CIS 520 Final Project

Predict Health Status of US counties using Tweets and Socio-Economic factors

TEAM: skynet

December 2018

Authors: Keshav Patil, Zachariah Vicars and Akarsh Varre

Introduction

The project requires us to develop a machine learning model that predicts the health outcomes of the US counties based on a dataset which is designed out of information on tweets by the inhabitants of the counties and other socio-economic factors. We begin by understanding the dataset, applying certain data transformations to it while providing the respective explanations, testing various regression models and also understanding the variable importance and interpretation based on statistical techniques. Finally we develop a novel method that helped us get an appreciably higher accuracy of prediction.

1 Models Implemented

We have tested various regression models and trained them on the dataset (of 1019 counties) that includes predictor variables: 21 Socio-economic factors and 2000 word topic frequencies obtained through Latent Dirichlet Allocation over the tweets by the inhabitants of the counties. Table 1 displays the regression models that we tested.

Method Implemented	Methods part of leaderboard submission
Linear Regression with Ridge Regularization	No
Elastic Net for Linear Regression	No
Partial Least Square Regression	Yes
Regression Ensemble	Yes
SVM Gaussian Kernel	Yes

Table 1: Tested regression models

The methods that made most difference and hence included in the final leaderboard submission were Partial Least Square Regression, Regression Ensemble and Gaussian Kernel SVM (averaged using all data, and K-means clustered data). Each of these functions worked in tandem to produce a net model that performed better than each of its individual components. Methods like Elastic Net work well in theory, but due to the extreme computation time required to perform the *'lasso'* function on the full dataset, the only feasible way to apply this model was using the K-means centered data.

1.1 Training and Testing Error for the methods

Method Implemented	Training Error	Testing Error
Linear Regression with Ridge Regularization	0.0546	0.0844
Elastic Net for Linear Regression	0.0853	0.0887
Partial Least Square Regression	0.0584	0.0797
Regression Ensemble	0.0074	0.0807
Gaussian SVM	0.537	0.761

Table 1.1: Training and testing error of implemented methods

2 Data Transformation

Since the predictor variables SES and LDA topic frequencies are on a different scale, it becomes necessary for some data transformation that would treat the skewness in it. We use two techniques here:

1. Logarithmic transformation
2. Standardization

2.1 Logarithmic Transformation

The training data has a total of 2021 columns, where the first 21 columns are demographic features and the rest of the columns are LDA topic frequencies which are basically probabilities.We observe that the LDA topic predictor variables have a positive skewness in them and hence deem to have a logarithmic transformation to treat the skewness [1]. This is illustrated in Fig 1 and Fig 2 .Taking log also reduces the computation since most of the process involves multiplication and addition. So, these go smoothly if we introduce logarithmic transform to the training

data. Moreover, taking the log is safe since all these columns are probabilities which are always positive.

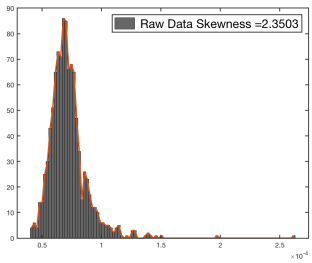

Figure 1: Raw data skewness.

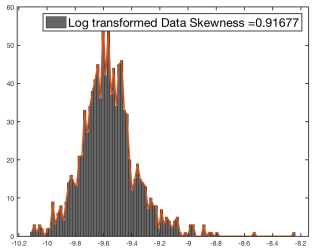

Figure 2: Log transformed data skewness

2.2 Standardization

For some of the regression models after logarithmic transformation of the LDA topic frequencies we also standardized the data by using the *zscore* function in Matlab. To treat all of the data equal i.e if we don't scale the features, a feature that has relatively little variation but is measured on a much larger scale than the other features may appear to have much greater variation relative to other features than it actually does. These result in good normal distributions to work with.

3 Description of the Models Implemented

3.1 Ridge Regularization

Linear regression was implemented using the matlab function: *fitrlinear*. The function: *fitrlinear* efficiently trains linear regression models with high-dimensional data. Given that we have features almost double the sample instances it became our first choice to test. As per the matlab documentation *fitrliear* uses stochastic gradient optimization techniques that helps reduce computation time.

To avoid overfitting, a common technique that is implemented is the Ridge regularization or the L-2 regularization where the weights undergo shrinkage. We tuned the regularization hyper parameters based on cross validation to get the lowest possible error. The hyper parameters for each of the response(or column in Y) column were determined separately. However this model was not the part of the leaderboard submission as we chose models superior to this.

We implement a 5-fold cross validation to determine the optimal regularization hyperparameter. We assign the vector of the regularization hyperparameter with values in orders of 10, and then zoom in the values which show minimum cross validation error to get much better estimate of the optimal hyperparameter. The error we chose here for convenience was mean squared error, implemented using matlab function *immse*. We here show only the plot for Y response column 2 in Fig 3

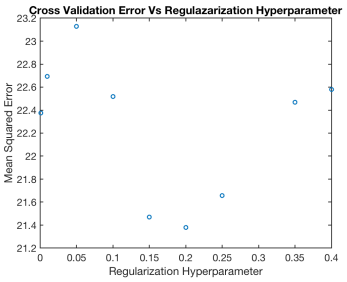

Figure 3: Cross validation

3.2 Elastic Net

Linear regression wherein the regularization term is both a L-1 norm and L-2 norm is Elastic Net [3]. This duality allows for the zeroing of redundant features in addition to shrinkage of weights. This is implemented using the matlab function: *Lasso*. However since the feature size is large, this model consumed considerably high time and had to be dropped in accordance with the competition guidelines of contraining the code run time to within 10 minutes.

3.3 Partial Least Square Regression (Discriminative)

PLS regression is an extension to Multi Linear Regression [2]. This also takes performs dimensionalty reduction. In PCR we regress over the principal components generated from the PCA. This doesn't take account into the responses or the dependent variable Y. But, this limitation is covered in PLS regression. This method performs a regression on Zs which are similar to principal components. The only difference between the both is that, the PCs are some linear combination of the feature elements, where as the Zs are the linear combination of the feature elements with weights or the coefficients capturing the variance in Y. This variance that is captured in Y is with respect to the corresponding feature element.

This method gave us good results when compared to normal Multi linear regression and hence formed a part of the leaderboard submission. The challenge

here was to determine the number of principal components that we intend to preserve for the regression. This number was determined individually for all the columns of labels and surprisingly the number was more or less the same : 16, suggesting a correlation among the features. The number of principal components to be included was obtained through cross validation.

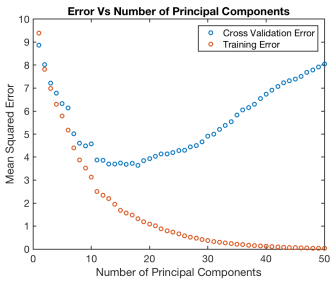

Figure 4: Cross validation

3.4 Regression Ensemble

The parameters tuned in this model are number of Learning cycles and Learning Rate. After various iterations and testing we found them to be 189 and 0.088 respectively. We had to limit the no.of Learning cycles to be below 200 as exceeding this would take more than 10 minutes to train. The predictions from all these models are taken and then averaged to obtain the final predictions that need to be considered for generating the error.

3.5 SVM Kernel Regression (Instance-Based)

We implemented five models in total with the training data not being equal on all the five. The final result of the prediction is the mean of the five models. Out of these five models, the first three models use Support Vector Machine method to train and predict. One of the SVM method is trained on the full training data using Gaussian Kernel. And the other SVM method is trained on the K means data, i.e we have obtained 30 Clusters in total for the training data and we took the mean of the feature columns that correspond to their respective clusters. This way we had 30 new features in total which we used as new training data to the second SVM model. The prediction step followed in a similar manner treating the testing data the same as we treated the training data in each of the methods. Coming to the box constraint and Kernel scale values of the Gaussian kernels, we have used a hyper parameter tuning function that gives us the output of a box constraint and a Kernel scale value for each of the nine predictions which gives us a good accuracy. So, in total we had two sets of box constraints and kernel scale values for the two SVM models. The third SVM model took in the first 21 columns of the training data, i.e We trained on the demographic features of the training data. This model too was accompanied by Box constraints and Kernel scales of the second SVM model (trained on K-means clustered data). Later, all the three predictions are averaged to obtain a final prediction vector from the SVM section. The training and cross-validation errors for each individual model in addition to the averaged model are shown in the table below.

	Training Error	CV Error
Z Score Data	0.0274	0.0814
K Means Data	0.0655	0.0829
Regional Features	0.0851	0.0856
Averaged	0.0537	0.0761

The choice of this model was largely an offshoot of looking at ways to get improved performance from *fitrlinear*. Performance data for *fitrsvm* using a gaussian kernel is shown in the table below for both the full, normalized (Z-score) data and the k-means clustered data set. It was found through trial in error that combining these two models yielded far better results than each of them alone. It can be observed that the full-data set drastically over fit the data, with a training error of 0.0274 with a test error of 0.0814. We believe that by merging the predictions of this model with the reduced data set using a simple average, it helped to substantially reduce the over fitting, yielding a final cross-validation error of 0.0758. The choice of parameters for the Gaussian Kernel SVM was made using MATLAB's built-in 'HyperparameterOptimization' flag in the *fitrsvm* function. The parameters optimized were the Box-constraint and Kernel-Scale. A representative plot for this is shown below, describing the optimization for the first prediction column using the full, normalized dataset. This optimization was independently performed on both the overall dataset and the K-means clustered data. All additional hyperparameter optimization figures are available in the SI.

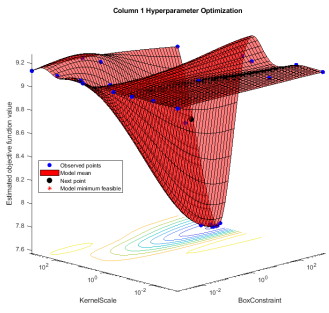

Figure 5: Hyperparameter optimization for for prediction of first column of labels using Z-score dataset.

The table below summarizes the results of hyperparameter optimization for the Z-score data.

Column	Box-Constraint	Kernel-Scale
1	996.7	23.7
2	134.8	34.6
3	1.711	11.96
4	19.88	18.23
5	25.9	12.04
6	253.12	84.66
7	813.2	70.97
8	6.09	17
9	558.8	139.5

There were also separate hyperparameters optimized for the K-means dataset and the regional data. The values obtained from hyperparameter optimization for these datasets are shown below.

Column	Box-Constraint	Kernel-Scale
1	628.5	77.5
2	467.21	132.9
3	11.724	91.5
4	695.6	98.6
5	41.176	107.5
6	14.22	71.14
7	53.4	79.5
8	12.14	109.1
9	12.5	83.9

3.6 Leaderboard Submission (also the novel method

3.6.1 Clean the Training Data

The first step we took is to clean the Training data. This is same as the Data Transformation section above.

3.6.2 K-Means Clustering (Generative)

In order to simplify the dataset and reduce overfitting, the twitter frequency data normalized using the 'zscore' function and the 'kmeans' function was used to assign each of the proposed features to a certain number of clusters. Feature columns of identical cluster number were averaged and this data was recombined with the regional data to reconstruct a reduced data set.It was found, using *fitrlinear* as a method of probing goodness of fit, that low values of K yielded the best cross-validation errors. Using default arguments for *fitrlinear*, we settled on a number of clusters K = 61, though any choice between 30-60 yielded good results. We speculate that the large spike as K approaches 100 is due to *fitrlinear* changing default parameters as the data crosses a threshold to being considered "high-dimensional" from the function, however we were unable to pin down the precise origin to control for it in our optimization of K. Below are the two figures showing the plot between Cross - Validation error vs K and Final Error vs K, where K is the no.of clusters.

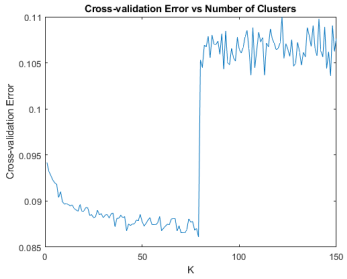

Figure 6: Cross-Validation Error vs K for K between 1 and 150 clusters, showing a sharp minima at low K.

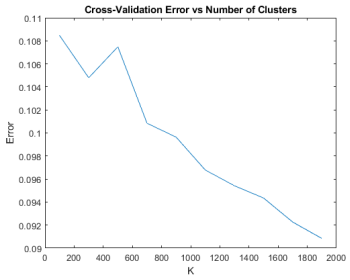

Figure 7: Cross-validation error vs. K for large K values, showing a general decrease as K increases.

3.6.3 Novel Method

Individually, our methods provided modest performance. However, through the process of trial and error, we found that by combining models, we could obtain a lower cross-validation error at the expense of training error. The only that needed to be addressed is how exactly to combine the models. The first approach we took was to predict labels for each model and output the labels, then we would combine the labels in a final model using a weight vector optimized through cross-validation error. $\hat{y_f} = w.*[\hat{y}_{pls}\hat{y}_{svm}\hat{y}_{ens}]$. These weights would be found by minimizing the cross-validation error. However, we quickly found that even with this approach, we were reducing our cross-validation error for any uneven weights. Thus, we eventually settled on a simple-average to combine the models. We believe that this works due to the fact that a set of models that separately overfit are likely to overfit in different ways. Therefore, by combining them in a simple average, the net overfitting is reduced, which manifests as an increase in training error and a reduction in cross-validation error. The choice of models that went into this was largely accomplished through trial and error. We found that similar models tended to show no improvement when averaged (such as fitrlinear and fitrsvm with a gaussian kernel), which different models combined well. Thus, the final models we settled on were PLS, SVM with a Gaussian Kernel, and an LSBoost ensemble fit. In total, our combined model

yielded a testing error of 0.0722 when submitted to the leaderboard with a training error of 0.0546.

4 Interpretation

4.1 What does our model tell about public health in US

Our model predicts the dependence of US health outcomes on tweet data and socio-economic factors. We illustrate our predictions on a test data of 319 counties as represented by the Fig 8 which reflects the actual data compared to our prediction Fig 9 for the health outcome of physical activity. Interesting interpretations are captured in the following subsections.

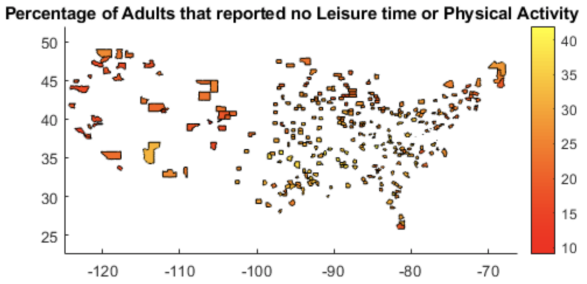

Figure 8: Actual data for the 319 test counties

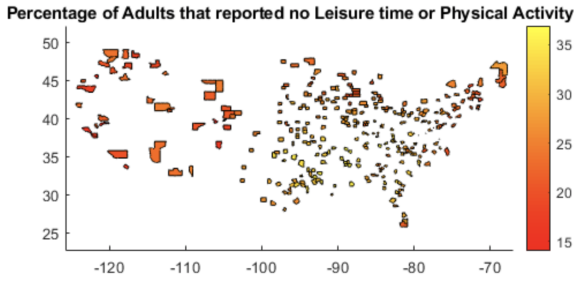

Figure 9: Predicted data for the 319 test counties

4.2 Correlation among the y-variables

We compute Pearson correlation coefficient between different US health outcomes. Now Pearson correlation might not capture non-linear relationship but we still explore the linear correlations. As an example: Fig 10 suggests higher correlation of obesity with other US health outcomes. However what is more interesting to see is the features that are most predictive of the county health and this is discussed in the next section.

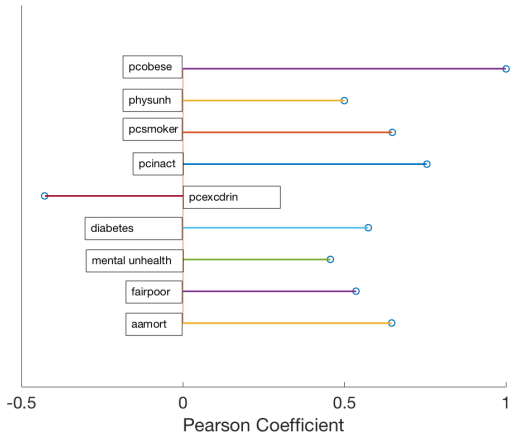

Figure 10: Pearson Coefficient of Percentage Obesity outcome with other US health outcomes of the counties

4.3 Features most predictive of county health

Ideally feature engineering is a broad topic and would involve hypothesis testing and concepts like p-value, however here we follow a simple approach of variable interpretation based on Pearson correlation between the individual US health outcomes and socio-economic features. We observe a similarity in the trend of all the health outcomes. There are certain features which consistently show a greater Pearson correlation with all the US health outcomes and these are college education, computer access, ses log hhinc, unemployment and percentage of black population, we consider them to be more predictive of the US health outcomes. An US health outcome of obesity's correlation is illustrated by Fig 11. Predictive word topics do make sense because while other features are based on demography and data collected by the respective authorities, a healthy living could be more closely understood by the mindset of the people which is very much reflected in the things they tweet.

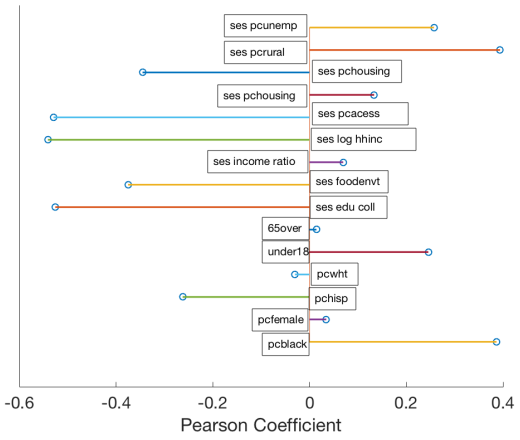

Figure 11: Pearson Coefficient of SES features with Percentage Obesity outcome of the counties

5 Conclusion

Throughout this project, we investigated several methods of modeling a variety of health outcomes using combination of information about the county (demographics, number of metro areas, etc) and a list of LDA twitter topic frequencies. Throughout the process, it quickly became apparent that due to the large number of features, the primary concern was avoiding overfitting. To address this, we found that by averaging several overfit models, we could mitigate the influence of over-fitting and recover a collective model with better performance than its individual parts. For this model, we combined a Gaussian kernel SVM regression, an LSBoost ensemble, and a PLS Regression. Individually, each of these models had an error of less than 0.08, with the combined Gaussian SVM fits yielding a cross-validation error of 0.0761, the PLS regression yielding an error of 0.0797, and the LSBoost ensemble having an error of 0.0807. While averaging our models allowed us to use the majority of our methods, we had less success with elastic net and ridge regression. For models that did not take an excessive amount of time to evaluate, cross-validation errors for these methods were restricted to the vicinity of 0.085 or greater. While these methods did not perform poorly, the simply did not perform as well as the ones we used in the final model. Possible reasons for this include extreme overfitting, in the case of a linear regression model, or excessive influence of penalization, in the case of elastic net regression. Some approaches that may improve the performance of these methods would include a better choice of hyperparameters, a bit more focus on dimensionality reduction, and possibly the employement of a two-step regression procedure where elastic net is used to zero features and a second, linear regression is used to obtain parameters that are less overfit.

References

[1] FENG Changyong, WANG Hongyue, LU Naiji, CHEN Tian, HE Hua, LU Ying, et al. Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, 26(2):105, 2014.

[2] Sijmen De Jong. Simpls: an alternative approach to partial least squares regression. *Chemometrics and intelligent laboratory systems*, 18(3):251–263, 1993.

[3] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

6 Supplementary Information

6.1 Gaussian SVM Hyperparameter Optimization Plots for ZScore Data

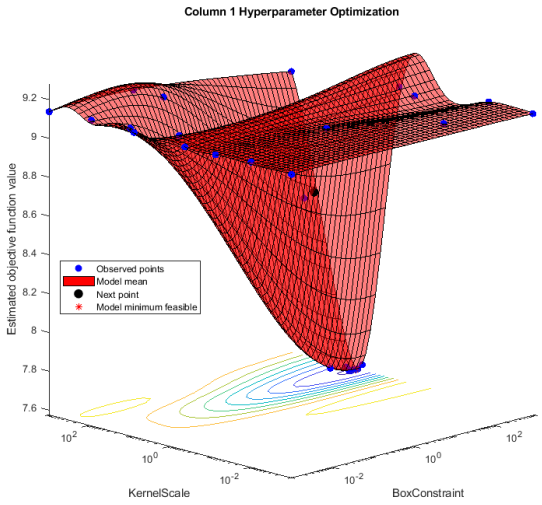

Figure 12: Box-constraint, Kernel Scale - Column 1

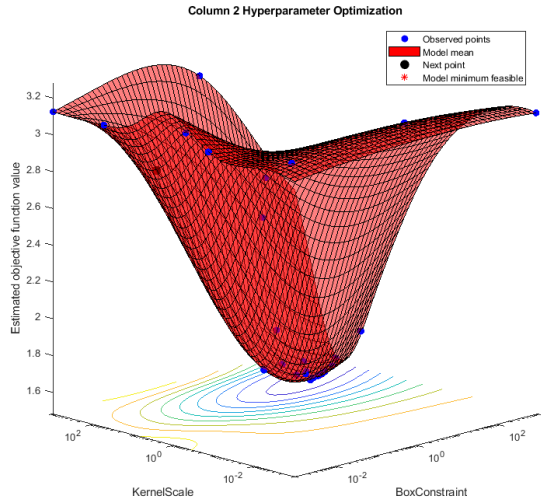

Figure 13: Box-constraint, Kernel Scale - Column 2

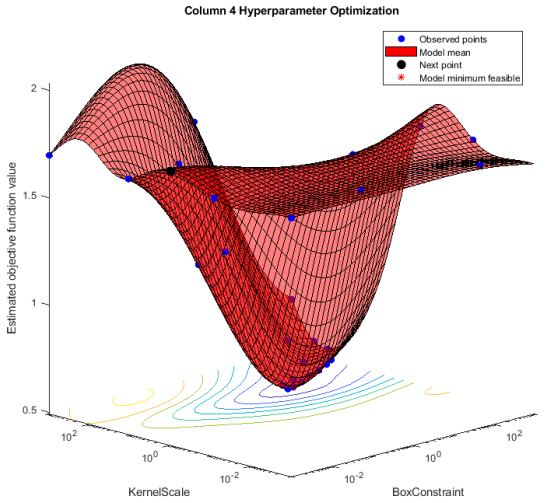

Figure 15: Box-constraint, Kernel Scale - Column 4

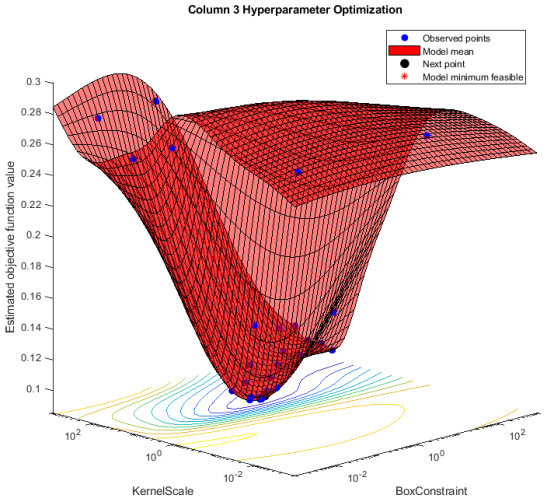

Figure 14: Box-constraint, Kernel Scale - Column 3

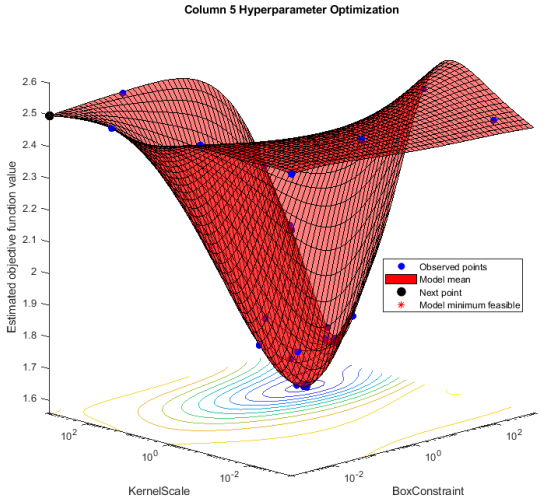

Figure 16: Box-constraint, Kernel Scale - Column 5

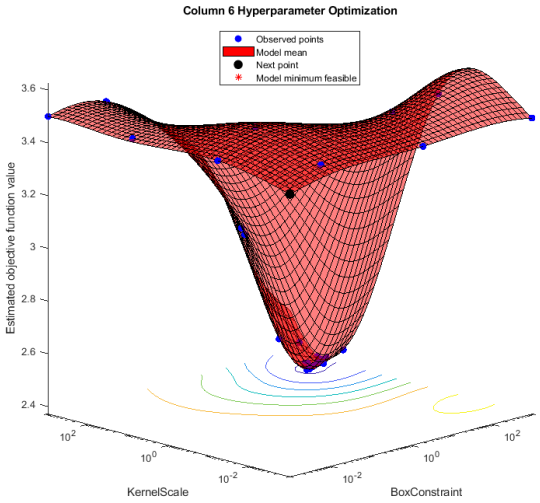

Figure 17: Box-constraint, Kernel Scale - Column 6

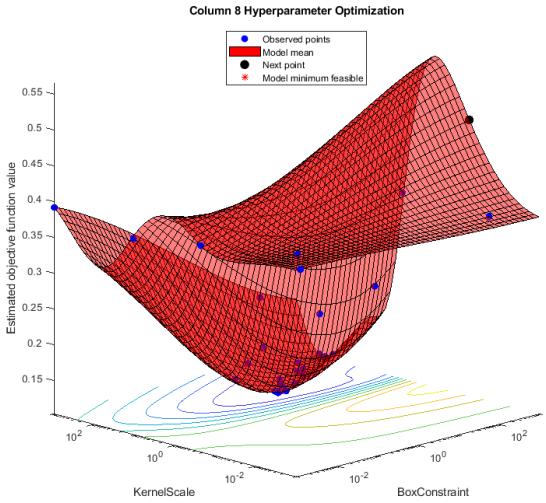

Figure 19: Box-constraint, Kernel Scale - Column 8

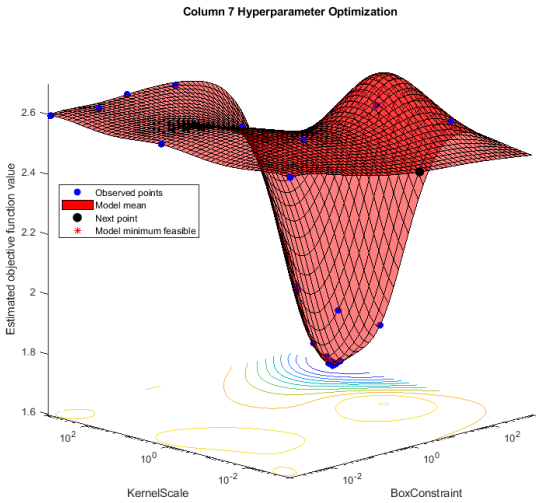

Figure 18: Box-constraint, Kernel Scale - Column 7

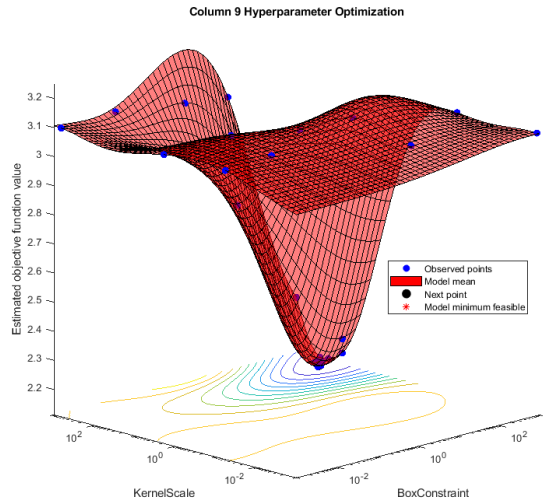

Figure 20: Box-constraint, Kernel Scale - Column 9

6.2 Gaussian SVM Hyperparameter Optimization Plots for KMeans Clustered Data

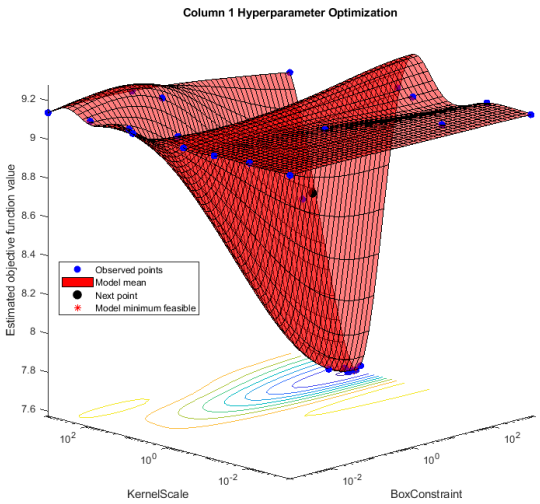

Figure 21: Box-constraint, Kernel Scale - Column 1

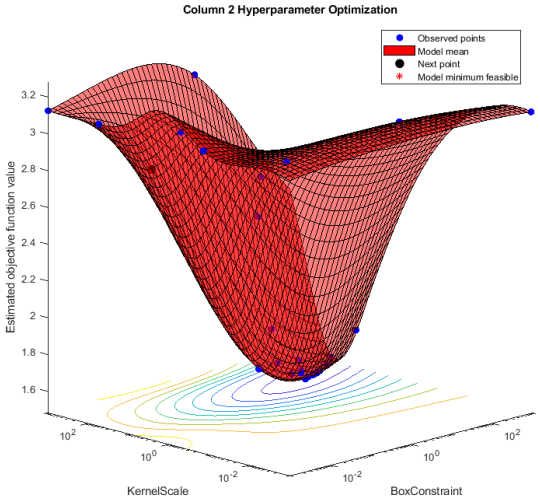

Figure 22: Box-constraint, Kernel Scale - Column 2

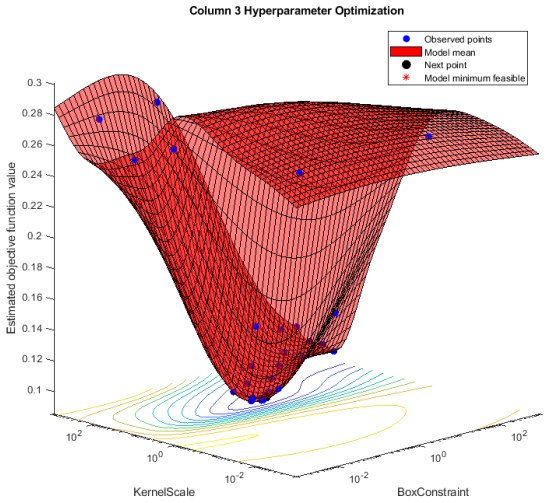

Figure 23: Box-constraint, Kernel Scale - Column 3

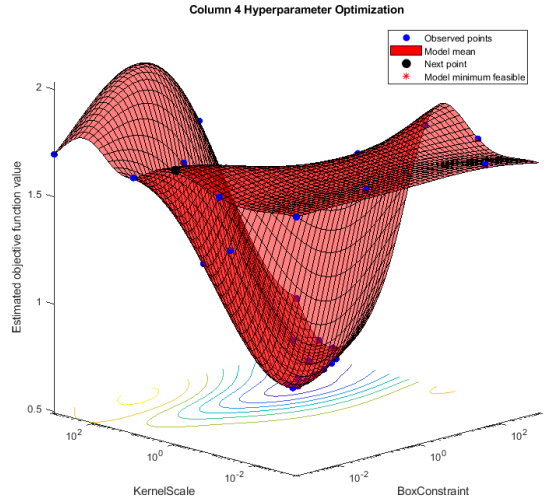

Figure 24: Box-constraint, Kernel Scale - Column 4

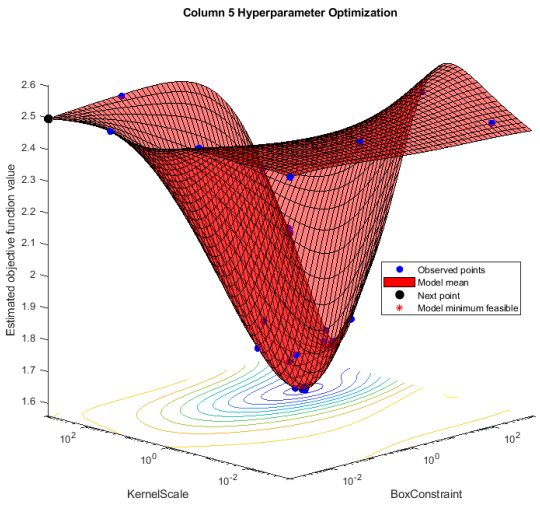

Figure 25: Box-constraint, Kernel Scale - Column 5

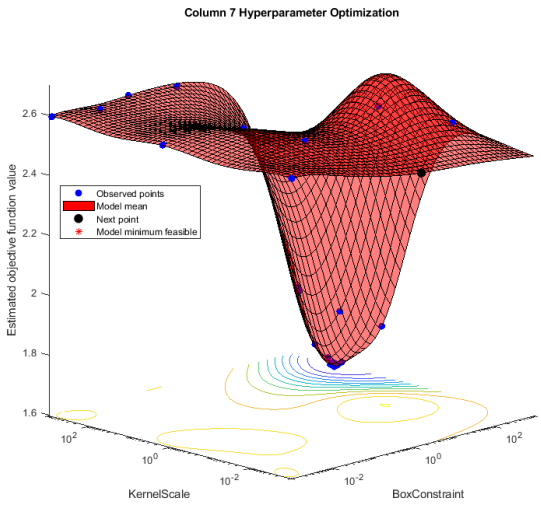

Figure 27: Box-constraint, Kernel Scale - Column 7

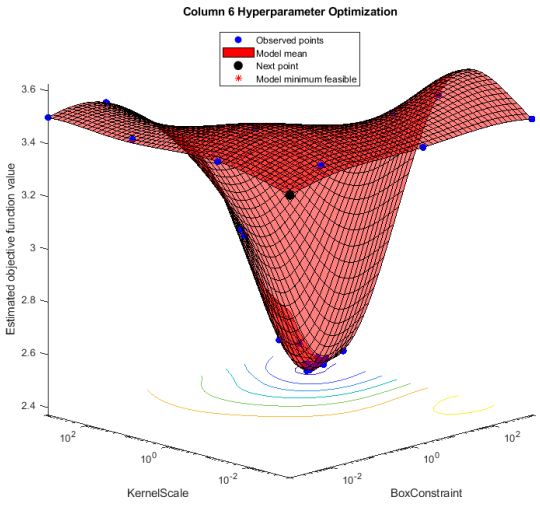

Figure 26: Box-constraint, Kernel Scale - Column 6

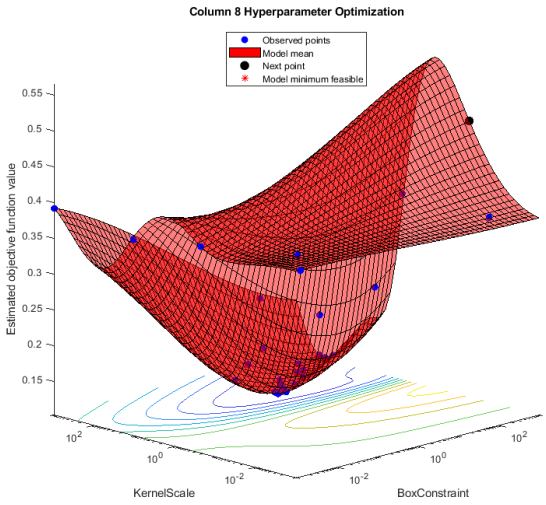

Figure 28: Box-constraint, Kernel Scale - Column 8

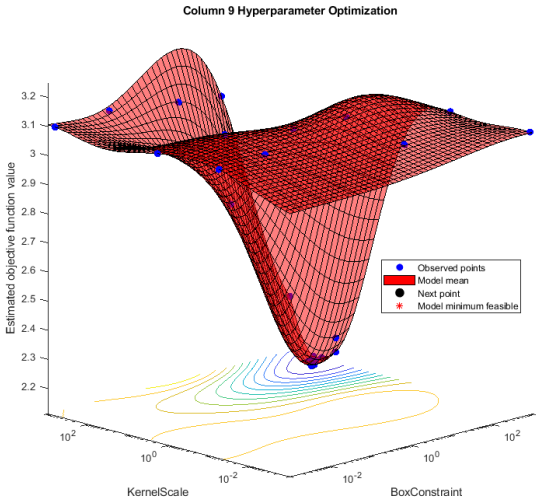

Figure 29: Box-constraint, Kernel Scale - Column 9