2b) Compute the simple least squares coefficient of these residuals on the j^{th} predictor β_j^* by taking an average of the vector of partial residuals times $x_{i,j}$ defined in the point above.
   - 2c) Update β_j by soft thresholdidng the predictor value.
     - 2a) In case of lasso, it is $\beta_j = \text{sign}(\beta_j^*)(|\beta_j^*| - \lambda_{L1})_+$.
     - 2b) In case of elastic net, we implement $\beta_j = \text{sign}(\beta_j^*)(|\beta_j^*| - \lambda_{L1})_+(1 + 2\lambda_{L2})^{-1}$. Note, that the soft threshold function is the same for both lasso and elastic net if λ_{L2} is equal to 0.

Our implementation of the Coordinate Descent code is the following (please note, that the full version of the code contains additional output analysis features):

```
coordinate_descent <- function(X, y,
                               lambda_l1, lambda_l2,
                               is_Lasso, max_iter, tol=1e-6) {
  p <- ncol(X)
  n <- min(nrow(X), nrow(y))
  betas_star <- rep(0, times = p)
  betas_history <- betas_star

  if (is_lasso) {lambda_l2 = 0}

  for(iter in 1:max_iter) {
    for (j in 1:p) {
      rho_j <- rep(0, times=n)
      for(i in 1:n) {rho_j[i] <- betas_star[j] +
                                  X[i,j]*(y[i] - (X[i,] %*% betas_star))}
      beta_star <- mean(rho_j)
      betas_star[j] <- (sign(beta_star)*
                         max((abs(beta_star) - lambda_l1),0))/(1+2*lambda_l2)

      if (j == p) {
        prev_betas <- tail(betas_history,1)
        betas_history <- rbind(betas_history, betas_star)
      }
    }

    if (max(abs(betas_star - prev_betas)) < tol) {
      betas_history <- rbind(betas_history, betas_star)
      break
    }
  }

  return(betas_star)
}
```

Notice the parameter `is_lasso` implemented so the function could be implemented on both Lasso and Elastic Net, depending on the parameter value. If `is_lasso` is set as **`FALSE`**, then λ_{L2} is overwritten with 0 prior to running the algorithm. The result of implementing one-at-a-time Coordinate Descent algorithm on Lasso with λ_{L1} = 0.2 is the following:

```
 - - - - - - - - - - - - - - - -
Converged at Iteration: 13
MSE: 7.769564
Betas: 2.702118 1.712317 0 0 1.708927 0.02608155 0 0.2013765
 - - - - - - - - - - - - - - - -
```

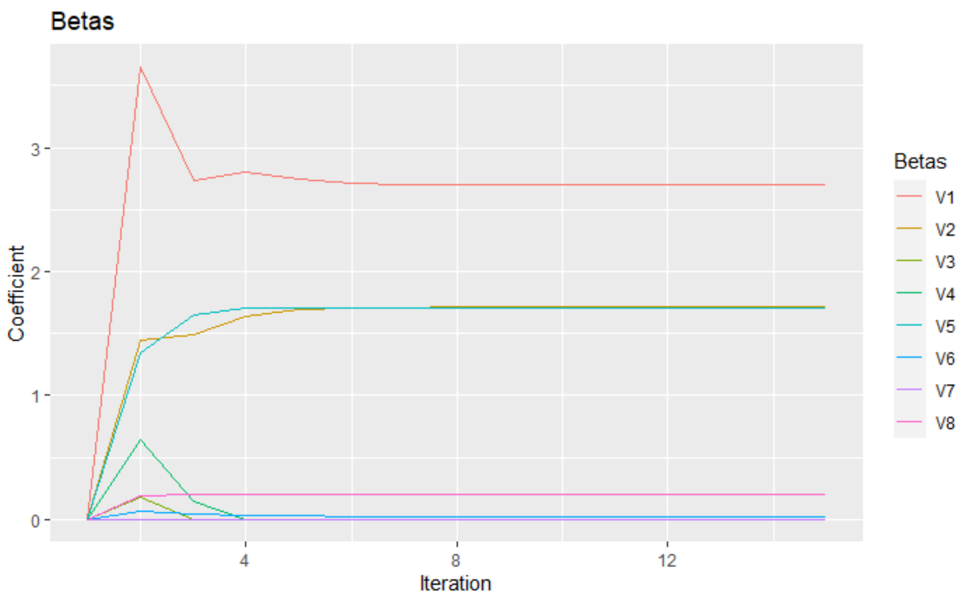


Figure 5: Beta convergence plot of Lasso with Coordinate Descent algorithm

We can observe the convergence of beta coefficients after several iterations.

The result of implementing one-at-a-time Coordinate Descent algorithm on Elastic Net with example values of λ_{L1} = 0.2, λ_{L2} = 0.2 is the following:

```
 - - - - - - - - - - - - - - - -
Converged at Iteration: 11
MSE: 9.189617
Betas: 2.008692 1.430039 0.1428643 0.1792577 1.120479 0.2196379 0 0.1574192
 - - - - - - - - - - - - - - - -
```

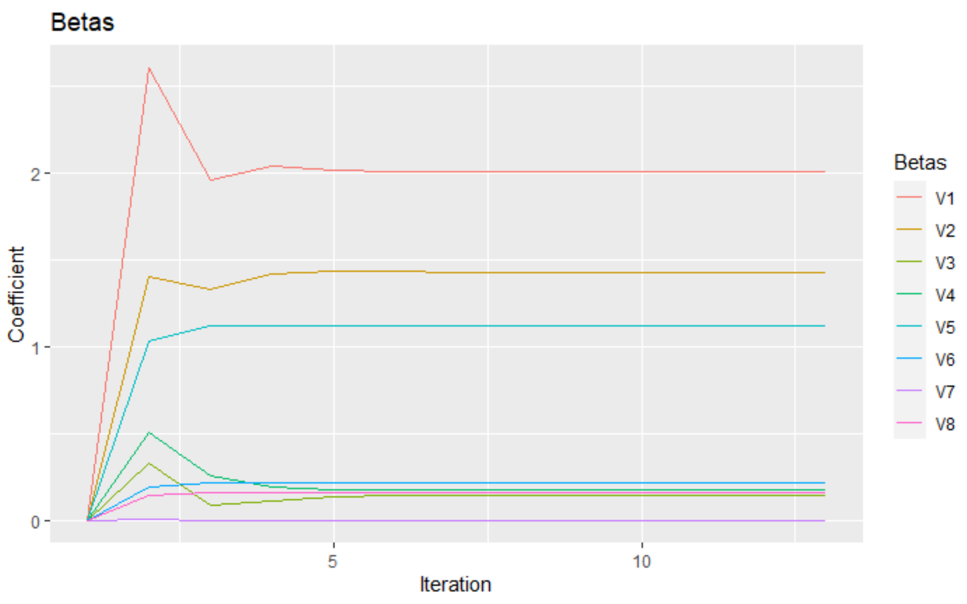


Figure 6: Beta convergence plot of Elastic Net with Coordinate Descent algorithm

Notice that the case of Elastic Net has higher MSE compared to the Lasso case. This is a demonstration of the coordinate descent algorithm with example, rather than tuned, hyperparameters. Hence, it is plausible to see that the result from Elastic Net is not yet optimized. We will observe the effect of Elastic Net by running simulations on the next section.

2.3 Simulation

With the constructed Coordinate Descent algorithm from section 2.2.3 the initial simulation is conducted with the following parameters:

- Number of hyperparameter tuning iterations set to 50.
- The train, validation and test data set each has 20, 20, 100 observations
- The true β is set to a vector $(3, 1.5, 0, 0, 2, 0, 0, 0)^T$
- The error sigma σ is set to 3
- The pairwise correlation $\rho^{|i-j|}$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ with $\rho = 0.5$

The code of the simulation functions is as follows (please note, that this is an extract for the illustrative purposes and the full version of the code contains additional output analysis features):

```r
Simulation <- function(n_train, n_test, n_validation, p,
                       pair_rho, error_sigma, is_lasso, true_b,
                       lambdas_l1_grid, lambdas_l2_grid,
                       max_reg_iter=50, num_tuning_iter,
                       verbose=TRUE, plot=TRUE) {

  if( is_lasso ) {lambdas_l2_grid <- c(0)}

  mse_df <- data.frame()
  betas_matrix <- c()

  for(i in 1:num_tuning_iter) {
    betas_list <- c()

    X_test <- generate_X_data(n_test, p, pair_rho, has_intercept)
    error_term_test <- rnorm(n_test, mean = 0, sd = error_sigma)
    y_test <- X_test %*% true_b + error_term_test

    X_train <- generate_X_data(n_train, p, pair_rho, has_intercept)
    error_term_train <- rnorm(n_train, mean = 0, sd = error_sigma)
    y_train <- X_train %*% true_b + error_term_train

    X_val <- generate_X_data(n_validation, p, pair_rho, has_intercept)
    error_term_val <- rnorm(n_validation, mean = 0, sd = error_sigma)
    y_val <- X_val %*% true_b + error_term_val

    for(lambda_l1 in lambdas_l1_grid) {
      for(lambda_l2 in lambdas_l2_grid) {
        betas_hat <- coordinate_descent(X_train, y_train, lambda_l1 = lambda_l1,
                                        lambda_l2 = lambda_l2, is_lasso= is_lasso,
                                        max_iter = max_reg_iter, tol=1e-6,
                                        plot_betas_convergence = FALSE,
                                        verbose = FALSE)
        y_val_hat <- X_val %*% betas_hat
        mse_val <- mean((y_val - y_val_hat)^2)
        mse_df <- rbind(mse_df, c(mse_val, lambda_l1, lambda_l2, i, 'Validation'))

        y_test_hat <- X_test %*% betas_hat
        mse_test <- mean((y_test - y_test_hat)^2)
        mse_df <- rbind(mse_df, c(mse_test, lambda_l1, lambda_l2, i, 'Test'))

        y_train_hat <- X_train %*% betas_hat
        mse_train <- mean((y_train - y_train_hat)^2)
        mse_df <- rbind(mse_df, c(mse_train, lambda_l1, lambda_l2, i, 'Train'))

        betas_matrix <- rbind(betas_matrix, c(betas_hat,lambda_l1,lambda_l2, i) )
      }
    }
  }

  #### Summarise MSEs ####
  colnames(mse_df) <- c('MSE', 'LambdaL1', 'LambdaL2', 'Iteration', 'Set')
  mse_df[, c('MSE', 'LambdaL1', 'LambdaL2', 'Iteration')] <- lapply(
    mse_df[, c('MSE', 'LambdaL1', 'LambdaL2', 'Iteration')], as.numeric
  )

  agg_mse_df <- mse_df %>%
    group_by(Set, LambdaL1, LambdaL2, .drop = FALSE) %>%
    summarise(
      MSE_mean = mean(MSE, na.rm = TRUE),
      MSE_se = se(MSE),
      .groups = 'drop'
    )
  agg_mse_df <- as.data.frame(agg_mse_df)
  agg_mse_df_val <- agg_mse_df[agg_mse_df$Set == 'Validation',]

  min_row <- agg_mse_df_val[which.min(agg_mse_df_val$MSE_mean), ]
  lambda_l1_hat <- min_row$LambdaL1
  lambda_l2_hat <- min_row$LambdaL2
  min_val_mse <- min_row$MSE_mean

  min_test_row <- agg_mse_df[(agg_mse_df$Set == 'Test')&(
    agg_mse_df$LambdaL1 == lambda_l1_hat)&(agg_mse_df$LambdaL2 == lambda_l2_hat),]
  min_test_mse <- min_test_row$MSE_mean

  betas_df <- as.data.frame(betas_matrix)
  colnames(betas_df) <- c(colnames(betas_df)[1:8], 'LambdaL1', 'LambdaL2', 'Run')
  average_betas <- betas_df %>%
    group_by(LambdaL1, LambdaL2, .drop = FALSE) %>%
    summarise(across(everything(), mean))
  average_betas <- select(average_betas, -one_of('Run'))

  betas_star <- average_betas[(average_betas$LambdaL1 == lambda_l1_hat)&(
    average_betas$LambdaL2 == lambda_l2_hat),]
  cols_to_drop <- c('LambdaL1', 'LambdaL2')
  betas_star <- subset(betas_star, select = -which(names(betas_star) %in% cols_to_drop))

}
```

The function is designed with the primary objective of conducting simulations for tuning Lasso and Elastic Net regression models. The input parameter `is_lasso` works as an indicator of whether we want to perform on Lasso or Elastic Net, similar to the case of section 2.2.3.

The function first generates data sets for training, test, and validation with initialized parameters by calling the data generating function described on section 2.1. We are sampling train, test, and validation datasets separately given that the data generating process is known rather than limited to one sample, and each observation is drawn randomly from a multivariate and univariate normal and independently to the past sampled values. Subsequently, the function enters a coordinate descent loop. It loops over tuning iterations, exploring combinations of λ_{L1} and λ_{L2} regularization parameters. The coordinate descent algorithm described in section 2.2.3 is applied to fit the model on the training data. Model evaluation is performed on validation and test sets, calculating mean-squared errors (MSEs), where the optimal λ_{L1} and λ_{L2} are those minimising the validation MSE. MSEs, along with corresponding regularization parameters and iteration numbers, are recorded for further analysis.

The function then computes the mean and standard error of MSEs for different sets (validation, test, and train). Optimal λ values are selected based on the minimum validation MSE.

2.3.1 Simulation Result on Lasso

The result of simulating on Lasso is the following:

```
L1 Lambda: 0.4
L2 Lambda: 0
Validation MSE: 11.1643
Test MSE: 11.55709
Betas: V1: 2.62429446920973, V2: 1.23001036398328,
V3: 0.185056138590908, V4: 0.14088002313666,
V5: 1.51463846425966, V6: 0.204415471533245,
V7: 0.0323494121068114, V8: -0.0716170220559618
```

We can observe from the data that L1 lambda value of 0.4 suggests a moderate level of Lasso regularization, indicating that some coefficients are likely to be close to zero. The Mean Square Measure values for Validation and Test simulations indicates the model's likeliness of predicting the data. With MSE value being 11.1643 and 11.55709 each for Validation and Test, it is reasonable to assume consistent performance on both of the simulations. L2 lambda is always set to 0 and hence irrelevant in lasso optimisation.

The estimated values for β coefficients shows that it follows the general trend of true β value of (3, 1.5, 0, 0, 2, 0, 0, 0), showing adequate strength in feature selection.

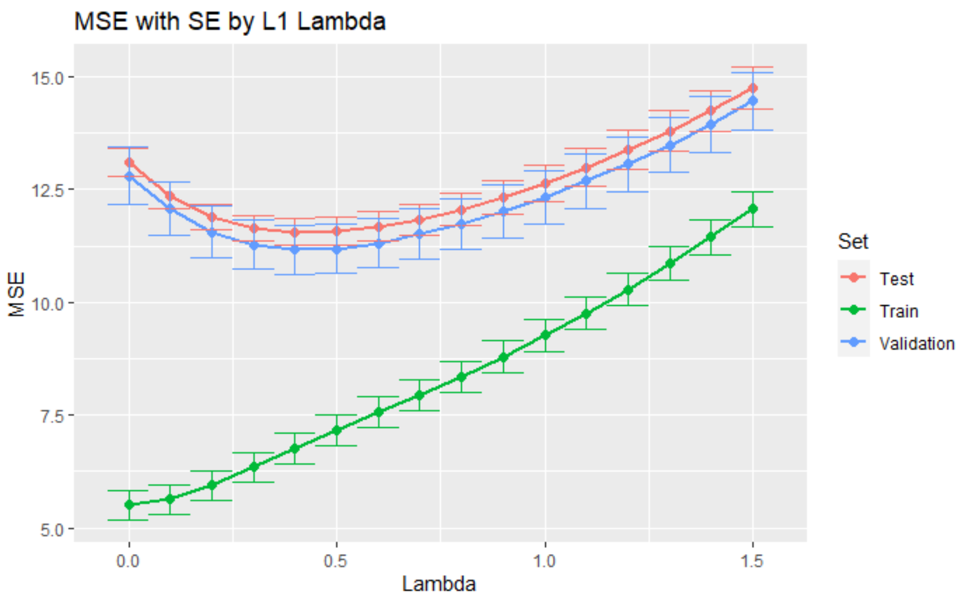


Figure 7: MSE of test/train/validation data simulation on Lasso

The figure above illustrates the MSE of three data set (Test, Train, Validation) from the simulation. As we can observe, the MSE value of Validation set follows closely with the Test set, and we can check that the λ value that minimises the MSE for Validation and Test is indeed 0.4.

We can also see on figure 8 how betas converge to 0 as the lambda increases. However, given that we have selected a lower value of λ_{L1} to minimise the validation MSE, some of the betas that truly are zero, are not selected as such in this optimisation.

We would run the simulation several more times to observe if the Lasso can generate consistent outcomes on different random samples.

The below table illustrates five simulations with the same parameters on Lasso. As we can observe, the simulations perform consistent results on every feature. λ is chosen at similar range, MSE for Validation and Test both remains on the similar boundaries, the β coefficients follows the similar trend.

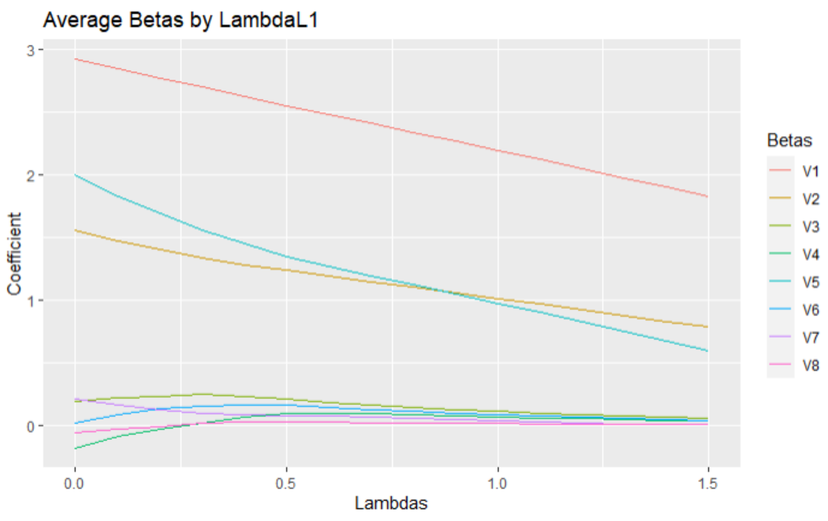


Figure 8: Average Betas across simulations by Lambda

#	L1	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.5	11.2909	12.10153	2.56	1.25	0.26	0.18	1.15	0.14	0.03	0.04
2	0.4	11.41159	11.33256	2.42	1.29	0.13	0.07	1.56	0.12	0.06	-0.05
3	0.5	11.57419	11.48956	2.57	1.23	0.13	0.19	1.26	0.18	0.01	0.01
4	0.5	10.54827	11.15051	2.55	1.16	0.12	0.17	1.39	0.09	0.00	-0.05
5	0.4	11.07785	11.35498	2.64	1.22	0.18	0.14	1.47	0.18	0.08	0.02

Figure 9: Multiple simulations on Lasso

2.3.2 Simulation Result on Elastic Net

The result of simulating on Elastic Net is the following:

```
L1 Lambda: 0.2
L2 Lambda: 0.1
Validation MSE: 12.15922
Test MSE: 11.35888
Betas: V1: 2.34314664631031, V2: 1.34073145916161,
V3: 0.333325833482702, V4: 0.154425234353689,
V5: 1.25929002879972, V6: 0.196431795869151,
V7: -0.0740256513138841, V8: 0.084071907304069
```

From the data we can observe the difference with the case of Lasso. L2 λ parameter is chosen with value 0.1 for optimal regression, and compared to the case with Lasso the L1 λ parameter is chose with smaller value of 0.2.

The estimated values for β coefficients also shows that it follows the general trend of true β value of (3, 1.5, 0, 0, 2, 0, 0, 0), showing adequate strength in feature selection.

The figure 10 above illustrates the Validation MSE of Elastic Net simulation with different λ settings. We can notice that the pair of L1 L2 λ values that minimizes the Validation MSE matches the result above, and is circled in red on the figure.

We can also see on figure 11 how betas change on average for both λ_{L1} and λ_{L2}. The former is a straight line decrease in line with the linearity of the term, λ_{L2} shape shows a classic ridge penalty shape with a convex decrease towards 0 to it.

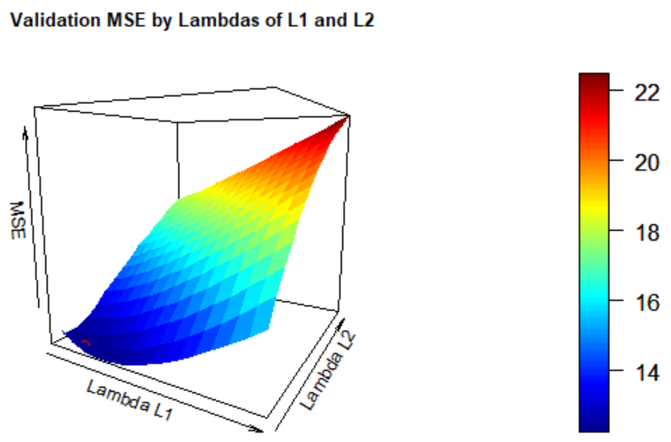
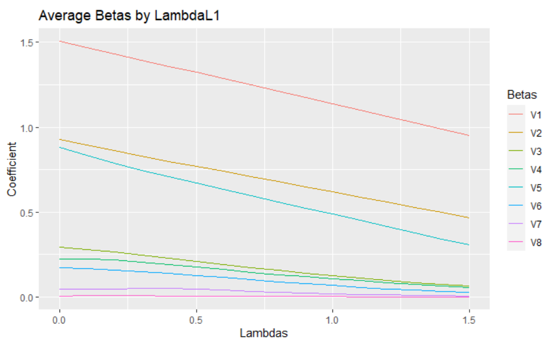
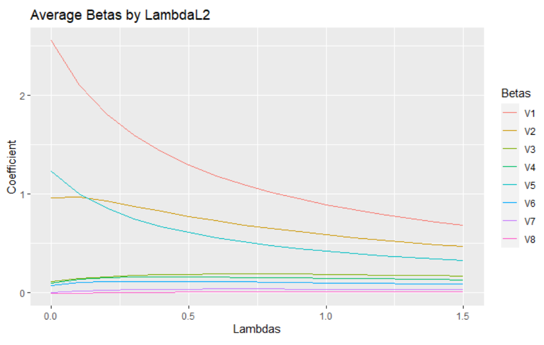


Figure 10: MSE of test/train/validation data simulation on Elastic Net. The red dot indicates the selected model.



(a) Average Betas across simulations by LambdaL1



(b) Average Betas across simulations by LambdaL2

Figure 11: Average Betas selected per tested λ_{L1} and λ_{L2}.

Similar to Lasso case, we would run the simulation several more times to observe if the Elastic Net also can generate consistent outcomes on different random samples.

#	L1	L2	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.1	0.1	12.34836	11.74359	2.28	1.40	0.14	0.07	1.56	0.16	0.03	0.03
2	0.2	0.1	11.24703	11.54194	2.45	1.23	0.05	0.18	1.26	0.11	0.04	-0.03
3	0.4	0	11.55484	11.23316	2.70	1.33	0.19	0.11	1.49	0.11	-0.01	0.02
4	0.4	0	10.5401	11.02945	2.66	1.30	0.18	0.25	1.39	0.06	0.06	0.02
5	0.1	0.1	11.35586	11.74787	2.38	1.28	0.11	0.25	1.37	0.21	0.05	0.02

Figure 12: Multiple simulations on Elastic Net

The above table illustrates five simulations with the same parameters on Elastic Net. As we can observe, the simulation results in similar boundaries for Validation and Test MSE, the β coefficients follows the similar trend, but the L1 L2 λ appear in two different combinations of values. In cases where L2 λ is zero as this additional regularisation was not needed, the Elastic Net shows similar results to Lasso.

A logical guess for the reason that the results vary for simulations lies in parameter settings. When an additional L2 regularisation term is not needed, the performance between the two methods will be similar. However, we expect to observe the full benefit of Elastic Net and added L2 regularisation in cases where some level of multicollinearity is present in the data (e.g. pairwise correlation between variables is high, or n is close in size to p.

2.4 Additional simulations

We would run additional simulations with different parameter settings to identify the flexibility and dominance of Elastic Net over Lasso on prediction accuracy and variable selection.

2.4.1 Case of pairwise correlation of 0.8

We would first try changing the pairwise correlation of the generating data to a higher value and observe the difference on the result.

#	L1	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.4	10.8616	11.28632	2.80	1.10	0.25	0.33	1.16	0.21	0.04	0.11
2	0.4	11.70673	11.21835	2.61	1.39	0.22	0.22	1.30	0.20	0.11	-0.04
3	0.4	11.48796	11.32343	2.73	1.11	0.35	0.21	1.32	0.22	-0.00	0.08

Figure 13: Simulation with pairwise correlation changed on Lasso

#	L1	L2	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.2	0.1	11.52885	11.18835	2.31	1.31	0.32	0.31	1.17	0.33	0.11	0.06
2	0.4	0	11.27332	10.87261	2.66	1.08	0.23	0.23	1.34	0.19	0.05	0.06
3	0.2	0.1	10.82427	11.67103	2.31	1.27	0.40	0.25	1.11	0.24	0.14	0.04

Figure 14: Simulation with pairwise correlation changed on Elastic Net

Again we can observe the similar result from our previous simulation. Elastic net has two different combination of L1 L2 λ values. In both cases the resulting MSE for Validation and Test lies on similar boundaries, both methods have good enough strength for variable selection.
However, the grid search for both $\lambda L2$ and $\lambda L1$ is done in the interval from 0 to 1.5 by 0.1 steps. We can observe in the table 14 that the selected $\lambda L2$ is often equal to 0 or 0.1. It is quite common to observe very small $\lambda L2$ of magnitude 1e-3, hence we likely could further benefit from the superiority of the Elastic Net by conducting a grid search of higher granularity. This is also where the efficiency of `glmnet` comes through given the operation in log of λ space.

2.4.2 Case of $n = p$

Next we will simulate case where n, the number of observations, equals to p, number of predictors.

#	L1	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.4	11.50948	11.67061	2.68	1.18	0.28	0.26	1.38	0.17	0.03	0.04
2	0.4	11.29318	11.32732	2.59	1.22	0.23	0.32	1.37	0.14	0.02	-0.00
3	0.4	11.34436	11.43378	2.67	1.32	0.31	0.25	1.39	0.21	0.03	0.02

Figure 15: Simulation with n = p on Lasso

We can observe both Validation and Test MSE for Elastic Net are on average noticeably less than of Lasso.

#	L1	L2	Validation MSE	Test MSE	$\beta 1$	$\beta 2$	$\beta 3$	$\beta 4$	$\beta 5$	$\beta 6$	$\beta 7$	$\beta 8$
1	0.2	0.1	11.10965	10.9511	2.39	1.33	0.18	0.29	1.20	0.21	0.07	0.06
2	0.1	0.1	11.16403	11.27089	2.43	1.28	0.34	0.36	1.19	0.20	0.14	0.08
3	0.2	0.1	11.18486	11.5731	2.27	1.27	0.25	0.34	1.19	0.27	0.01	0.08

Figure 16: Simulation with n=p changed on Elastic Net

2.4 Conclusion

Throughout question 2 we have generated multivariate normal sample, constructed the Coordinate Descent algorithm, applied one-at-a-time approach of the algorithm on Lasso and Elastic Net, and ran simulations with test, train, validation data set on Lasso and Elastic net on various parameter settings.

Through the simulation results we have observed the dominance of Elastic Net over Lasso on prediction accuracy and variable selection. It provides greater flexibility, robustness to multicollinearity, and in all simulated scenarios performs at least as well as Lasso.

Bibliography

[1] Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2), December 2007.

[2] Adarsh Pal Singh, Vivek Jain, Sachin Chaudhari, Frank Alexander Kraemer, Stefan Werner, and Vishal Garg. Machine learning-based occupancy estimation using multivariate sensor nodes. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2018.

Appendix

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light
mean	25.45	25.55	25.06	25.75	25.45	26.02	34.25	13.22
std	0.35	0.59	0.43	0.36	51.01	67.30	58.40	19.60
min	24.94	24.75	24.44	24.94	0.00	0.00	0.00	0.00
25%	25.19	25.19	24.69	25.44	0.00	0.00	0.00	0.00
50%	25.38	25.38	24.94	25.75	0.00	0.00	0.00	0.00
75%	25.63	25.63	25.38	26.00	12.00	14.00	50.00	22.00
max	26.38	29.00	26.19	26.56	165.00	258.00	280.00	74.00

	S1_Sound	S2_Sound	S3_Sound	S4_Sound	S5_CO2	S5_CO2_Slope	S6_PIR	S7_PIR
mean	0.17	0.12	0.16	0.10	460.86	-0.00	0.09	0.08
std	0.32	0.27	0.41	0.12	199.96	1.16	0.29	0.27
min	0.06	0.04	0.04	0.05	345.00	-6.30	0.00	0.00
25%	0.07	0.05	0.06	0.06	355.00	-0.05	0.00	0.00
50%	0.08	0.05	0.06	0.08	360.00	0.00	0.00	0.00
75%	0.08	0.06	0.07	0.10	465.00	0.00	0.00	0.00
max	3.88	3.44	3.67	3.40	1270.00	8.98	1.00	1.00

Figure 17: Summary Statistics of Features

Model	**Accuracy Score**	**F1 Score**	**ROC AUC Score**	**Time Taken**
Random Forest	99.75%	99.16%	99.97%	0.296 s
XGBoost	99.70%	98.99%	99.99%	0.548 s
KNN	99.56%	98.51%	99.97%	0.001 s
Decision Tree	99.56%	98.62%	99.55%	0.009 s
SVM	99.46%	98.20%	99.98%	0.405 s
Logistic Regression	99.36%	97.85%	99.99%	4.085 s
QDA	99.26%	97.62%	99.93%	0.016 s
LDA	98.57%	95.70%	99.97%	0.060 s
Naive Bayes	96.69%	90.53%	99.73%	0.003 s

Figure 18: Selection of metrics

Model	**Best Parameters**
Random Forest	’criterion’: ’entropy’ ’max_depth’: None ’max_features’: ’sqrt’ ’min_samples_leaf’: 1 ’min_samples_split’: 2 ’n_estimators’: 100
XGBoost	’colsample_bytree’: 0.5 ’learning_rate’: 0.1 ’max_depth’: 10 ’n_estimators’: 100 ’subsample’: 1.0
KNN	’metric’: ’manhattan’ ’n_neighbors’: 5 ’p’: 1 ’weights’: ’distance’
Decision Tree	’criterion’: ’entropy’ ’max_depth’: None ’min_samples_leaf’: 1 ’min_samples_split’: 2 ’splitter’: ’best’
SVM	’C’: 10 ’gamma’: 1 ’kernel’: ’linear’
Logistic Regression	’C’: 100.0 ’class_weight’: None ’multi_class’: ’multinomial’
QDA	-
LDA	-
Naive Bayes	-

Figure 19: Parameters for all models

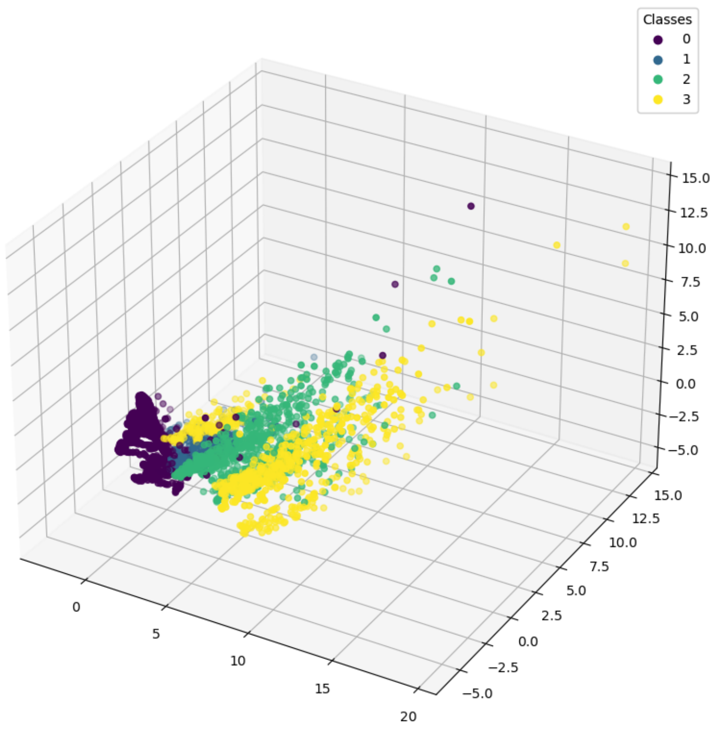


Figure 20: PCA 3D scatterplot

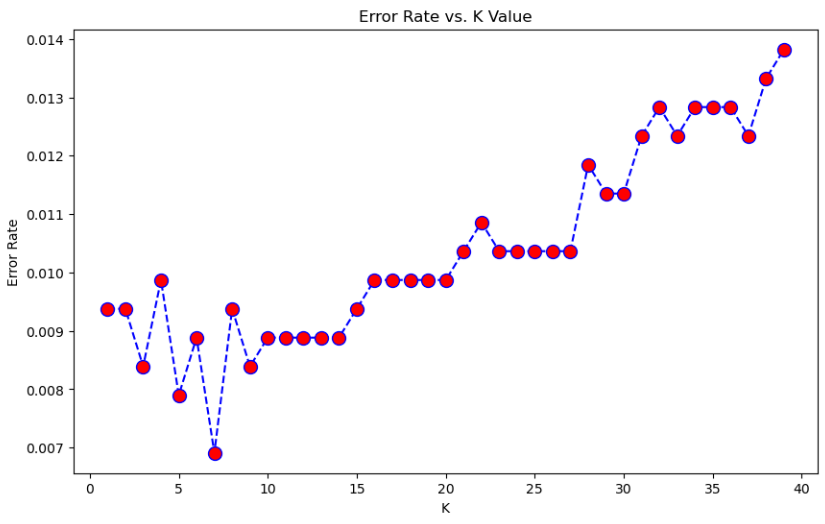


Figure 21: knn error rate vs k

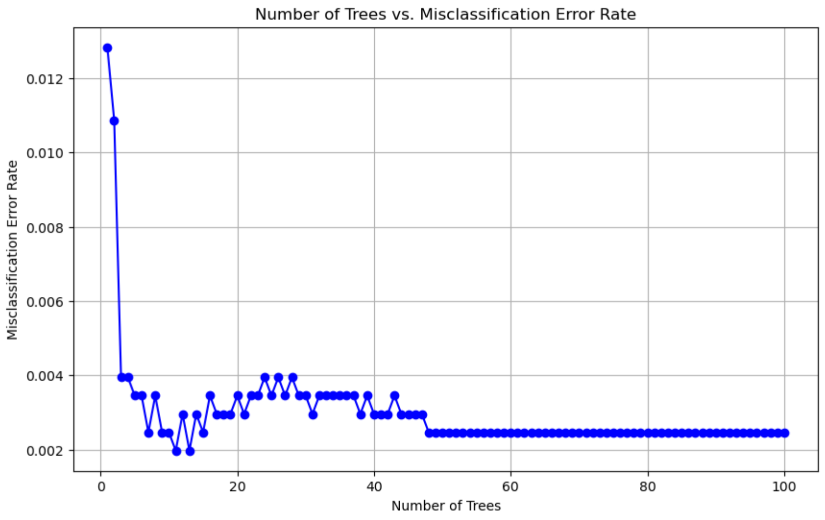


Figure 22: rf ntree error

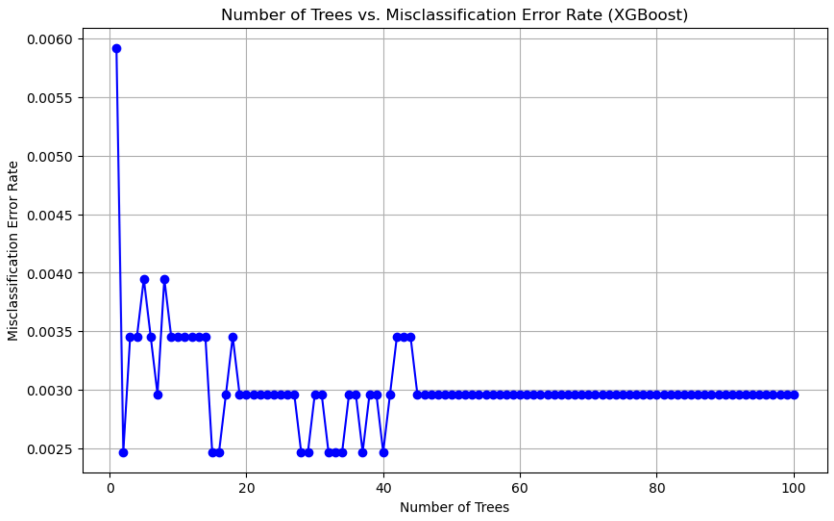


Figure 23: xgb ntree

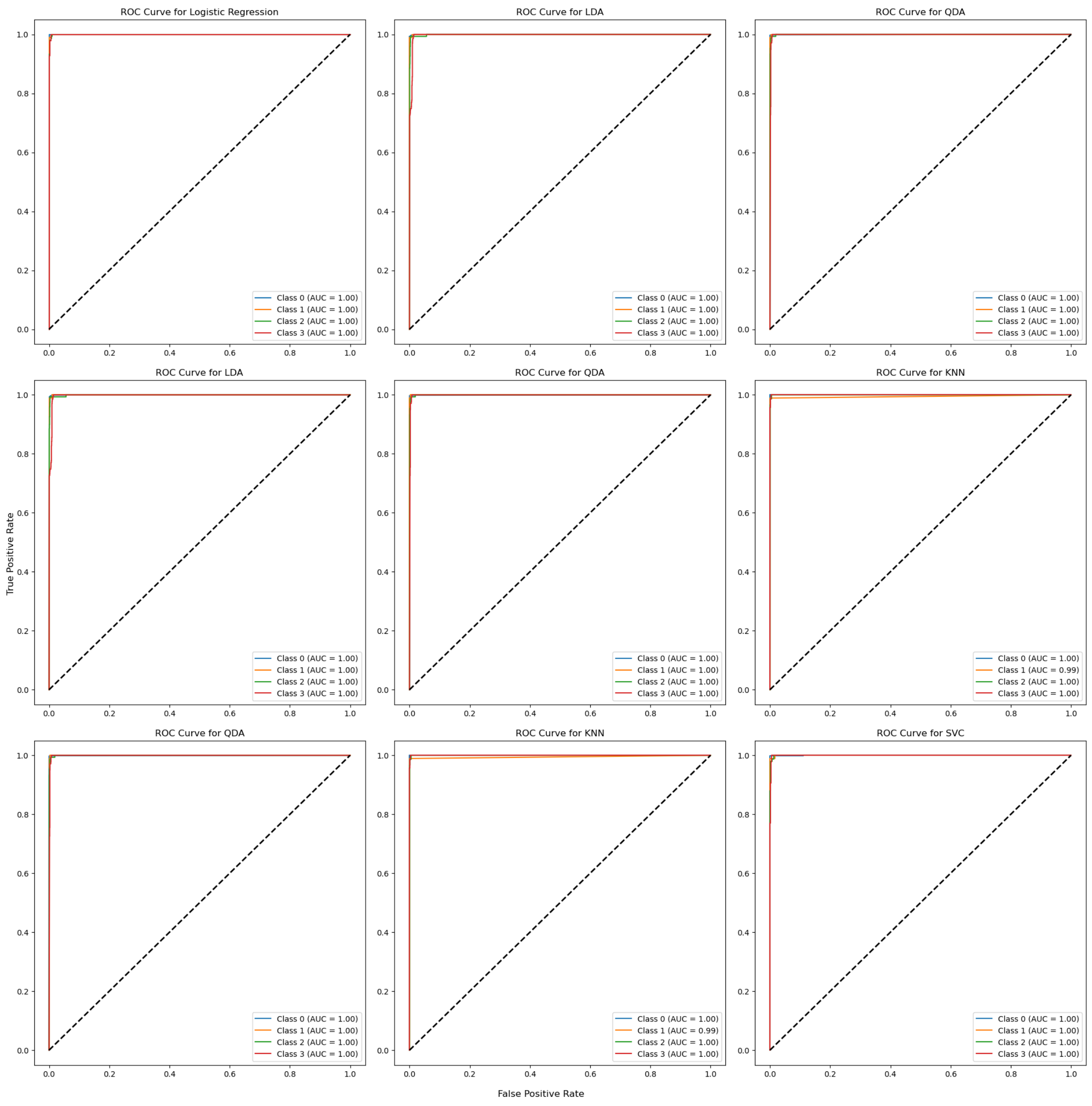


Figure 24: ROC curves