# Predictive Model on Protein Folding Accuracy

Weixi Zhuo
(STAT331 Final Project)
*20722735*

(Dated: 12 December 2020)

## I. SUMMARY

The objective of this project is to use multiple linear regression (MLR) based model to predict how close that computer-generated protein structure is to a known benchmark structure. This report outlines the process of obtaining a MLR model with optimal predictive power by considering model selection processes on the original feature space and extended feature space with kernel Principal Component Analysis (KPCA). After performing 2 cross-validation schemes, we obtain an optimal model from the forward-backward model search algorithm with respect to the root-mean-squared-prediction-error (RMSPE). The report is finished with some additional findings that the selected model reveals.

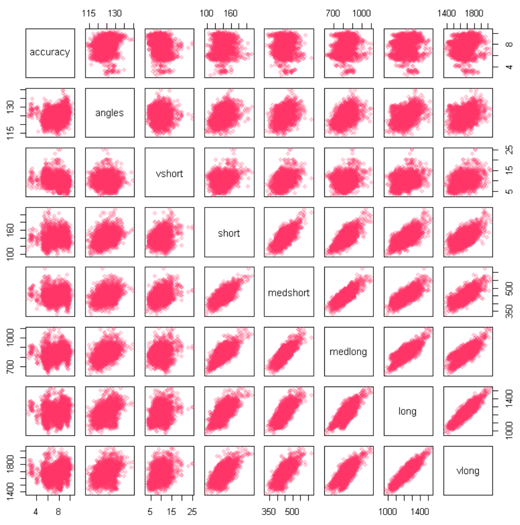
## II. EXPLORATORY DATA ANALYSIS

We observe that, besides the `angles` predictor, which is a score based on the configuration of angles in the structure, other predictors are of the form `atomtype1_atomtype2_lengthtype`. We reversely count the occurrence of each atom type and length type in a single data row. This provides us 24 aggregated data columns. Even though it is not recommended to use them as additional features since they have exact multicolinearities with the given features by construction, they provide us the counts of different types of atoms or distances (see table: III for detailed summary). We perform correlation analysis on these aggregated features. In particular, from figure: 1a, we note that some of the longer distance types have clear positive correlations. While in figure: 1b, some atom types have high positive correlations, such as bbN,bbCA,bbC,bbO. We also noticed that the response variable, `accuracy`, has a left-skewed distribution while most other features have right-skewed or close-to-symmetric distributions (see figure:9 in Appendix:VI A). In particular, based on the multiple linear regression model assumption, we have $Y_i \sim N(\mu_i, \sigma^2)$. Thus, such right-skewed response might require transformation, such as Box-Cox transformation. Last but not least, the angle score seems to be a drastically different feature since it is continuous while the others are discrete counts.

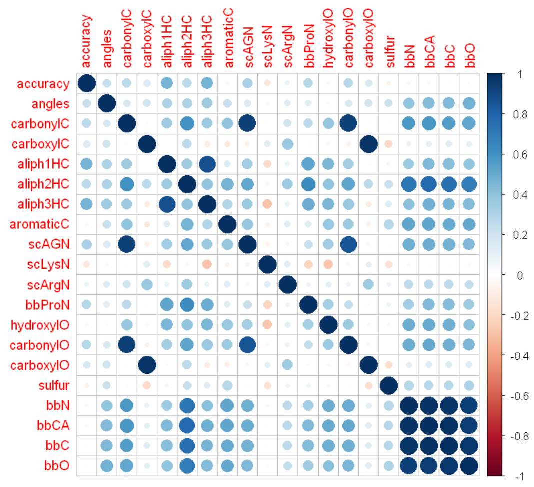
## III. METHODS

We use the basic multiple linear regression (MLR) model as our benchmark model. Additionally, we have two different approaches to construct a set of candidate models.

FIG. 1: Correlation Graphical Summary

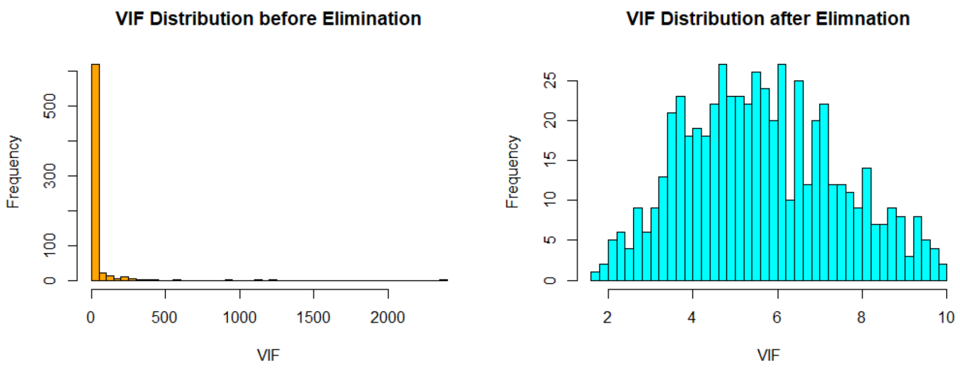


(a) Pair Plots of Length Types



(b) Atom Types Correlation Heat Map

FIG. 2: VIF distributions before and after the iterative elimination



## A. MLR on Original Feature Space

We first perform iterative elimination of features with high variance inflation factor (VIF). This is to reduce potential multicolinearity. The threshold we take for the feature VIF in iterative elimination process is 10, which is a common rule of thumb. This reduces the feature space from dimension of 685 to 568. The dimension of the feature space, even after this reduction, is still large, the histograms (see figure:2) shows the VIFs before and after this reduction. We consider both deterministic and stochastic model selection processes. The deterministic processes include forward selection (**Forward**) and forward-backward (**FB**) selection while the stochastic process is the iterative conditional minimization (**ICM**). For model selection criteria, we consider the $L0$-penalized likelihood defined by

$$q\lambda - 2\ln L(\hat{\theta}) \tag{L0}$$

where $\lambda = 0.5 \times \log(n)$ and $n$ is the number of data rows. To pick the hyperparameter, $\lambda$, we perform a linear parameter search for $\lambda \in [0.1, 1.0]$. In particular, when $\lambda = 1.0$, the criteria is exactly the Bayesian Information Criteria (BIC). Then, pick the $\lambda$ that result in the optimal set of RMSPE in the cross-validation schemes mentioned in III C. Detailed $\lambda$ tuning process is given by Algorithm:2 in VI B 1.

## B. MLR on Extended Feature Space

To consider potential linear and non-linear interactions of the original features, we propose to consider kernel principal component analysis (KPCA) with polynomial kernels. The theoretical details and principal component selections is in Appendix:VI C. This provides a set of extracted features, selected principal components derived directly on the original feature space, for us to fit MLR. We consider the degree hyperparameter for the polynomial kernel to be $1, 2$, or $3$. This provides us another set of candidate models. For our kernel PCA method, we directly perform cross-validation schemes to decide the degree hyperparameter instead of obtaining candidate models on the whole training set as in III A.

## C. Final Model Selection

After obtaining our candidate models using the two approaches mentioned, we perform two types of cross-validation schemes. First one is to consider 200-Fold training/validation set split and compare the RMSPE distribution on the validation sets for all candidate models. This scheme uses most of the given data for training, thus, can give us an idea how well our model can be trained using this given dataset. The second scheme is to consider 2-Fold training/validation set split and resample 100 times. This corresponds to the fact that our training/testing ratio is 1-to-1 eventually. We want to test how well the candidate models can accommodate such size difference. (see Appendix:VI D for details) One important thing to know for the candidates on the original feature space is that they are fitted on the whole dataset, which will incur significant bias if the cross-validation process is not proper. To counter this, we have adjusted cross-validation algorithm for them shown in algorithm:1 . After selecting the final candidate model, we shall also compare it with the version that performs Box-Cox transformation on the response variable in terms of the cross-validated RMSPE. In particular, we have seen cases where Box-Cox transformation is not invertible on the predicted values. Hence, we propose an extended Box-Cox transformation with defined inverse (**BCE**) shown in VI C 4. Thus, this process yields our final predictive MLR model.

## IV. RESULTS

After performing both deterministic and stochastic model selectionS on the whole training set, we obtain 3 candidate models with the following performance summary displayed in table:I. We observe that the forward selection and forward-backward selection evaluated many more models than ICM's case. This relates to the fact that our **Forward** and **FB** models have many more predictors than the **ICM** candidate. Even though **ICM** has considerably fewer predictors, its Akaike Information Criteria (AIC), BIC, and $L0-$penalty

| | **Forward** | **FB** | **ICM** |
|---|---|---|---|
| Number of Fitted Models | 78975 | 82783 | 5690 |
| Number of Predictors | 162 | 158 | 27 |
| AIC | 2803.00 | 2796.00 | 3961.85 |
| BIC | 3717.06 | 3687.77 | 4123.49 |
| $L0-$Penalty | 3096.03 | 3081.89 | 4013.67 |
| $R^2_{adj}$ | 89.50% | 89.52% | 79.67% |

TABLE I: Performance Summary of Candidate Models on Original Feature Space

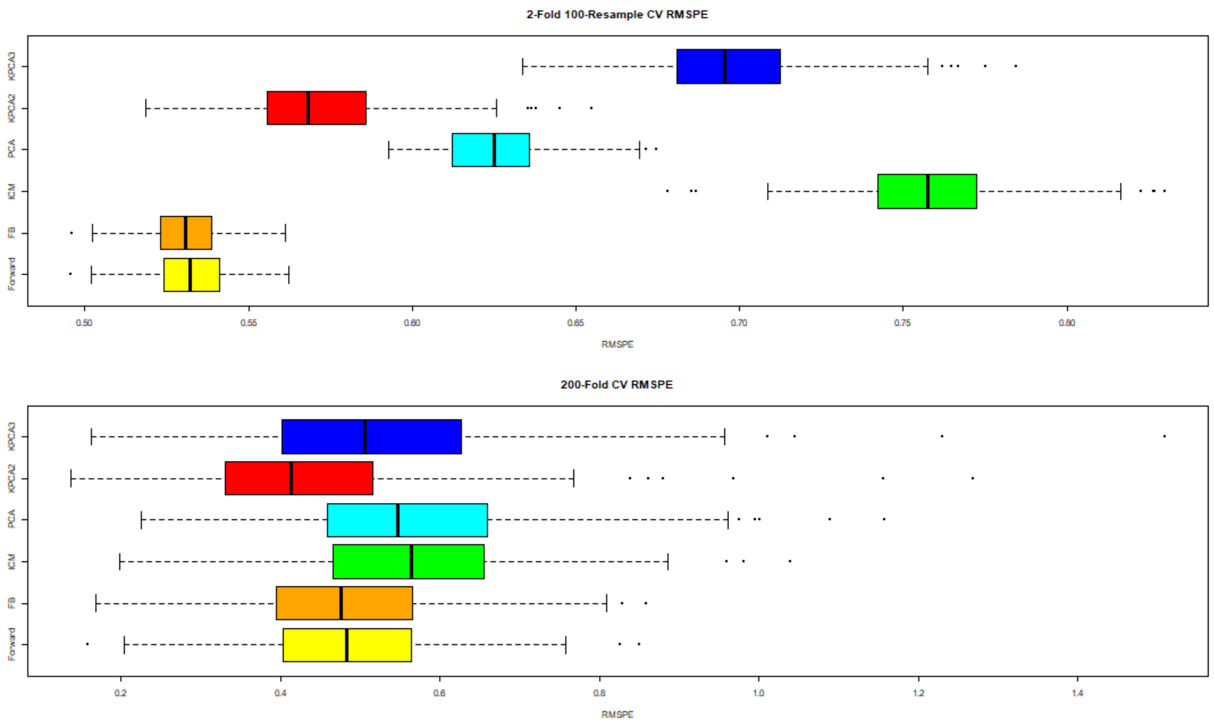
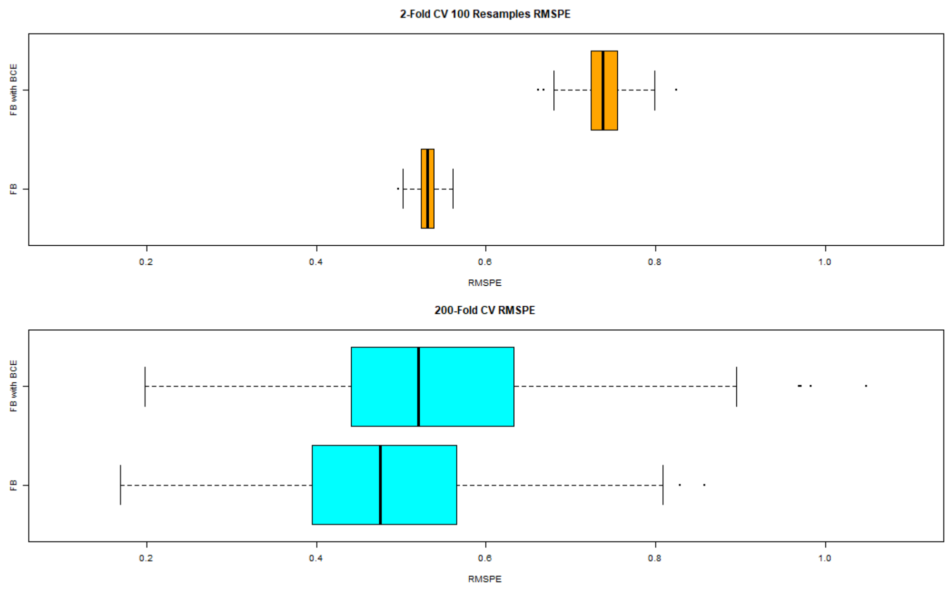


FIG. 3: RMSPE Boxplots for Candidates Models within 2 CV Schemes

defined by L0 display inferiority compared to the **Forward** and **FB** models. We need to be self-conscious about the fact that these model selection processes were conducted on the whole training set. This does incur bias on our later cross-validation results and makes the adjusted $R^2$ not a great metric to select our final model among them. Overall, our **FB** model seems to have an edge across multiple metrics among these. Nonetheless, the final selection depends on the cross-validation result with RMSPE as the predictive performance metric. To counter the bias mentioned, we have a cross-validation adjustment with details in Appendix:VI D.

As mentioned in III C, we have two cross-validation schemes and the results are displayed in figure:3. We note that in the 200-fold case, the kernel PCA model with degree 2 polynomial kernel (**KPCA2**) and our **FB** model tend to have the lower ranges of RMSPE. For the detail of computing RMSPE for the kernel PCA, see Appendix:VI C 3. However, for the 2-fold-100-

FIG. 4: RMSPE Comparison Before and After BCE



resample case, our **Forward** and **FB** models seem to outperform the rest of the models. This gives us some degree of certainty that **KPCA2** is the best model derived from the kernel PCA method while **FB** model has the best predictive power among models on the original feature spaces. **KPCA2** has one advantage over **FB** model, which is the fact that it is trained within the cross-validation process instead of on the whole training set. However, we note that the **KPCA2** model's RMSPE has a much larger range than the **FB** model's RMSPE range. Similar case goes for their interquartile ranges (IQR). It would be concerning for the variance of the RMSPE estimate on the testing set if we select **KPCA2** as our final model. Moreover, one clear advantage of the **FB** is interpretability of the estimated coefficients over the **KPCA2** where we do not have clear interpretation of the principal components with high-degree polynomial kernel. Thus, our final candidate model is **FB**.

Finally, we test whether **BCE** can improve the RMSPE. As we can see in figure:4, the anticipated Box-Cox transform with **BCE** does not improve the performance of our **FB** model across 2 CV schemes. Hence, we decide to keep our proposed **FB** model. One of the immediate surprises is that the special feature, angle score, mentioned in II, is not included in the final **FB** model. We conduct a manual model change by adding the angle score to the **FB** model to see if the predict performance changes. As we can see in figure:5, our manual adjustment with the addition of angle score do not show improvement of predictive power across 2 CV schemes. Hence, we select **FB** as our final model and the detailed model formula is given in Appendix:VI E and the model summary is given in table:IV.

Since our final **FB** model has 158 predictors, we give contextual interpretations of the predictor coefficients for the largest 5 estimates in magnitude besides $\beta_0$ in table:II. From the detailed model summary in IV, we note that all the p-values for individual predictor's t-test is extremely small. This provides reassurance for the inclusion of these predictors. Our estimated RMSPE on the actual testing set is 0.5307 which is the mean RMSPE of **FB** model on 2-fold-100-resample cross-validation process. We believe this CV method is considerably more realistic. Also, if we consider the naive Gaussian assumption of the

FIG. 5: RMSPE Comparison Before and After Adding Angle Score

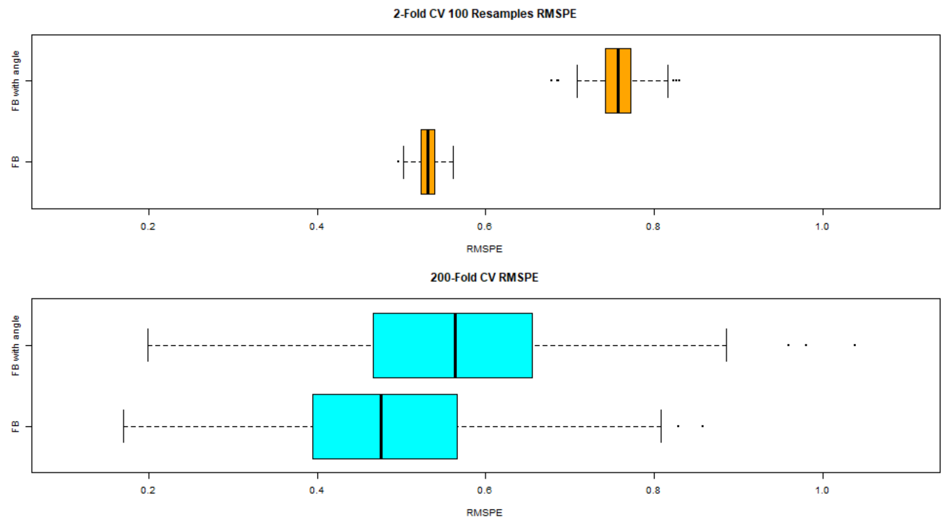


TABLE II: Top-5 Predictors Interpretation with Largest Estimate Magnitudes

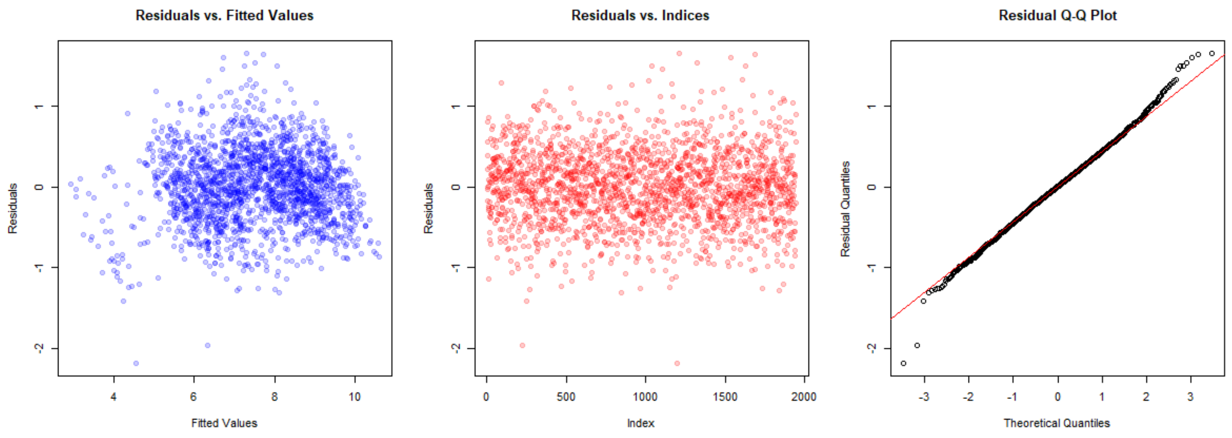
| **Feature Name** | **Estimate** | **Interpretation** |
|---|---|---|
| scArgN_bbO_medlong | 0.3646 | The accuracy measure is expected to increase by 0.3646 for every increment of medium-long pair of scArgN and bbO atoms |
| bbCA_bbO_vshort | -0.3532 | The accuracy measure is expected to decrease by 0.3532 for every increment of very-short pair of bbCA and bbO atoms |
| aliph1HC_bbProN_medlong | 0.3191 | The accuracy measure is expected to increase by 0.3191 for every increment of medium-long pair of aliphlHC and bbProN atoms |
| aliph1HC_scArgN_long | 0.2877 | The accuracy measure is expected to increase by 0.2877 for every increment of long pair of aliphlHC and scArgN atoms |
| scArgN_bbO_long | -0.2605 | The accuracy measure is expected to decrease by 0.2605 for every increment of long pair of scArgN and bbO atoms |

RMSPE distribution, we have the following 95% prediction interval for the testing RMSPE:

$$\overline{\mathsf{RMSPE}_{cv}} \pm a\mathsf{SD}(\mathsf{RMSPE}_{cv})\sqrt{1+\frac{1}{2\times 100}} = [0.5065, 0.5549]$$

where $a$ is the 97.5% percentile of a $t(199)$ distribution.

FIG. 6: Residual Plots Summary (without BCE)



## A. Model Diagnostics

For model diagnostics, even though the primary objective of the project is to provide a MLR model with optimal predictive power, it is worthwhile to check whether our final model can do well throughout this process so that we can reflect on our methodology. We consider the residual plots shown in figure:6. We note that the scatter plot between residuals and fitted values displays a clear cluster. This implies that we might want to consider the Box-Cox transformation. In fact, we have the transformed version shown in figure: 12 in Appendix:VI E where indeed the scatter plot looks more spreaded out around 0. As for the scatter plot between residuals and indices in figure:6, it displays quite evenly spreaded out points around 0. This means that the residuals of our **FB** model are independent and not subject to autocorrelation. From the residual Q-Q plot in figure:6, we can see that most of the residuals are on a straight line except for some points near two tails. This means that our **FB** model's residual respect the normality assumption of MLR. We also conduct an analysis to look out for certain outlier data rows. In particular, we consider the studentized residual, leverage, and Cook's distance. By figure:7, we can see that we have only a few observations in the training set that have studentized residuals higher than 3. This implies that overall we do not have a lot of extreme values of accuracy responses. Also, not a lot of observations have leverages higher than 2 times average leverage. This means the feature data restricted to our **FB** model are not very extreme. Finally, we do not have observations that have extremely large Cook's distance. This means our training set do not have large influence observation subject to our **FB** model. This provides us with further confidence that our **FB** model is not derived on an outlier dataset.

## V. DISCUSSION

### A. Model Generalization

Based on table:I's AIC and BIC metrics, we believe the $L0$-penalty that we defined in L0 is adequate enough so that **FB** model does not bear the issue of overfitting to this training

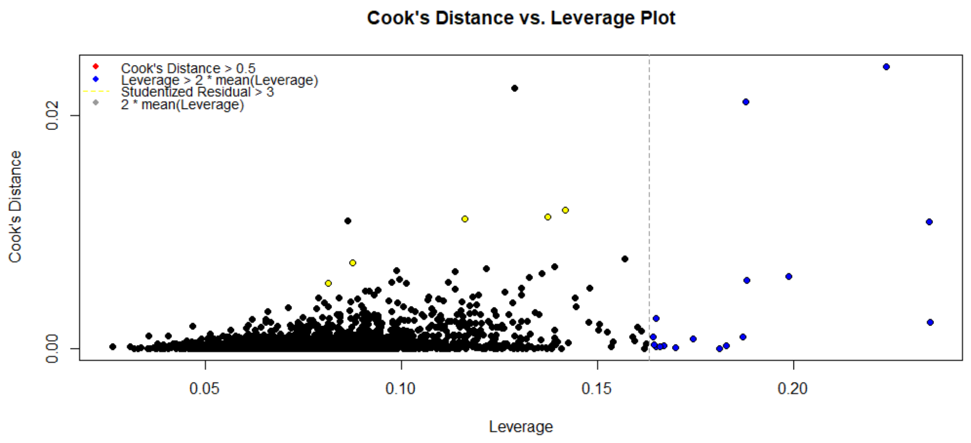


FIG. 7: Cook's Distance vs. Leverage Plot

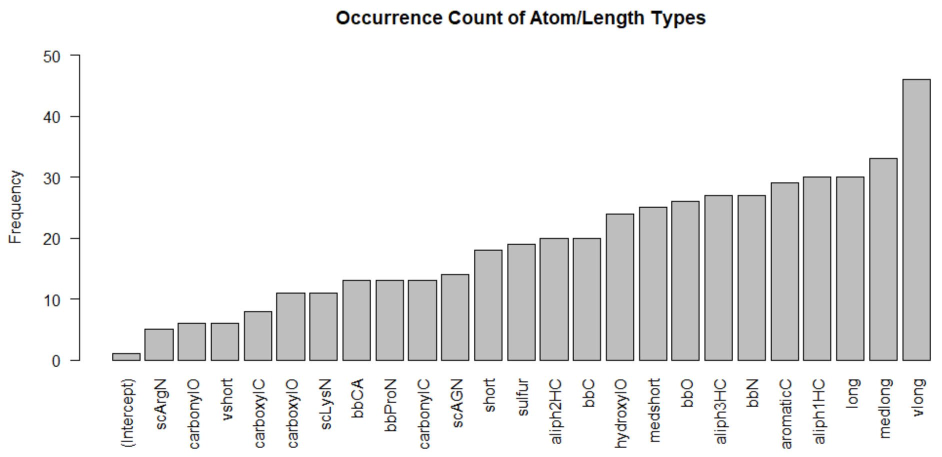


FIG. 8: Count of Atom/Length Types Occurrences

set. Especailly, in figure:10, the process of selecting our desired $\lambda$ in fact is testing whether the searched model has the tendency to overfit by comparing the RMSPE results.

## B. Other Findings

We count the atom/length types occurrence as part of the **FB** model predictors shown in figure:8. We note that the count for very-long pairs of atoms is the most. This might imply that the accuracy of the computer simulated protein folding structure are more prone to be impacted by very-long pairs of atoms, either positively or negatively, which cannot be observed from this chart. Meanwhile, the atom type with the most count is `aliph1HC` while it is not that frequent to see this atom type in observations by table:III. This might lead to further research on this atom type's effect on the accuracy measure.

## VI. APPENDIX

### A. Exploratory Data Analysis

TABLE III: Aggregated Feature Summary

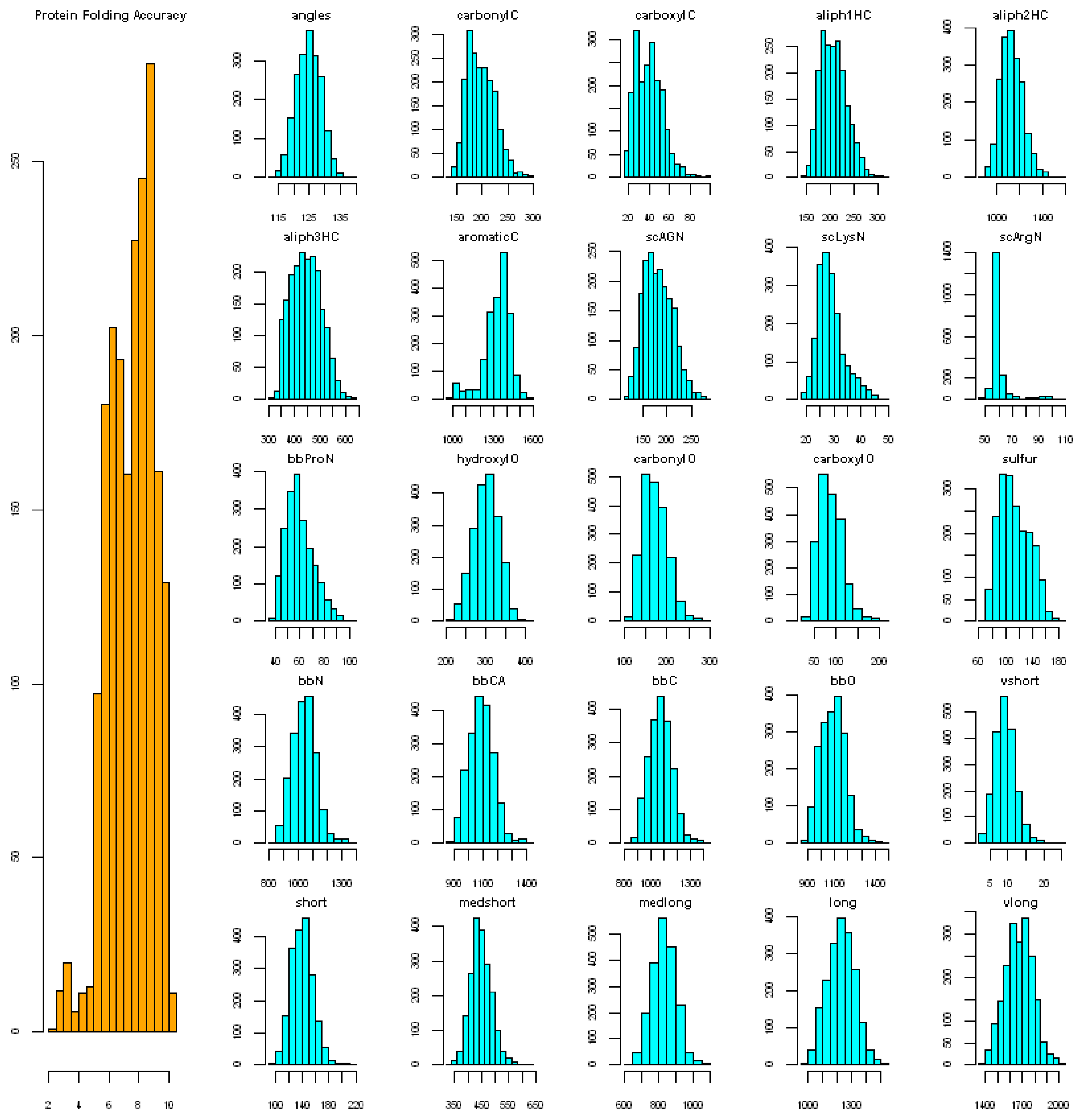
| | accuracy | angles | carbonylC | carboxylC | aliph1HC | aliph2HC | aliph3HC |
|---|---|---|---|---|---|---|---|
| Min | 2.37 | 113.80 | 139.00 | 15.00 | 146.00 | 875.00 | 314.00 |
| 1st Qu | 6.37 | 122.00 | 177.00 | 29.00 | 186.00 | 1066.00 | 400.00 |
| Median | 7.70 | 124.90 | 195.00 | 40.00 | 205.00 | 1126.00 | 444.00 |
| Mean | 7.49 | 124.80 | 197.80 | 40.39 | 207.40 | 1135.00 | 446.10 |
| 3rd Qu | 8.68 | 127.70 | 216.00 | 49.00 | 225.00 | 1199.00 | 489.00 |
| Max | 10.48 | 139.60 | 300.00 | 98.00 | 314.00 | 1559.00 | 631.00 |

| | aromaticC | scAGN | scLysN | scArgN | bbProN | hydroxylO | carbonylO |
|---|---|---|---|---|---|---|---|
| Min | 979.00 | 118.00 | 18.00 | 47.00 | 38.00 | 213.00 | 110.00 |
| 1st Qu | 1275.00 | 158.00 | 26.00 | 57.00 | 52.00 | 280.00 | 151.00 |
| Median | 1348.00 | 178.00 | 28.00 | 58.00 | 59.00 | 302.00 | 169.00 |
| Mean | 1329.00 | 181.30 | 29.33 | 60.57 | 60.34 | 301.70 | 172.30 |
| 3rd Qu | 1395.00 | 202.00 | 32.00 | 60.00 | 67.00 | 324.00 | 191.00 |
| Max | 1591.00 | 282.00 | 49.00 | 106.00 | 102.00 | 411.00 | 288.00 |

| | carboxylO | sulfur | bbN | bbCA | bbC | bbO | vshort |
|---|---|---|---|---|---|---|---|
| Min | 26.00 | 61.00 | 838.00 | 871.00 | 848.00 | 864.00 | 3.00 |
| 1st Qu | 65.00 | 96.00 | 986.00 | 1030.00 | 1012.00 | 1022.00 | 8.00 |
| Median | 85.00 | 110.00 | 1043.00 | 1090.00 | 1071.00 | 1089.00 | 10.00 |
| Mean | 87.52 | 113.70 | 1044.00 | 1090.00 | 1074.00 | 1090.00 | 9.84 |
| 3rd Qu | 105.00 | 131.00 | 1095.00 | 1147.00 | 1131.00 | 1152.00 | 12.00 |
| Max | 218.00 | 184.00 | 1374.00 | 1416.00 | 1416.00 | 1471.00 | 25.00 |

| | short | medshort | medlong | long | vlong |
|---|---|---|---|---|---|
| Min | 98.00 | 329.00 | 620.00 | 968.00 | 1376.00 |
| 1st Qu | 129.00 | 422.00 | 785.00 | 1159.00 | 1597.00 |
| Median | 140.00 | 445.00 | 830.00 | 1228.00 | 1676.00 |
| Mean | 140.50 | 447.60 | 831.00 | 1227.00 | 1674.00 |
| 3rd Qu | 151.00 | 472.00 | 877.00 | 1293.00 | 1752.00 |
| Max | 213.00 | 623.00 | 1098.00 | 1532.00 | 2045.00 |

FIG. 9: Feature Histogram Summary



## B. Model Selection

### 1. Deterministic and Stochastic Model Selection

Here we outline two algorithms used in deterministic and stochastic model selection process.

**Algorithm 1 (Bias Reduction CV Process)** *For a given $K$-fold training/validation set split and a candidate model* ***MOD****:*

1. ***STEP 1:*** *initialize local training set and validation set.*

2. ***STEP 2:*** *update **MOD** by changing the fitting data set to the local training set, thus, the estimated coefficients are changed.*

3. ***STEP 3:*** *compute prediction on the validation set using the updated **MOD***

As we mentioned in III A, our candidate models are fitted on the entire training set. If we use these models directly on the validation sets when doing cross-validation, it would incur huge bias as the estimated coefficients do account for the validation data in the fitting process.

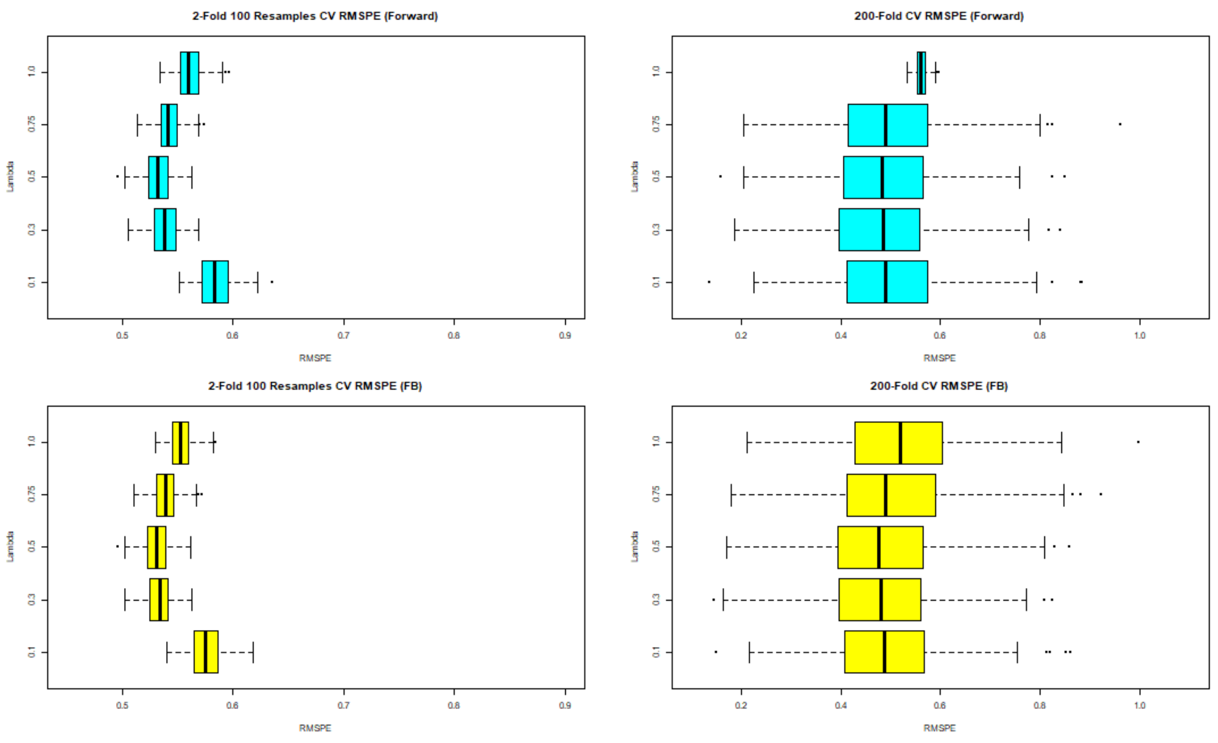
**Algorithm 2 (Linear $\lambda$ Tuning)** *Let $\vec{\lambda} = \{a = \lambda_1 < \cdots < \lambda_n = b\}$, **MOD** be a model that required $\lambda$ as the parameter for selection.*

1. ***STEP 1:*** *for each $\lambda_i$, perform 2 cross-validation schemes mentioned in III C and store corresponding RMSPE results $\mathsf{RMSPE}_{\lambda_i}$ respectively.*

2. ***STEP 2:*** *compare the mean RMSPE among $\{\mathsf{RMSPE}_{\lambda_i}\}_{i=1}^{n}$ **return** $\lambda_i^*$ that leads to the lowest mean RMSPE.*

Indeed, this is a greedy algorithm that might not give us the global optimum. However, this does let us test different $\lambda$ values when fitting the candidate models need in III A in considerably and computationally efficient way. In the actual process, we tested $\lambda \in \{0.1, 0.3, 0.5, 0.75, 1.0\}$. As $\lambda$ decreases, it takes much longer to obtain a candidate model as the penalty is not as high as other $\lambda$ and it leads to additional rounds of search in the **Forward**, **FB**, and **ICM** models. However, as shown below, we can see some upside by performing this analysis.

FIG. 10: RMSPE for Search Stratgies with Different $\lambda$



From figure:10, we can see that $\lambda = 0.5$ has consistently better resulting RMSPE across two different CV schemes. Thus, we have $\lambda = 0.5$ in L0.

## C. Kernel PCA[1]

### 1. Principal Component Analysis

Principal Component Analysis (PCA) is a common tool when it comes to dimension reduction and extraction of uncorrelated features from original feature data. In simplest term, it aims to find new features that account for most amount of the variance in the original feature space. Let $X \in \mathbb{R}^{N\times P}$ denote the data matrix where $X_{i,\cdot}$ is a $P \times 1$ vector with each original feature ($P$ in total) as its entry. PCA aims to find a subspace of dimension $Q < P$ and project the original data rows into this subspace. One important property of this subspace is that it has a orthogonal basis formed by *principal components (PC)*. We denote these basis vectors (PC) as $\{z_1, \cdots, z_Q\}$ To avoid dominance from features with large variance, we standardize each feature by the following formula

$$X^*_{i,j} = \frac{X_{i,j} - \bar{X}_{\cdot,j}}{\sigma(X_{\cdot,j})}, i \in \{1, \cdots, N\}, j \in \{1, \cdots, P\} \tag{std}$$

where $\bar{X}_{\cdot,j}$ is the sample mean of the $j$-th feature and $\sigma(X_{\cdot,j})$ is the sample standard deviation of the $j$-th feature. Since the change of basis map is linear, the process of solving for the $j-$th PC can be formulated as a constrained optimization problem outlined below:

$$\begin{aligned} \max \quad & \mathsf{Var}(z_j) \\ \text{s.t.} \quad a_j &= [a_{1j}, \cdots, a_{pj}] \\ z_j &= a_j^\top X^* \\ \mathsf{Cov}(z_j, z_h) &= 0, \;\; j > h \geq 1 \\ a_j^\top a_j &= 1 \end{aligned}$$

This yields the maximized result

$$\mathsf{Var}(z_j) = a_j^\top S a_j = \lambda_j$$

where $S = \frac{1}{N-1} X^{*\top} X^*$ is the sample covariance matrix of the data matrix $X^*$ (or correlation matrix of the original matrix $X$) and $\lambda_j$ is the $j$-th largest eigenvalue of $S$ where $a_j$ is the corresponding eigenvector. Let

$$A = [a_1, \cdots, a_P]$$

, which is orthogonal, then all PCs are given by $z = A^\top X^*$. Since the eigenvalues $\{\lambda_1, \cdots, \lambda_P\}$ represent the amount of variance captured by each PC. Thus, we can select $Q$ PCs based certain threshold $0\% < L < 100\%$ as follow

$$\min Q, \qquad \text{s.t.} \; \frac{\sum_{k=1}^{Q} \lambda_k}{\sum_{i=1}^{P} \lambda_i} \geq L \tag{thr}$$

These PCs will be used as extracted features to proceed with multiple linear regression against the response variate.

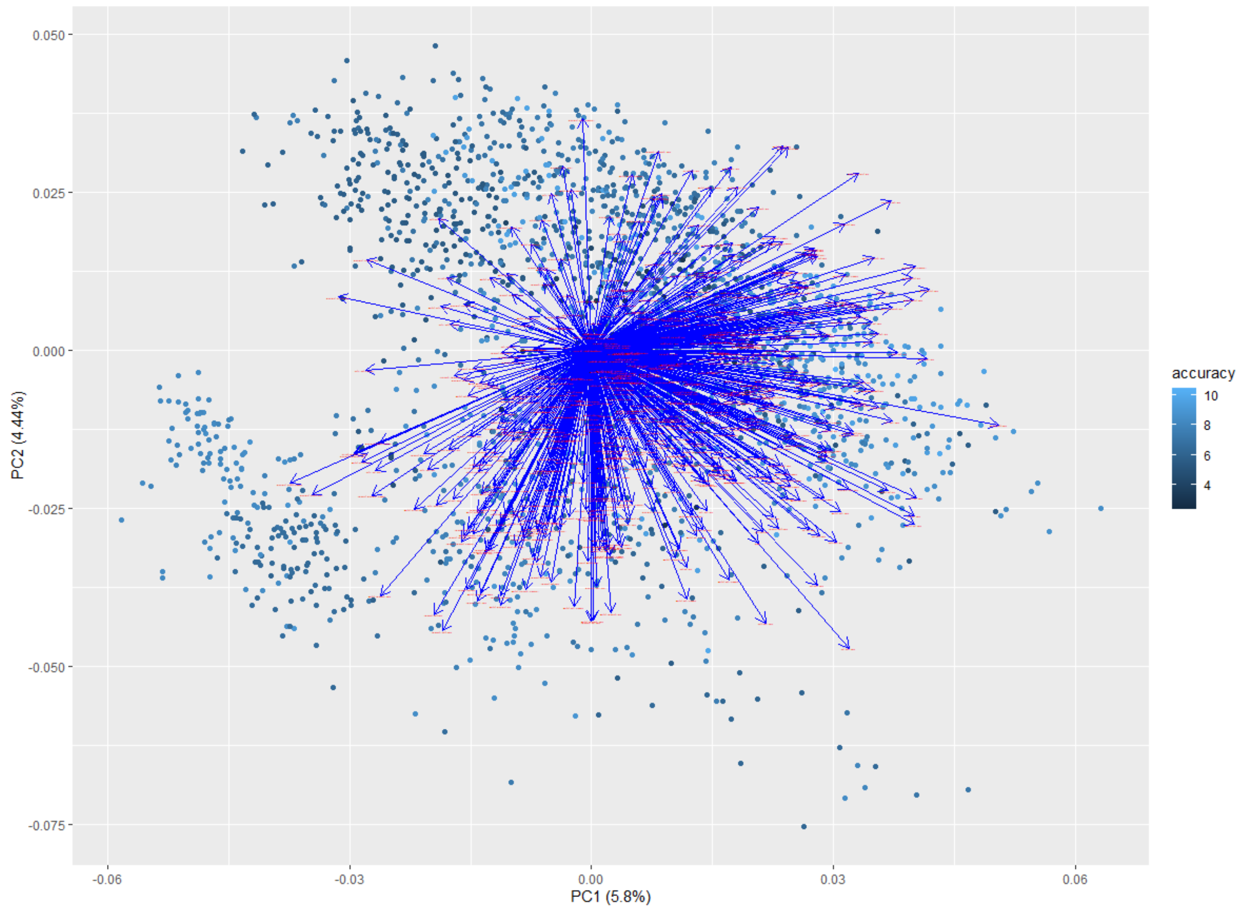


FIG. 11: PCA Biplot

From this PCA biplot (figure:11), we see that the first and second principal components do not explain the variance in the feature space well. And all the other PCs do not have a clear and centralized direction. This indicates that the original feature space might not contain high degree of correlated features. And if we want to use PCs as extracted features, we might want to consider the following kernel PCA method.

### *2. Kernel Principal Component Analysis*

PCA in general works well with linearly related features. Therefore, it is also called the linear PCA sometimes. However, for this project, one of the major obstacle is to consider any interaction between features as this area of study is not fully developed. It would be blind to consider manual interaction of features without any scientific evidence. The size of the predictors also erodes the idea to perform detailed observations (scatterplot, histogram, etc.) on these predictors to observe any patterns.

Kernel PCA provides us the possibility to touch on the non-linear feature space. In other to consider interaction terms, one of the major concerns for this many predictors is the humongous dimension of the extended feature space and our data rows are simply not large enough to have a invertible $X^\top X$ which is required to perform multiple linear regression. Instead of extending the feature space to include further interactions and use PCA on the extended feature space, the kernel PCA can perform these two steps at once without explicitly define the mapping from the original feature space to the extended feature space. Instead of finding the eigenvalues of $S$, the kernel PCA tells us that we can consider

$$U_{i,j} = K(X^*_{i,\cdot}, X^*_{j,\cdot}), i, j \in \{1, \cdots, N\}$$

where $K$ is the kernel function, essentially, any inner product on $\mathbb{R}^P \times \mathbb{R}^P$. By diagonalization of $U$ and ranking the eigenvalues, we can find $N$ PCs. As we can see that since the dimension of the extended feature space is huge (could be infinite), our PC candidates are much more than the case in PCA, which was $P$. Nonetheless, if we consider the polynomial kernel $K(x_i, x_j) = \langle x_i, x_j \rangle^d$, which is a popular one that extends the feature space to polynomial feature space, this provides us a path consider non-linear relationship between features without doing it explicitly.

### *3. Computational Kernel PCA*

Again, we consider the standardized feature matrix $X^*$ shown in (std). For a given training set and testing set, we perform kernel PCA on the training set to get the PCs $\{z_1, \cdots, z_h\}$ where $h < N$ with respect to some threshold. For our analysis, we use the common threshold of 90% based on (thr). Correspondingly, we save the projection matrix

$$A = [a_1, \cdots, a_h]$$

this is reserved so that we can project the testing feature space into the same space as the projected feature space in the training set.

### *4. BCE - Box-Cox-Extended Transformation*

The original Box-Cox transformation for a given $\lambda$ and vector $y$ is defined by

$$g(y_i) = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \log(y_i) & \lambda = 0 \end{cases}$$

As an injective map, it does have an inverse function given below:

$$g^{-1}(x_i) = \begin{cases} (\lambda x_i + 1)^{1/\lambda} & \lambda \neq 0 \\ \exp(x_i) & \lambda = 0 \end{cases}$$

However, when we use this transform to fit a MLR, the predicted value might not be in the domain of $g^{-1}$. This will result in missing predicted value. To ensure that the inverse is defined, we consider the Box-Cox-Extended transformation (**BCE**) with a defined inverse.

**Definition 1 (BCE)** *Let $g$ be the original Box-Cox transform on vector $y \in \mathbb{R}^n$ and pa-*

*rameter is $\lambda$. Its **BCE** inverse is*

$$\mathit{BCE}(x_i) = \begin{cases} g^{-1}(x_i) & x_i \in \mathit{Dom}(g^{-1}) \\ \min(y) & x_i \notin \mathit{Dom}(g^{-1}) \end{cases}$$

*where* $\min(y)$ *is the smallest response in the training set. We do know that this special treatment is rarely used but serve as a fail save for our future prediction.*

## D. Cross Validation Schemes

### 1. 2-Fold with 100 Resample

This cross-validation scheme is designed to test whether the candidate model can withstand the 1-to-1 training/testing ratio, which will be used in practice. Indeed, if we only do 2-Fold cross validation, there will be only two RMSPE to show us evidence of predictability. Thus, we decide to resample the 2-Fold training/validation sets for 100 times to give us 200 RMSPE estimate in the end. For computation's sake, we used the random resampling method without replacement. This do incur a possibility of resampling the same 2-Fold training/validation sets. However, we believe 100 resampling times is small enough to encounter such case.

### 2. 200-Fold

This cross-validation scheme is designed to test when the candidate model is training on most of the known data how well it can perform on a smaller validation set.

## E. Final Model and Summary Result

Our final **FB** model has the following form

```
accuracy ~ aliph1HC_aliph2HC_long + scLysN_bbC_vlong + aliph2HC_bbN_medshort
+ aliph1HC_aromaticC_medshort + aromaticC_hydroxylO_medlong + car-
bonylC_aromaticC_short + aliph1HC_aromaticC_vlong + bbN_bbCA_medlong +
aromaticC_sulfur_short + aliph1HC_aromaticC_long + aliph1HC_aliph1HC_vlong
+ bbC_bbC_medshort + scAGN_bbN_long + carboxylC_bbN_vlong + aro-
maticC_hydroxylO_long + sulfur_bbC_medlong + aliph2HC_bbO_vshort +
carboxylC_bbN_long + aliph1HC_bbO_long + aliph2HC_aromaticC_vlong +
aliph1HC_aromaticC_medlong + aromaticC_aromaticC_vlong + aliph3HC_bbC_vlong
+ aliph3HC_bbN_short + aliph1HC_bbO_medlong + aliph1HC_bbCA_medshort
+ carbonylC_bbC_medlong + scAGN_bbN_medlong + aliph2HC_bbN_medlong +
aliph1HC_sulfur_short + aliph2HC_aliph3HC_vlong + aliph2HC_scArgN_vlong + aro-
maticC_sulfur_long + sulfur_bbC_vlong + scArgN_bbO_medlong + bbO_bbO_short +
aliph3HC_hydroxylO_short + carbonylC_bbProN_medlong + aromaticC_bbO_vlong +
aliph3HC_aromaticC_long + aliph1HC_scArgN_long + carboxylO_carboxylO_vlong +
aliph2HC_bbN_vlong + bbN_bbC_vlong + bbCA_bbO_vshort + aliph1HC_bbCA_vlong
+ aliph1HC_bbProN_medlong + bbO_bbO_long + aliph1HC_aliph3HC_short +
aliph1HC_bbProN_vlong + scArgN_carboxylO_long + carboxylC_carboxylC_vlong
+ scLysN_carboxylO_long + scLysN_bbN_vlong + aliph1HC_aliph3HC_medlong
+ aliph1HC_aliph1HC_long + aliph1HC_bbN_medshort + carbonylC_bbProN_long
+ aliph2HC_aliph3HC_short + aromaticC_bbO_vshort + aromaticC_bbC_short
+ scAGN_bbC_vlong + carbonylC_hydroxylO_vlong + bbCA_bbCA_vlong +
bbN_bbN_medshort + aliph2HC_scLysN_medlong + sulfur_bbCA_short + sul-
fur_sulfur_vlong + carbonylC_scLysN_vlong + carbonylC_sulfur_short + hydrox-
ylO_sulfur_short + aliph1HC_sulfur_medshort + hydroxylO_sulfur_long + hydrox-
ylO_sulfur_medshort + hydroxylO_sulfur_vlong + aliph3HC_aromaticC_medlong
+ aliph3HC_bbN_medlong + aliph3HC_bbN_long + bbProN_carboxylO_vlong +
aliph1HC_hydroxylO_vlong + aliph1HC_hydroxylO_long + aromaticC_scLysN_vlong
+ aliph3HC_aromaticC_vlong + aliph1HC_aliph3HC_vlong + scAGN_hydroxylO_long
+ aromaticC_scAGN_long + aliph3HC_bbCA_short + aliph1HC_hydroxylO_medlong
+ aliph1HC_hydroxylO_medshort + aliph3HC_hydroxylO_medshort + aromat-
icC_bbO_medlong + carbonylC_aliph3HC_medlong + aliph1HC_bbO_medshort
+ aromaticC_bbCA_vlong + scArgN_bbO_long + bbCA_bbC_medshort +
aliph2HC_aliph3HC_medlong + scAGN_bbProN_medshort + aliph3HC_aliph3HC_vlong
+ aliph2HC_aliph2HC_short + aromaticC_hydroxylO_vshort + carbonylC_scAGN_short
+ scLysN_hydroxylO_long + carboxylO_bbC_medshort + sulfur_bbO_long +
sulfur_bbO_vlong + aromaticC_carbonylO_medlong + bbProN_bbCA_medshort
+ bbProN_bbN_long + carbonylC_bbProN_vlong + carboxylC_bbC_vlong +
carbonylO_bbO_long + aliph3HC_bbO_medshort + aliph3HC_bbProN_medlong
+ scAGN_bbO_short + hydroxylO_sulfur_medlong + bbProN_bbN_medlong +
aliph3HC_scAGN_medshort + carbonylC_aliph3HC_short + bbC_bbO_medlong
+ scAGN_bbO_medlong + aliph2HC_aromaticC_medshort + aromat-
icC_hydroxylO_medshort + aliph2HC_aliph3HC_medshort + scLysN_bbCA_long +
scLysN_carbonylO_vlong + aliph2HC_scLysN_medshort + carboxylC_hydroxylO_short
+ bbN_bbO_vlong + aliph3HC_bbProN_long + carbonylC_bbN_vlong + car-
bonylO_bbN_medlong + scAGN_bbN_vlong + scAGN_hydroxylO_vshort + aromat-
icC_scLysN_medlong + hydroxylO_bbN_medlong + aliph1HC_aliph2HC_medshort
+ bbCA_bbC_vlong + scAGN_carboxylO_vshort + carboxylO_bbC_vlong
+ bbN_bbO_medshort + scAGN_bbO_long + aliph2HC_bbC_vlong +
aliph2HC_bbC_long + carboxylC_bbC_medlong + bbC_bbO_vlong + aromat-
icC_bbN_vlong + aromaticC_bbProN_medlong + aliph1HC_carbonylO_medshort +
carboxylC_aliph2HC_long + carboxylO_bbN_vlong + hydroxylO_hydroxylO_medlong
+ aliph1HC_carboxylO_vlong + carboxylO_bbN_medlong + aliph3HC_sulfur_medshort
+ aliph3HC_sulfur_medlong + hydroxylO_bbO_long + carbonylC_carbonylO_vlong
```

The model summary is given below.

| | Estimate | Standard Error | p - value |
|---|---|---|---|
| Intercept | 3.68e+00 | 4.03e-01 | 1.66e-19 |

| | | | |
|---|---|---|---|
| aliph1HC__aliph2HC__long | 3.73e-02 | 7.28e-03 | 3.24e-07 |
| scLysN__bbC__vlong | -1.20e-01 | 1.71e-02 | 3.03e-12 |
| aliph2HC__bbN__medshort | -3.69e-02 | 5.72e-03 | 1.49e-10 |
| aliph1HC__aromaticC__medshort | 1.08e-01 | 1.06e-02 | 8.50e-24 |
| aromaticC__hydroxylO__medlong | -3.87e-02 | 7.15e-03 | 6.74e-08 |
| carbonylC__aromaticC__short | 1.05e-01 | 2.75e-02 | 1.46e-04 |
| aliph1HC__aromaticC__vlong | 7.99e-02 | 7.90e-03 | 1.96e-23 |
| bbN__bbCA__medlong | -3.15e-02 | 4.15e-03 | 5.02e-14 |
| aromaticC__sulfur__short | -5.58e-02 | 1.43e-02 | 1.02e-04 |
| aliph1HC__aromaticC__long | 7.99e-02 | 8.90e-03 | 6.65e-19 |
| aliph1HC__aliph1HC__vlong | 1.30e-01 | 2.35e-02 | 3.73e-08 |
| bbC__bbC__medshort | -2.21e-02 | 9.32e-03 | 1.81e-02 |
| scAGN__bbN__long | 4.62e-02 | 6.32e-03 | 4.10e-13 |
| carboxylC__bbN__vlong | 3.12e-02 | 1.31e-02 | 1.77e-02 |
| aromaticC__hydroxylO__long | -3.35e-02 | 5.45e-03 | 9.41e-10 |
| sulfur__bbC__medlong | -5.48e-02 | 9.63e-03 | 1.43e-08 |
| aliph2HC__bbO__vshort | 6.33e-02 | 2.30e-02 | 5.96e-03 |
| carboxylC__bbN__long | 7.21e-02 | 1.75e-02 | 3.97e-05 |
| aliph1HC__bbO__long | -8.86e-02 | 8.48e-03 | 6.79e-25 |
| aliph2HC__aromaticC__vlong | 1.28e-02 | 2.45e-03 | 1.99e-07 |
| aliph1HC__aromaticC__medlong | 7.56e-02 | 9.31e-03 | 8.94e-16 |
| aromaticC__aromaticC__vlong | -1.08e-02 | 2.43e-03 | 8.97e-06 |
| aliph3HC__bbC__vlong | 1.17e-02 | 3.86e-03 | 2.41e-03 |
| aliph3HC__bbN__short | 8.07e-02 | 1.77e-02 | 5.54e-06 |
| aliph1HC__bbO__medlong | -1.24e-01 | 1.10e-02 | 7.24e-29 |
| aliph1HC__bbCA__medshort | 7.32e-02 | 1.93e-02 | 1.60e-04 |
| carbonylC__bbC__medlong | -2.66e-02 | 8.35e-03 | 1.49e-03 |
| scAGN__bbN__medlong | 1.58e-02 | 7.21e-03 | 2.84e-02 |
| aliph2HC__bbN__medlong | 9.64e-03 | 3.65e-03 | 8.43e-03 |
| aliph1HC__sulfur__short | 2.32e-01 | 4.40e-02 | 1.43e-07 |
| aliph2HC__aliph3HC__vlong | 1.87e-02 | 4.27e-03 | 1.23e-05 |
| aliph2HC__scArgN__vlong | 1.46e-01 | 2.08e-02 | 3.27e-12 |
| aromaticC__sulfur__long | 3.63e-02 | 9.43e-03 | 1.21e-04 |
| sulfur__bbC__vlong | -1.88e-02 | 6.96e-03 | 7.00e-03 |
| scArgN__bbO__medlong | 3.65e-01 | 7.89e-02 | 4.14e-06 |
| bbO__bbO__short | -4.94e-02 | 1.12e-02 | 1.05e-05 |
| aliph3HC__hydroxylO__short | -1.31e-01 | 1.96e-02 | 2.61e-11 |
| carbonylC__bbProN__medlong | -1.32e-01 | 2.88e-02 | 4.75e-06 |
| aromaticC__bbO__vlong | 1.31e-02 | 2.37e-03 | 3.56e-08 |
| aliph3HC__aromaticC__long | -1.85e-02 | 3.76e-03 | 9.74e-07 |
| aliph1HC__scArgN__long | 2.88e-01 | 8.23e-02 | 4.83e-04 |
| carboxylO__carboxylO__vlong | -5.93e-02 | 1.99e-02 | 2.97e-03 |
| aliph2HC__bbN__vlong | 6.56e-03 | 2.55e-03 | 1.02e-02 |
| bbN__bbC__vlong | -1.34e-02 | 2.82e-03 | 2.28e-06 |
| bbCA__bbO__vshort | -3.53e-01 | 6.20e-02 | 1.40e-08 |

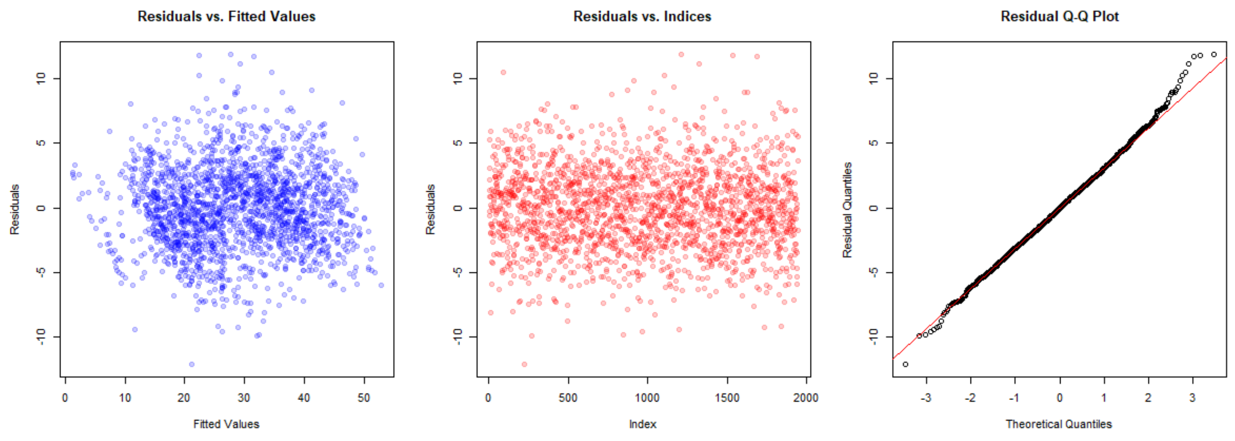
| | | | |
|---|---|---|---|
| aliph1HC__bbCA__vlong | 2.19e-02 | 6.63e-03 | 9.53e-04 |
| aliph1HC__bbProN__medlong | 3.19e-01 | 4.86e-02 | 6.89e-11 |
| bbO__bbO__long | -1.76e-02 | 4.55e-03 | 1.18e-04 |
| aliph1HC__aliph3HC__short | 1.64e-01 | 3.13e-02 | 1.89e-07 |
| aliph1HC__bbProN__vlong | 6.47e-02 | 2.39e-02 | 6.90e-03 |
| scArgN__carboxylO__long | -7.17e-02 | 2.36e-02 | 2.44e-03 |
| carboxylC__carboxylC__vlong | -1.62e-01 | 3.48e-02 | 3.31e-06 |
| scLysN__carboxylO__long | 1.45e-01 | 2.61e-02 | 3.29e-08 |
| scLysN__bbN__vlong | -5.21e-02 | 1.59e-02 | 1.09e-03 |
| aliph1HC__aliph3HC__medlong | -5.76e-02 | 1.38e-02 | 3.28e-05 |
| aliph1HC__aliph1HC__long | 1.03e-01 | 3.35e-02 | 2.01e-03 |
| aliph1HC__bbN__medshort | -6.96e-02 | 1.51e-02 | 4.56e-06 |
| carbonylC__bbProN__long | -9.49e-02 | 2.21e-02 | 1.77e-05 |
| aliph2HC__aliph3HC__short | 5.74e-02 | 1.25e-02 | 4.24e-06 |
| aromaticC__bbO__vshort | -2.18e-01 | 4.19e-02 | 2.24e-07 |
| aromaticC__bbC__short | 1.06e-01 | 2.04e-02 | 2.50e-07 |
| scAGN__bbC__vlong | -1.73e-02 | 4.64e-03 | 1.98e-04 |
| carbonylC__hydroxylO__vlong | -4.14e-02 | 1.02e-02 | 5.35e-05 |
| bbCA__bbCA__vlong | -1.75e-02 | 4.24e-03 | 3.81e-05 |
| bbN__bbN__medshort | -6.17e-02 | 9.63e-03 | 1.86e-10 |
| aliph2HC__scLysN__medlong | -5.87e-02 | 1.85e-02 | 1.51e-03 |
| sulfur__bbCA__short | -1.05e-01 | 2.30e-02 | 4.85e-06 |
| sulfur__sulfur__vlong | 1.16e-01 | 3.53e-02 | 1.05e-03 |
| carbonylC__scLysN__vlong | 9.74e-02 | 3.03e-02 | 1.32e-03 |
| carbonylC__sulfur__short | -1.77e-01 | 3.32e-02 | 1.21e-07 |
| hydroxylO__sulfur__short | 1.71e-01 | 3.36e-02 | 3.73e-07 |
| aliph1HC__sulfur__medshort | -6.40e-02 | 2.32e-02 | 5.88e-03 |
| hydroxylO__sulfur__long | 7.39e-02 | 1.80e-02 | 4.34e-05 |
| hydroxylO__sulfur__medshort | 1.43e-01 | 2.62e-02 | 5.25e-08 |
| hydroxylO__sulfur__vlong | 4.44e-02 | 1.45e-02 | 2.25e-03 |
| aliph3HC__aromaticC__medlong | -2.35e-02 | 5.17e-03 | 5.75e-06 |
| aliph3HC__bbN__medlong | 2.56e-02 | 5.68e-03 | 6.94e-06 |
| aliph3HC__bbN__long | 2.05e-02 | 4.79e-03 | 1.86e-05 |
| bbProN__carboxylO__vlong | 6.57e-02 | 3.28e-02 | 4.52e-02 |
| aliph1HC__hydroxylO__vlong | 9.86e-02 | 1.44e-02 | 9.64e-12 |
| aliph1HC__hydroxylO__long | 1.04e-01 | 1.79e-02 | 7.50e-09 |
| aromaticC__scLysN__vlong | -5.06e-02 | 1.89e-02 | 7.64e-03 |
| aliph3HC__aromaticC__vlong | -1.98e-02 | 3.51e-03 | 1.82e-08 |
| aliph1HC__aliph3HC__vlong | -3.08e-02 | 9.68e-03 | 1.52e-03 |
| scAGN__hydroxylO__long | -3.14e-02 | 1.12e-02 | 5.19e-03 |
| aromaticC__scAGN__long | 2.16e-02 | 6.36e-03 | 7.10e-04 |
| aliph3HC__bbCA__short | 7.95e-02 | 2.09e-02 | 1.44e-04 |
| aliph1HC__hydroxylO__medlong | 1.55e-01 | 2.57e-02 | 1.76e-09 |
| aliph1HC__hydroxylO__medshort | 1.75e-01 | 2.95e-02 | 3.18e-09 |
| aliph3HC__hydroxylO__medshort | -7.88e-02 | 1.44e-02 | 4.97e-08 |

| | | | |
|---|---|---|---|
| aromaticC__bbO__medlong | 1.43e-02 | 3.94e-03 | 2.77e-04 |
| carbonylC__aliph3HC__medlong | 3.10e-02 | 1.02e-02 | 2.44e-03 |
| aliph1HC__bbO__medshort | -5.49e-02 | 1.33e-02 | 3.67e-05 |
| aromaticC__bbCA__vlong | 8.69e-03 | 2.81e-03 | 2.01e-03 |
| scArgN__bbO__long | -2.61e-01 | 5.60e-02 | 3.51e-06 |
| bbCA__bbC__medshort | -2.83e-02 | 7.23e-03 | 9.59e-05 |
| aliph2HC__aliph3HC__medlong | 2.08e-02 | 5.76e-03 | 3.03e-04 |
| scAGN__bbProN__medshort | -9.67e-02 | 3.98e-02 | 1.50e-02 |
| aliph3HC__aliph3HC__vlong | 3.23e-02 | 8.91e-03 | 2.92e-04 |
| aliph2HC__aliph2HC__short | 4.45e-02 | 1.62e-02 | 6.11e-03 |
| aromaticC__hydroxylO__vshort | -1.69e-01 | 3.54e-02 | 1.99e-06 |
| carbonylC__scAGN__short | -6.65e-02 | 1.79e-02 | 2.06e-04 |
| scLysN__hydroxylO__long | 1.10e-01 | 3.56e-02 | 2.06e-03 |
| carboxylO__bbC__medshort | 4.35e-02 | 2.11e-02 | 3.94e-02 |
| sulfur__bbO__long | -2.59e-02 | 8.22e-03 | 1.67e-03 |
| sulfur__bbO__vlong | -2.47e-02 | 7.21e-03 | 6.15e-04 |
| aromaticC__carbonylO__medlong | 1.90e-02 | 7.30e-03 | 9.23e-03 |
| bbProN__bbCA__medshort | 9.55e-02 | 2.11e-02 | 6.56e-06 |
| bbProN__bbN__long | 5.33e-02 | 1.24e-02 | 1.70e-05 |
| carbonylC__bbProN__vlong | -6.70e-02 | 1.94e-02 | 5.83e-04 |
| carboxylC__bbC__vlong | -3.39e-02 | 1.28e-02 | 8.09e-03 |
| carbonylO__bbO__long | 1.63e-02 | 6.06e-03 | 7.31e-03 |
| aliph3HC__bbO__medshort | 1.92e-02 | 7.86e-03 | 1.48e-02 |
| aliph3HC__bbProN__medlong | -9.11e-02 | 2.31e-02 | 8.12e-05 |
| scAGN__bbO__short | 5.24e-02 | 1.48e-02 | 4.11e-04 |
| hydroxylO__sulfur__medlong | 6.87e-02 | 2.28e-02 | 2.61e-03 |
| bbProN__bbN__medlong | 4.10e-02 | 1.33e-02 | 2.09e-03 |
| aliph3HC__scAGN__medshort | 3.54e-02 | 1.24e-02 | 4.21e-03 |
| carbonylC__aliph3HC__short | -6.61e-02 | 2.15e-02 | 2.19e-03 |
| bbC__bbO__medlong | -9.72e-03 | 4.04e-03 | 1.63e-02 |
| scAGN__bbO__medlong | 2.48e-02 | 8.19e-03 | 2.45e-03 |
| aliph2HC__aromaticC__medshort | 2.19e-02 | 4.20e-03 | 2.06e-07 |
| aromaticC__hydroxylO__medshort | -3.20e-02 | 8.82e-03 | 2.98e-04 |
| aliph2HC__aliph3HC__medshort | 2.07e-02 | 7.98e-03 | 9.76e-03 |
| scLysN__bbCA__long | -7.13e-02 | 2.18e-02 | 1.07e-03 |
| scLysN__carbonylO__vlong | -6.90e-02 | 2.95e-02 | 1.97e-02 |
| aliph2HC__scLysN__medshort | -9.55e-02 | 2.76e-02 | 5.60e-04 |
| carboxylC__hydroxylO__short | -1.56e-01 | 5.20e-02 | 2.69e-03 |
| bbN__bbO__vlong | 6.77e-03 | 2.80e-03 | 1.55e-02 |
| aliph3HC__bbProN__long | -4.34e-02 | 1.78e-02 | 1.47e-02 |
| carbonylC__bbN__vlong | -2.02e-02 | 5.46e-03 | 2.20e-04 |
| carbonylO__bbN__medlong | -2.33e-02 | 6.81e-03 | 6.20e-04 |
| scAGN__bbN__vlong | 1.02e-02 | 5.35e-03 | 5.70e-02 |
| scAGN__hydroxylO__vshort | 5.77e-02 | 2.21e-02 | 9.20e-03 |
| aromaticC__scLysN__medlong | -6.73e-02 | 2.34e-02 | 4.08e-03 |

| | | | |
|---|---|---|---|
| hydroxylO__bbN__medlong | 1.91e-02 | 7.14e-03 | 7.51e-03 |
| aliph1HC__aliph2HC__medshort | -3.07e-02 | 1.31e-02 | 1.95e-02 |
| bbCA__bbC__vlong | -7.74e-03 | 3.03e-03 | 1.07e-02 |
| scAGN__carboxylO__vshort | -9.16e-02 | 3.65e-02 | 1.21e-02 |
| carboxylO__bbC__vlong | 2.46e-02 | 8.11e-03 | 2.45e-03 |
| bbN__bbO__medshort | -1.55e-02 | 5.74e-03 | 7.08e-03 |
| scAGN__bbO__long | 1.33e-02 | 5.59e-03 | 1.77e-02 |
| aliph2HC__bbC__vlong | 7.59e-03 | 2.48e-03 | 2.24e-03 |
| aliph2HC__bbC__long | 8.54e-03 | 2.91e-03 | 3.35e-03 |
| carboxylC__bbC__medlong | 4.21e-02 | 1.78e-02 | 1.81e-02 |
| bbC__bbO__vlong | -6.48e-03 | 2.91e-03 | 2.63e-02 |
| aromaticC__bbN__vlong | 7.34e-03 | 2.77e-03 | 8.16e-03 |
| aromaticC__bbProN__medlong | -5.07e-02 | 1.90e-02 | 7.78e-03 |
| aliph1HC__carbonylO__medshort | 4.14e-02 | 1.96e-02 | 3.50e-02 |
| carboxylC__aliph2HC__long | -3.55e-02 | 1.32e-02 | 7.39e-03 |
| carboxylO__bbN__vlong | 2.11e-02 | 8.45e-03 | 1.26e-02 |
| hydroxylO__hydroxylO__medlong | 4.13e-02 | 2.07e-02 | 4.58e-02 |
| aliph1HC__carboxylO__vlong | -6.35e-02 | 2.77e-02 | 2.18e-02 |
| carboxylO__bbN__medlong | 3.11e-02 | 1.66e-02 | 6.02e-02 |
| aliph3HC__sulfur__medshort | 4.02e-02 | 1.43e-02 | 4.90e-03 |
| aliph3HC__sulfur__medlong | 2.96e-02 | 1.35e-02 | 2.92e-02 |
| hydroxylO__bbO__long | 1.20e-02 | 5.78e-03 | 3.83e-02 |
| carbonylC__carbonylO__vlong | 2.82e-02 | 1.48e-02 | 5.80e-02 |

TABLE IV: Final Model Summary

FIG. 12: Residual Plots Summary (with BCE)



## F. Source Code

```r
%# Required Packages
library(caret)
library(faraway)
library(scales)
library(kernlab)
library(MASS)
library(glmnet)
library(kableExtra)
library(ggfortify)
library(ggplot2)
library(doParallel)
library(corrplot)
library(xtable)

# Training data import
protein <-
read.csv("protein-train.csv")
protein_clean = readRDS(file =                          "protein_clean.Rda")

##############################
#      Helper Functions      #
##############################

## Check for static index
static_index <- function(vec){
    for (index in 1:(length(vec) - 1)){
        if (vec[index] == vec[index + 1]){
            return(index)
        }
    }
}

## Exclude exact multicolinearity
get_exclude_col <- function(df){
    exclude_col <- c()
    naive_model <- lm(accuracy ~ ., data = df)
    coe <- naive_model$coefficients
    exclude_index_vec <- which(is.na(coe))
    exclude_col <- c(exclude_col,
                     names(coe)[exclude_index_vec])
    return(exclude_col)
}

## Compute RMSE and RMSPE
RMSE <- function(test, pred){
  sqrt(mean((test - pred)^2))
}

## Iterative VIF Elimination
get_vif_exclude_col <- function(df, exclude_col){
    mult_col <- TRUE
    local_protein <- df
    while(mult_col){
        model <- lm(accuracy ~ .,
                    data = local_protein)
        vif_vec <- vif(model)
        max_vif <- max(vif_vec)
        if (max_vif < 10){
            mult_col = FALSE
            m2 <- model
        }
        else{
            max_index <- which(vif_vec == max_vif)
            var_remove_name <- names(vif_vec)[max_index]
            exclude_col <- c(exclude_col, var_remove_name)
            local_protein <-
                df[,-which(names(df) %in% exclude_col)]
        }
    }
    return(exclude_col)
}

## Standardization Scaler
standard_scale <- function(train, test){
  normParam <- preProcess(train)
  norm.train <- predict(normParam, train)
  norm.test <- predict(normParam, test)
  return(list(train = norm.train,
              test = norm.test))
}

## PCA Transform
pca_transform <- function(train, test){
    pca <- prcomp(train, scale. = T)
    pca.var <- pca$sdev^2
    pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
    stop_index <- static_index(cumsum(pca.var.per))
    pca_train <- data.frame(pca$x[, 1:stop_index])
    pca_test <- data.frame(predict(pca, newdata = test))
    return(list("pca_train" = pca_train,
                "pca_test" = pca_test))
}

## Model assumption check plot
model_check_plot <- function(model){
  par(mfrow = c(1,3))
  plot(model$fitted.values, model$residuals,
       xlab = "Fitted Values",
       ylab = "Residuals",
       main = "Residuals vs. Fitted Values",
       pch = 19,
       col = rgb(red=0,
                 green=0,
                 blue=1.0,
                 alpha=0.2))
  plot(1:length(model$fitted.values),
       model$residuals,
       xlab = "Index",
       ylab = "Residuals",
       main = "Residuals vs. Indices",
       pch = 19,
       col = rgb(red=1.0,
                 green=0,
                 blue=0,
                 alpha=0.2))
  qqnorm(model$residuals,
         main = "Residual Q-Q Plot",
         ylab = "Residual Quantiles")
  qqline(model$residuals, col = "red")
}

## Elastic-net lambda linear search (not used in report, but it is part of the
    process)

elas_net_summary <- function(trainSet, validSet,
                             S = 3){
  list.of.fits <- list()
  for (i in 0:S){
    fit.name <- paste0("alpha", i/S)
    list.of.fits[[fit.name]] <-
    cv.glmnet(as.matrix(trainSet[,-1]),
    trainSet$accuracy,
    type.measure = "mse",
    alpha = i/S,
    family = "gaussian")
  }
  results <- data.frame()
  for (i in 0:S){
    fit.name <- paste0("alpha", i/S)
    pred <- predict(list.of.fits[[fit.name]],
    s = list.of.fits[[fit.name]]$lambda.1se,
    newx = as.matrix(
    validSet[,-1])
  )
    RMSPE <- RMSE(validSet$accuracy, pred)
    temp <- data.frame(alpha = i/ S RMSPE = RMSPE,
    fit.name = fit.name)
    results <- rbind(results, temp)
  }
  return(results)
}

## ICM model generator
icm_model_gen <- function(trainSet, pen){
  pen <- pen * log(nrow(trainSet))
  varlist = c()
  varnames = names(trainSet)
  n = nrow(trainSet)
  varorder <- sample(1:ncol(trainSet))
  minCrit = Inf
  noChange = FALSE
  num_model = 0
  while (! noChange) {
    noChange = TRUE
    for (i in varorder) {
      num_model = num_model + 1
      if (i == 1) {
        next
      }
      if (i %in% varlist & length(varlist) > 1) {
        index = c(1, varlist[varlist != i])
        trainVars = trainSet[, index]
        fit = lm(accuracy ~ .,
                 data = trainVars)
        if (AIC(fit, k = pen) < minCrit) {
          minCrit = AIC(fit, k = pen)
          varlist = varlist[varlist != i]
          best.model = fit
          noChange = FALSE
        }
```

```r
      }
      else if (!i %in% varlist) {
        index = c(1, varlist, i)
        trainVars = trainSet[, index]
        fit = lm(accuracy ~ .,
                 data = trainVars)
        if (AIC(fit, k = pen) < minCrit) {
          minCrit = AIC(fit, k = pen)
          varlist = c(varlist, i)
          best.model = fit
          noChange = FALSE
        }
      }
    }
  }
  print(num_model)
  return(best.model)
}

## ICM model lambda linear tuning

icm_summary <- function(trainSet, validSet){
  pen_vec <- seq(from = 0.05, to = 2, by = 0.5)
  pen_vec <- pen_vec * log(nrow(trainSet))
  RMSPE_vec <- c()
  RMSE_vec <- c()
  pred_num_vec <- c()
  best_rmspe <- Inf
  for (pen in pen_vec){
    icm_best <- icm_model_gen(trainSet, pen)
    pred_num <- length(icm_best$coefficients)
    pred_num_vec <- c(pred_num_vec, pred_num)
    pred_icm <- predict(icm_best,
    newdata = validSet)
    local_RMSE <- sqrt(mean(icm_best$residuals^2))
    local_RMSPE <- RMSE(validSet[["accuracy"]], pred_icm)
    RMSE_vec <- c(RMSE_vec, local_RMSE)
    RMSPE_vec <- c(RMSPE_vec, local_RMSPE)
    if (local_RMSPE < best_rmspe){
      best_rmspe <- local_RMSPE
      best_model <- icm_best
    }
  }
  summary_df <- data.frame(pen_factor = pen_vec,
  pred_num = pred_num_vec,
  RMSE = RMSE_vec,
  RMSPE = RMSPE_vec)
  return(list(best_model = best_model,
  summary = summary_df))
}

## Backward selection model generator (not used)
backward_selection <- function(empty, full, pen){
  step.model <- MASS::stepAIC(object = full,
                              scope =
                                list(upper = full,
                                     lower = empty),
                              direction = "backward",
                              trace = T,
                              k = pen)
  print(summary(step.model))
  return(step.model)
}

## Forward selection model generator
forward_selection <- function(empty, full, pen){
  step.model <- MASS::stepAIC(object = empty,
                              scope =
                                list(upper = full,
                                     lower = empty),
                              direction = "forward",
                              trace = T,
                              k = pen)
  print(summary(step.model))
  return(step.model)
}

## Forward-backward selection model generator
fb_selection <- function(empty, full, pen){
  step.model <- MASS::stepAIC(object = empty,
                              scope =
                                list(upper = full,
                                     lower = empty),
                              direction = "both",
                              trace = T,
                              k = pen)
  print(summary(step.model))
  return(step.model)
}

```

```r
## Box-Cox Transformtion and BCE

boxcox_lambda <- function(model){
  bc <- boxcox(model)
  lambda <- bc$x[which.max(bc$y)]
  return(lambda)
}
boxcox_transform <- function(lambda, vec){
  if (lambda == 0){
    return(log(vec))
  }
  return((vec ^ lambda - 1) / lambda)
}
boxcox_transform_inv <- function(lambda, vec, train_vec){
  if (lambda == 0){
    result = exp(vec)
  }
  result = (lambda * vec + 1)^(1 / lambda)
  result = ifelse(is.na(result), min(train_vec),
                  result)
  return(result)
}

## Customized static index finder
static_x_index <- function(vec, x){
  for (index in 1:length(vec)){
    if (vec[index] > x){
      return(index)
    }
  }
  return(length(vec))
}

## KPCA Transform with plot
kpca_transform <- function(train, test,
                           degree = 2,
                           x = 90){
    scale_pair <- standard_scale(train, test)
    train <- scale_pair$train
    test <- scale_pair$test
    kpca_model <- kpca(~.,
                data = train,
                kernel = "polydot",
                kpar = list(degree = degree))

    kpca.var <- cumsum(eig(kpca_model))*100/
      sum(eig(kpca_model))
    # plot(1:length(kpca.var),kpca.var)
    stop_index <- static_x_index(kpca.var,
                                 x = x)
    kpca_train <- data.frame(
      predict(kpca_model, train))[, 1:stop_index]
    kpca_test <- data.frame(
      predict(kpca_model, test))[, 1:stop_index]
    return(list("kpca_train" = kpca_train,
                "kpca_test" = kpca_test))
}

## Optimal degree search for KPCA's polynomial kernel
kpca_degree_run <- function(validSetSplits, deg,
                            x){
  K <- 10
  local_RMSE_kpca <- c()
  for (k in 1:K){
    validSet <- protein[validSetSplits==k,]
    trainSet <- protein[validSetSplits!=k,]
    ## kpca

    exp_var_pair <- kpca_transform(trainSet[,-1],
                                   validSet[,-1],
                                   degree = deg,
                                   x = x)
    kpca_train <- cbind(trainSet[,1],
    exp_var_pair[["kpca_train"]])
    names(kpca_train) <- c("accuracy",
                           names(kpca_train)[-1])
    kpca_test <- cbind(validSet[,1],
    exp_var_pair[["kpca_test"]])
    names(kpca_test) <- c("accuracy",
                          names(kpca_test)[-1])
    model <- lm(accuracy ~ ., data = kpca_train)
    pred <- predict(model, newdata = kpca_test)
    local_RMSE <- RMSE(kpca_test$accuracy, pred)
    local_RMSE_kpca <- c(local_RMSE_kpca,
                          local_RMSE)
    cat("deg=",
        deg, ":", k, "-th run's kpca RMSE=",
        local_RMSE, "\n", sep = "")
  }
  return(mean(local_RMSE_kpca))
```

```r
}

## K-Fold cross validation for a candidate model on original feature space
cv_fold <- function(K, model,
                    validSetSplits){
  local_RMSE_vec <- c()
  for (k in 1:K){
    if (k % % 100 == 0) cat(run,
                           ":",
                           k,
                           "-th run\n",
                           sep = "")
    validSet <- protein_vif[validSetSplits==k,]
    trainSet <- protein_vif[validSetSplits!=k,]
    local_model <- update(model,
                          data = trainSet)
    local_pred <- predict(local_model,
                          newdata = validSet)
    local_RMSE <- RMSE(validSet$accuracy,
                       local_pred)
    local_RMSE_vec <- c(local_RMSE_vec,
                        local_RMSE)
  }
  return(local_RMSE_vec)
}

## K-Fold-N-Resample cross validation process for all candidate models
cv_run <- function(K, run_num){
  cl <- makePSOCKcluster(8)
  registerDoParallel(cl)
  start.time <- proc.time()
  set.seed(2020)
  N <- nrow(protein_vif)
  run_total <- run_num
  forward_RMSE <- c()
  fb_RMSE <- c()
  icm_RMSE <- c()
  kpca_RMSE_1 <- c()
  kpca_RMSE_2 <- c()
  kpca_RMSE_3 <- c()
  for (run in 1:run_total){
    validSetSplits <- sample((1:N)% %K + 1)
    local_forward_RMSE <- cv_fold(K,
                                  forward_model,
                                  validSetSplits)
    local_fb_RMSE <- cv_fold(K,
                             fb_model,
                             validSetSplits)
    local_icm_RMSE <- cv_fold(K,
                             icm_model,
                             validSetSplits)
    local_kpca_RMSE_1 <-
      kpca_cv_fold(K,validSetSplits, deg = 1)
    local_kpca_RMSE_2 <-
      kpca_cv_fold(K,validSetSplits, deg = 2)
    local_kpca_RMSE_3 <-
      kpca_cv_fold(K,validSetSplits, deg = 3)
    forward_RMSE <- c(forward_RMSE,
                       local_forward_RMSE)
    fb_RMSE <- c(fb_RMSE,
                       local_fb_RMSE)
    icm_RMSE <- c(icm_RMSE,
                       local_icm_RMSE)
    kpca_RMSE_1 <- c(kpca_RMSE_1,
                    local_kpca_RMSE_1)
    kpca_RMSE_2 <- c(kpca_RMSE_2,
                    local_kpca_RMSE_2)
    kpca_RMSE_3 <- c(kpca_RMSE_3,
                    local_kpca_RMSE_3)
  }

  RMSE_summary_df <-
    data.frame(
      forward = forward_RMSE,
      fb = fb_RMSE,
      icm = icm_RMSE,
      PCA = kpca_RMSE_1,
      KPCA2 = kpca_RMSE_2,
      KPCA3 = kpca_RMSE_3
    )
  stop.time <- proc.time()
  run.time <- stop.time - start.time
  print(run.time)
  stopCluster(cl)
  return(RMSE_summary_df)
}

## K-Fold-N-Resample for only candidate models on the original feature space
cv_run_ori <- function(K, run_num){
  cl <- makePSOCKcluster(8)
  registerDoParallel(cl)
  start.time <- proc.time()
  set.seed(2020)
  N <- nrow(protein_vif)
  run_total <- run_num
  forward_RMSE <- c()
  fb_RMSE <- c()
  icm_RMSE <- c()
  for (run in 1:run_total){
    validSetSplits <- sample((1:N)% %K + 1)
    local_forward_RMSE <- cv_fold(K,
                                forward_model,
                                validSetSplits)
    local_fb_RMSE <- cv_fold(K,
                                fb_model,
                                validSetSplits)
    local_icm_RMSE <- cv_fold(K,
                                icm_model,
                                validSetSplits)
    forward_RMSE <- c(forward_RMSE,
                      local_forward_RMSE)
    fb_RMSE <- c(fb_RMSE,
                      local_fb_RMSE)
    icm_RMSE <- c(icm_RMSE,
                      local_icm_RMSE)
  }

  RMSE_summary_df <-
    data.frame(
      forward = forward_RMSE,
      fb = fb_RMSE,
      icm = icm_RMSE
    )
  stop.time <- proc.time()
  run.time <- stop.time - start.time
  print(run.time)
  stopCluster(cl)
  return(RMSE_summary_df)
}

## K-Fold cross-validation on KPCA method with different degrees
kpca_cv_fold <- function(K, validSetSplits,
                         deg){
  local_RMSE_vec <- c()
  for (k in 1:K){
    if (k % % 100 == 0) cat(run,
                             ":",
                             k,
                             "-th run\n",
                             sep = "")
    validSet <- protein_whole[validSetSplits==k,]
    trainSet <- protein_whole[validSetSplits!=k,]
    exp_var_pair <-
      kpca_transform(trainSet[,-1],
                     validSet[,-1],
                     degree = deg)
    kpca_train <- cbind(trainSet[,1],
    exp_var_pair[["kpca_train"]])
    names(kpca_train) <- c("accuracy",                           names(
   kpca_train)[-1])
    kpca_test <- cbind(validSet[,1],
    exp_var_pair[["kpca_test"]])
    names(kpca_test) <- c("accuracy",
                          names(kpca_test)[-1])
    local_model <- lm(accuracy ~ ., data = kpca_train)
    local_pred <- predict(local_model,
                    newdata = kpca_test)
    local_RMSE <- RMSE(validSet$accuracy,
                       local_pred)
    local_RMSE_vec <- c(local_RMSE_vec,
                        local_RMSE)
  }
  return(local_RMSE_vec)
}

## Summary boxplot of RMSPE across different candidate models
RMSE_boxplot <- function(forward_RMSE,
        fb_RMSE,
        icm_RMSE,
        kpca_RMSE_1 = kpca_RMSE_1,
        kpca_RMSE_2 = kpca_RMSE_2,
        kpca_RMSE_3 = kpca_RMSE_3,
        title){
boxplot(forward_RMSE,
        fb_RMSE,
        icm_RMSE,
        kpca_RMSE_1,
        kpca_RMSE_2,
        kpca_RMSE_3,
        horizontal = T,
```

```r
        main = title,
        xlab = "RMSPE",
        pch = 20,
        col = c("yellow",
                "orange",
                "green",
                "cyan",
                "red",
                "blue"),
        names = c("Forward",
                  "FB",
                  "ICM",
                  "PCA",
                  "KPCA2",
                  "KPCA3"))
}

## Cross-validation comparison between a model and a model with BCE
cv_bc_run <- function(K, run_num){
  set.seed(2020)
  N <- nrow(protein_vif)
  run_total <- run_num
  fb_RMSE <- c()
  fb_RMSE_bc <- c()
  for (run in 1:run_total){
    validSetSplits <- sample((1:N)% %K + 1)
    for (k in 1:K){
      validSet <- protein_vif[validSetSplits==k,]
      trainSet <- protein_vif[validSetSplits!=k,]
      local_model <- update(fb_model,
                            data = trainSet)
      local_pred <- predict(local_model,
                                  newdata = validSet)
      local_RMSE <- RMSE(validSet$accuracy,
                            local_pred)
      fb_RMSE <- c(fb_RMSE, local_RMSE)
      accuracy_lambda <- boxcox_transform(lambda,
                                          trainSet$accuracy)
      trainSet = cbind(accuracy_lambda, trainSet[,-1])
      local_model_bc <- update(fb_model_bc,
                               data = trainSet[,-1])

      local_pred_bc <- predict(local_model_bc,
                            newdata = validSet[,-1])
      local_pred_bc <- boxcox_transform_inv(lambda,
              local_pred_bc,
                                            trainSet$accuracy)
      local_RMSE_bc <- RMSE(validSet$accuracy,
                            local_pred_bc)
      fb_RMSE_bc <- c(fb_RMSE_bc,
                            local_RMSE_bc)
    }
  }
  return(data.frame(
    fb_RMSE = fb_RMSE,
    fb_RMSE_bc = fb_RMSE_bc
  ))
}

## Manual addition of 'angles' compared to FB model
cv_angle_run <- function(K, run_num){
  set.seed(2020)
  N <- nrow(protein_vif)
  run_total <- run_num
  fb_RMSE <- c()
  fb_RMSE_angle <- c()
  for (run in 1:run_total){
    validSetSplits <- sample((1:N)% %K + 1)
    for (k in 1:K){
      validSet <- protein_vif[validSetSplits==k,]
      trainSet <- protein_vif[validSetSplits!=k,]
      local_model <- update(fb_model,
                            data = trainSet)
      local_pred <- predict(local_model,
                                  newdata = validSet)
      local_RMSE <- RMSE(validSet$accuracy,
                            local_pred)
      fb_RMSE <- c(fb_RMSE, local_RMSE)
      local_model_angle <- update(fb_model_angle,
                               data = trainSet)
      local_pred_angle <- predict(local_model_angle,
                            newdata = validSet)
      local_RMSE_angle <- RMSE(validSet$accuracy,
                            local_pred_angle)
      fb_RMSE_angle <- c(fb_RMSE_angle,
                            local_RMSE_angle)
    }
  }
  return(data.frame(
    fb_RMSE = fb_RMSE,
    fb_RMSE_bc = fb_RMSE_angle
  ))
}

##############################
#      Data Preprocessing    #
##############################

## Aggregate atom/length types
name_list <- names(protein)
total_list <- c()
for (name in name_list){
    for (elem in strsplit(name, "_")[[1]]){
        if (!elem %in% total_list) {
            total_list <- c(total_list, elem)
        }
    }
}
atom_list = list()
for (coln in total_list){
    atom_list[coln] = NA
}
extra_df = as.data.frame(atom_list)
for (i in 1:dim(protein)[1]){
    local_list = list()
    for (coln in total_list){
        local_list[coln] = 0
    }
    for (j in 1:(dim(protein)[2] - 2)){
        local_name = strsplit(names(protein[,-c(1,2)])[j], "_")[[1]]
        for (name in local_name){
            local_list[name] = local_list[[name]] + protein[i,j + 2]
        }
    }
    new_df = as.data.frame(local_list)
    extra_df = rbind(extra_df, new_df)
}
extra_df = extra_df[-1,]
extra_df = cbind(protein$accuracy, extra_df)
new_df = cbind(protein[,1:2], extra_df[,-1])
saveRDS(new_df, file = "protein_clean.Rda")

## Length type pairplot
pairs(protein_clean[,c(1, 2,21:26)], pch = 19,
      col = rgb(red=1.0,
                green=0.2,
                blue=0.4,
                alpha=0.2))

## Atom type correlation heatmap
corrplot(cor(protein_clean[,c(1, 2, 3:20)]))

## Aggregated variable histogram summary
matrix(2:26, 5, 5, byrow = T)
format.matrix = cbind(matrix(rep(1, 5), 5, 1), matrix(2:26, 5, 5, byrow = T))
par(cex.axis = .6,
    mar = c(2,2,1,2),
    cex.main = 0.7)
layout(format.matrix,
   widths=c(2.5,rep(2, 5)))
hist(protein_clean[,1],
        col = "orange",
        main = "Protein Folding Accuracy",
        xlab = names(protein_clean)[1])

for (i in 2:(dim(protein_clean)[2])){
    hist(protein_clean[,i],
        col = "cyan",
        main = names(protein_clean)[i],
        xlab = names(protein_clean)[i])
}

## Generate feature summaries

temp1 = data.frame(summary(protein_clean))$Freq
temp1 = as.numeric(substr(temp1, 9, 100L))
temp1 = data.frame(matrix(temp1,
                          nrow=6,
                          ncol=26,
                          byrow=FALSE))
colnames(temp1) = colnames(protein_clean)
rownames(temp1) = c("Min", "1st Qu", "Median", "Mean","3rd Qu","Max")
protein_clean_summary = temp1
print(xtable(protein_clean_summary[,22:26]),
      booktabs=TRUE)

## Combine features
protein_clean = readRDS(file =                          "protein_clean.Rda")
protein_whole = cbind(protein, protein_clean[-c(1,2)])

```

```r
## PCA and Biplot
protein_exclude <- protein[,-which(names(protein)
%in% exclude_col)]
pca <- prcomp(protein_exclude, scale. = T)
autoplot(pca, data = protein,
         colour = 'accuracy',
         loadings = TRUE,
         loadings.colour = 'blue',
         loadings.label = TRUE,
         loadings.label.size = 0.1)

## Exclude multicolinearity and high VIF features
exclude_col <- get_exclude_col(protein_whole)
exclude_col <- get_vif_exclude_col(
  protein_whole[,-which(names(protein_whole) %in%
                         exclude_col)],
  exclude_col)
protein_vif <- protein_whole[,
                       -which(names(protein_whole)
                              %in%
                                exclude_col)]

##############################
#      Pre-Model Selection   #
##############################

## Compare basic MLR, PCA, KPCA2, and Elastic Net
set.seed(20201125)
N <- nrow(protein)
K <- 10
repeat_RMSE_naive <- c()
repeat_RMSE_pca <- c()
repeat_RMSE_kpca <- c()
repeat_RMSE_elas <- c()
for (run in 1:10){
  local_RMSE_naive <- c()
  local_RMSE_kpca <- c()
  local_RMSE_pca <- c()
  local_RMSE_elas <- c()
  validSetSplits <- sample((1:N)% %K + 1)
  for (k in 1:K){
    cat(run, ":", k, "-th run\n", sep = "")
    validSet <- protein[validSetSplits==k,]
    trainSet <- protein[validSetSplits!=k,]

    ## naive
    exclude_col <- get_exclude_col(trainSet)
    trainSet <- trainSet[, -which(names(trainSet)
    %in% exclude_col)]
    validSet <- validSet[, -which(names(validSet)
    %in% exclude_col)]
    model_naive <- lm(accuracy ~ ., data = trainSet)
    pred_naive <- predict(model_naive,
                          newdata = validSet)
    local_RMSE <- sqrt(mean((validSet$accuracy -
    pred_naive)^2))
    local_RMSE_naive <- c(local_RMSE_naive,
                          local_RMSE)
    cat(run, ":", k, "-th run's naive RMSE=",
        local_RMSE, "\n", sep = "")

    ## kpca

    exp_var_pair <- kpca_transform(trainSet[,-1],
                                   validSet[,-1])
    kpca_train <- cbind(trainSet[,1],
    exp_var_pair[["kpca_train"]])
    names(kpca_train) <- c("accuracy",
                           names(kpca_train)[-1])
    kpca_test <- cbind(validSet[,1],
    exp_var_pair[["kpca_test"]])
    names(kpca_test) <- c("accuracy",
                          names(kpca_test)[-1])
    model <- lm(accuracy ~ ., data = kpca_train)
    pred <- predict(model, newdata = kpca_test)
    local_RMSE <- RMSE(kpca_test$accuracy, pred)
    local_RMSE_kpca <- c(local_RMSE_kpca,
                          local_RMSE)
    cat(run, ":", k, "-th run's kpca RMSE=",
        local_RMSE, "\n", sep = "")

    ## pca
    exp_var_pair <- pca_transform(trainSet[,-1],
                                  validSet[,-1])
    pca_train <- cbind(trainSet[,1],
                       exp_var_pair[["pca_train"]])
    names(pca_train) <- c("accuracy",
                          names(pca_train)[-1])
    pca_test <- cbind(validSet[,1],
                      exp_var_pair[["pca_test"]])
```

```r
    names(pca_test) <- c("accuracy",
                         names(pca_test)[-1])
    model <- lm(accuracy ~ ., data = pca_train)
    pred <- predict(model, newdata = pca_test)
    local_RMSE <- RMSE(pca_test$accuracy, pred)
    local_RMSE_pca <- c(local_RMSE_pca,
                        local_RMSE)
    cat(run, ":", k, "-th run's pca RMSE=",
        local_RMSE, "\n", sep = "")

    ## Elas-Net
    elas_summ <- elas_net_summary(trainSet,
                                  validSet)
    local_RMSE <- min(elas_summ$RMSPE)
    cat(run, ":", k, "-th run's elas RMSE=",
        local_RMSE, "\n", sep = "")
    local_RMSE_elas <- c(local_RMSE_elas,
                         local_RMSE)
    cat("\n")
  }
  repeat_RMSE_naive <- c(repeat_RMSE_naive,
                         mean(local_RMSE_naive))
  repeat_RMSE_pca <- c(repeat_RMSE_pca,
                       mean(local_RMSE_pca))
  repeat_RMSE_kpca <- c(repeat_RMSE_kpca,
                        mean(local_RMSE_kpca))
  repeat_RMSE_elas <- c(repeat_RMSE_elas,
                        mean(local_RMSE_elas))
}
master_comparison_df <- data.frame(
  naive = repeat_RMSE_naive,
  pca = repeat_RMSE_pca,
  kpca = repeat_RMSE_kpca,
  elas = repeat_RMSE_elas
)
master_comparison_df

## KPCA with different polynomial kernel degrees
set.seed(12345678)
N <- nrow(protein)
K <- 10

deg_names <- paste("deg=", 2:10, sep = "")
kpca_deg_summary <- data.frame()
for (k in deg_names) kpca_deg_summary[[k]] <-
```

```
  as.numeric()
for (run in 1:3){
  cat(run, "-th CV Run\n", sep = "")
  validSetSplits <- sample((1:N)% %K + 1)
  run_summary <- c()
  for (deg in 2:10){
    local_RMSE <- kpca_degree_run(validSetSplits,
                                  deg)
    run_summary <- c(run_summary,
                     local_RMSE)
  }
  cat("\n")
  kpca_deg_summary <- rbind(kpca_deg_summary,
                            run_summary)
}

## KPCA method with elastic net regression
set.seed(20201126)
K <- 10
validSetSplits <- sample((1:N)% %K + 1)
for (k in 1:K){
  validSet <- protein[validSetSplits==k,]
  trainSet <- protein[validSetSplits!=k,]
  exp_var_pair <- kpca_transform(trainSet[,-1],
                                 validSet[,-1])
  kpca_train <- cbind(trainSet[,1],
  exp_var_pair[["kpca_train"]])
  names(kpca_train) <- c("accuracy",
                         names(kpca_train)[-1])
  kpca_test <- cbind(validSet[,1],
  exp_var_pair[["kpca_test"]])
  names(kpca_test) <- c("accuracy",
                        names(kpca_test)[-1])
  full <- lm(accuracy ~ ., data = kpca_train)

  elas_summ <- elas_net_summary(kpca_train,
                                kpca_test,
                                S = 10)
  RMSE_b <- min(elas_summ$RMSPE)

  pred_f <- predict(full, newdata = kpca_test)
  RMSE_f <- RMSE(kpca_test$accuracy, pred_f)
  cat(k, "th run:")
  cat("kpca d2 RMSPE =", RMSE_f, "\n")
  cat("kpca elas RMSPE =", RMSE_b, "\n")
```

```r
  cat("\n")
}

## KPCA Method with different cutoffs threshold (instead of 90%)
set.seed(123456)
N <- nrow(protein)
K <- 10

deg_names <- paste("cut=",
                   seq(70, 90, by = 5), sep = "")
kpca_cut_summary <- data.frame()
for (k in deg_names) kpca_cut_summary[[k]] <-
  as.numeric()
for (run in 1:3){
  cat(run, "-th CV Run\n", sep = "")
  validSetSplits <- sample((1:N)% %K + 1)
  run_summary <- c()
  for (x in seq(70, 90, by = 5)){
    cat("cutoff =", x, "\n")
    local_RMSE <- kpca_degree_run(validSetSplits,
                                 deg = 2,
                                 x = x)
    run_summary <- c(run_summary,
                     local_RMSE)
  }
  cat("\n")
  kpca_deg_summary <- rbind(kpca_deg_summary,
                            run_summary)
}

## Test on whether Box-Cox transformation can improve RMSPE on basic MLR and
    KPCA methods
cl <- makePSOCKcluster(8)
registerDoParallel(cl)

start.time <- proc.time()
set.seed(2020)
N <- nrow(protein)
K <- 10
repeat_RMSE_naive <- c()
repeat_RMSE_kpca <- c()
for (run in 1:10){
  local_RMSE_naive <- c()
  local_RMSE_kpca <- c()
  validSetSplits <- sample((1:N)% %K + 1)
```

```
for (k in 1:K){
  cat(run, ":", k, "-th run\n", sep = "")
  validSet <- protein[validSetSplits==k,]
  trainSet <- protein[validSetSplits!=k,]

  ## naive
  exclude_col <- get_exclude_col(trainSet)
  exclude_col <- get_vif_exclude_col(
    trainSet[,-which(names(trainSet) %in%
                       exclude_col)],
    exclude_col)
  trainSet <- trainSet[, -which(names(trainSet)
  %in% exclude_col)]
  validSet <- validSet[, -which(names(validSet)
  %in% exclude_col)]
  print("cleaned train set")
  model_naive <- lm(accuracy ~ ., data = trainSet)
  lambda <- boxcox_lambda(model_naive)
  accuracy_lambda <- boxcox_transfrom(
    lambda,
    trainSet$accuracy)
  model_naive <- lm(accuracy_lambda ~ .,
                        data = trainSet[, -1])
  pred_naive <- predict(model_naive,
                      newdata = validSet)
  pred_naive <- boxcox_transfrom_inv(lambda,
                                   pred_naive)
  local_RMSE <- sqrt(mean((validSet$accuracy -
  pred_naive)^2))
  local_RMSE_naive <- c(local_RMSE_naive,
                      local_RMSE)
  cat(run, ":", k, "-th run's naive RMSE=",
      local_RMSE, "\n", sep = "")

  ## kpca

  exp_var_pair <- kpca_transform(trainSet[,-1],
                               validSet[,-1])
  kpca_train <- cbind(trainSet[,1],
  exp_var_pair[["kpca_train"]])
  names(kpca_train) <- c("accuracy",
                       names(kpca_train)[-1])
  kpca_test <- cbind(validSet[,1],
  exp_var_pair[["kpca_test"]])
  names(kpca_test) <- c("accuracy",
                          names(kpca_test)[-1])
    model <- lm(accuracy ~ ., data = kpca_train)
    lambda <- boxcox_lambda(model)
    accuracy_lambda <- boxcox_transfrom(
      lambda,
      kpca_train$accuracy)
    model <- lm(accuracy_lambda ~ .,
                data = kpca_train[, -1])
    pred <- predict(model,
                          newdata = kpca_test)
    pred <- boxcox_transfrom_inv(lambda,
                                 pred)
    local_RMSE <- RMSE(kpca_test$accuracy, pred)
    local_RMSE_kpca <- c(local_RMSE_kpca,
                          local_RMSE)
    cat(run, ":", k, "-th run's kpca RMSE=",
        local_RMSE, "\n", sep = "")

    cat("\n")
  }
  repeat_RMSE_naive <- c(repeat_RMSE_naive,
                         mean(local_RMSE_naive))
  repeat_RMSE_kpca <- c(repeat_RMSE_kpca,
                         mean(local_RMSE_kpca))
}
stop.time <- proc.time()
run.time <- stop.time - start.time
print(run.time)
stopCluster(cl)

##############################
#      Model Selection       #
#     & Cross Validation     #
##############################

## Model selection on original feature space
cl <- makePSOCKcluster(8)
registerDoParallel(cl)
start.time <- proc.time()
set.seed(2020)
backward_AIC = c()
forward_AIC = c()
fb_AIC = c()
icm_AIC = c()
full <- lm(accuracy ~., data = protein_vif)
```

```
empty <- lm(accuracy ~1, data = protein_vif)

pen <- log(nrow(protein_vif)) #Manual change of lambda

# backward_model <- backward_selection(empty, full, pen)
forward_model <- forward_selection(empty, full, pen)
fb_model <- fb_selection(empty, full, pen)
icm_model <- icm_model_gen(protein_vif, pen)
stop.time <- proc.time()
run.time <- stop.time - start.time
print(run.time)
stopCluster(cl)

## Save generated models
saveRDS(forward_model, file = "forward_model.Rda")
saveRDS(fb_model, file = "fb_model.Rda")
saveRDS(icm_model, file = "icm_model.Rda")

## Five different lambda tuning cross-validation results (with 2 schemes)
fold_2_100_run = cv_run(2, 100) ## 2-Fold CV with 100 different samples
fold_200_1_run = cv_run(200, 1) ## 200-Fold CV with 1 sample

saveRDS(fold_200_1_run,
        file = "fold_200_1_run.Rda")
saveRDS(fold_2_100_run,
        file = "fold_2_100_run.Rda")

fold_2_100_run_1 = cv_run_ori(2,100)
fold_200_1_run_1 = cv_run_ori(200,1)

saveRDS(fold_2_100_run_1, file = "fold_2_100_run_1.Rda")
saveRDS(fold_200_1_run_1, file = "fold_200_1_run_1.Rda")

fold_2_100_run_2 = cv_run_ori(2,100)
fold_200_1_run_2 = cv_run_ori(200,1)

saveRDS(fold_2_100_run_2, file = "fold_2_100_run_2.Rda")
saveRDS(fold_200_1_run_2, file = "fold_200_1_run_2.Rda")

fold_2_100_run_3 = cv_run_ori(2,100)
fold_200_1_run_3 = cv_run_ori(200,1)

saveRDS(fold_2_100_run_3, file = "fold_2_100_run_3.Rda")
saveRDS(fold_200_1_run_3, file = "fold_200_1_run_3.Rda")

```

```
fold_2_100_run_4 = cv_run_ori(2,100)
fold_200_1_run_4 = cv_run_ori(200,1)

saveRDS(fold_2_100_run_4, file = "fold_2_100_run_4.Rda")
saveRDS(fold_200_1_run_4, file = "fold_200_1_run_4.Rda")

## Summary Boxplot of RMSPE across different run of lambda tuning

### First run with lambda = 0.1
par(mfrow = c(2,1),
    cex = 0.5)
RMSE_boxplot(fold_2_100_run$forward,
             fold_2_100_run$fb,
             fold_2_100_run$icm,
             fold_2_100_run$PCA,
             fold_2_100_run$KPCA2,
             fold_2_100_run$KPCA3,
             title = "2-Fold 100-Resample CV RMSPE")
RMSE_boxplot(fold_200_1_run$forward,
             fold_200_1_run$fb,
             fold_200_1_run$icm,
             fold_200_1_run$PCA,
             fold_200_1_run$KPCA2,
             fold_200_1_run$KPCA3,
             title = "200-Fold CV RMSPE")

### Second run with lambda = 0.5
par(mfrow = c(2,1),
    cex = 0.5)
RMSE_boxplot(fold_2_100_run_1$forward,
             fold_2_100_run_1$fb,
             fold_2_100_run_1$icm,
             fold_2_100_run$PCA,
             fold_2_100_run$KPCA2,
             fold_2_100_run$KPCA3,
             title = "2-Fold 100-Resample CV RMSPE")
RMSE_boxplot(fold_200_1_run_1$forward,
             fold_200_1_run_1$fb,
             fold_200_1_run_1$icm,
             fold_200_1_run$PCA,
             fold_200_1_run$KPCA2,
             fold_200_1_run$KPCA3,
             title = "200-Fold CV RMSPE")

### Third Run with lambda = 0.3
```

```
par(mfrow = c(2,1),
    cex = 0.5)
RMSE_boxplot(fold_2_100_run_2$forward,
             fold_2_100_run_2$fb,
             fold_2_100_run_2$icm,
             fold_2_100_run$PCA,
             fold_2_100_run$KPCA2,
             fold_2_100_run$KPCA3,
             title = "2-Fold 100-Resample CV RMSPE")
RMSE_boxplot(fold_200_1_run_2$forward,
             fold_200_1_run_2$fb,
             fold_200_1_run_2$icm,
             fold_200_1_run$PCA,
             fold_200_1_run$KPCA2,
             fold_200_1_run$KPCA3,
             title = "200-Fold CV RMSPE")

### Fourth run with lambda = 0.75
par(mfrow = c(2,1),
    cex = 0.5)
RMSE_boxplot(fold_2_100_run_3$forward,
             fold_2_100_run_3$fb,
             fold_2_100_run_3$icm,
             fold_2_100_run$PCA,
             fold_2_100_run$KPCA2,
             fold_2_100_run$KPCA3,
             title = "2-Fold 100-Resample CV RMSPE")
RMSE_boxplot(fold_200_1_run_3$forward,
             fold_200_1_run_3$fb,
             fold_200_1_run_3$icm,
             fold_200_1_run$PCA,
             fold_200_1_run$KPCA2,
             fold_200_1_run$KPCA3,
             title = "200-Fold CV RMSPE")

## Lambda tuning resulting RMSPE boxplots
par(mfrow = c(2,2),
    mar = c(4,4,4,4),
    cex = 0.5)
boxplot(fold_2_100_run$forward,
        fold_2_100_run_2$forward,
        fold_2_100_run_1$forward,
        fold_2_100_run_3$forward,
        fold_2_100_run_4$forward,
        horizontal = T,
        main = "2-Fold 100 Resamples CV RMSPE (Forward)",
        ylab = "Lambda",
        xlab = "RMSPE",
        pch = 20,
        col = "cyan",
        names = c("0.1",
                  "0.3",
                  "0.5",
                  "0.75",
                  "1.0"
                  ),
        ylim = c(0.45, 0.9))
boxplot(fold_200_1_run$forward,
        fold_200_1_run_2$forward,
        fold_200_1_run_1$forward,
        fold_200_1_run_3$forward,
        fold_2_100_run_4$forward,
        horizontal = T,
        main = "200-Fold CV RMSPE (Forward)",
        ylab = "Lambda",
        xlab = "RMSPE",
        pch = 20,
        col = "cyan",
        names = c("0.1",
                  "0.3",
                  "0.5",
                  "0.75",
                  "1.0"
                  ),
        ylim = c(0.1, 1.1))
boxplot(fold_2_100_run$fb,
        fold_2_100_run_2$fb,
        fold_2_100_run_1$fb,
        fold_2_100_run_3$fb,
        fold_2_100_run_4$fb,
        horizontal = T,
        main = "2-Fold 100 Resamples CV RMSPE (FB)",
        ylab = "Lambda",
        xlab = "RMSPE",
        pch = 20,
        col = "yellow",
        names = c("0.1",
                  "0.3",
                  "0.5",
                  "0.75",
                  "1.0"
                  ),
        ylim = c(0.45, 0.9))
boxplot(fold_200_1_run$fb,
        fold_200_1_run_2$fb,
        fold_200_1_run_1$fb,
        fold_200_1_run_3$fb,
        fold_200_1_run_4$fb,
        horizontal = T,
        main = "200-Fold CV RMSPE (FB)",
        ylab = "Lambda",
        xlab = "RMSPE",
        pch = 20,
        col = "yellow",
        names = c("0.1",
                  "0.3",
                  "0.5",
                  "0.75",
                  "1.0"
                  ),
        ylim = c(0.1, 1.1))


## Separate run to test whether we should consider degree 4 KPCA
cl <- makePSOCKcluster(8)
registerDoParallel(cl)
start.time <- proc.time()
K <- 100
N <- nrow(protein_vif)
run_total <- 1
kpca_RMSE_3 <- c()
kpca_RMSE_4 <- c()
set.seed(2020)
for (run in 1:run_total){
  validSetSplits <- sample((1:N)% %K + 1)
  local_kpca_RMSE_3 <-
    kpca_cv_fold(K,validSetSplits, deg = 3)
  local_kpca_RMSE_4 <-
    kpca_cv_fold(K,validSetSplits, deg = 4)
  kpca_RMSE_3 <- c(kpca_RMSE_3,
                   local_kpca_RMSE_3)
  kpca_RMSE_4 <- c(kpca_RMSE_4,
                   local_kpca_RMSE_4)
}
stop.time <- proc.time()
```

```r
run.time <- stop.time - start.time
print(run.time)
stopCluster(cl)

## Generate BCE transformed FB model
lambda = boxcox_lambda(fb_model)
accuracy_lambda <- boxcox_transform(lambda, protein_vif$accuracy)
fb_model_bc <- lm(accuracy_lambda ~. ,data = protein_vif[,which(colnames(
    protein_vif) %in% names(fb_model$coefficients))])

## Cross-validation between FB model and FB model with BCE
fold_2_100_bc = cv_bc_run(2, 100)
fold_200_1_bc = cv_bc_run(200, 1)

## RMSPE performance result between FB model and FB model with BCE
par(mfrow = c(2,1),
    mar = c(4,4,4,4),
    cex = 0.6)
boxplot(fold_2_100_bc$fb_RMSE,
        fold_2_100_bc$fb_RMSE_bc,
        horizontal = T,
        main = "2-Fold CV 100 Resamples RMSPE",
        xlab = "RMSPE",
        pch = 20,
        col = "orange",
        names = c("FB",
                  "FB with BCE"
                  ),
        ylim = c(0.1, 1.1))
boxplot(fold_200_1_bc$fb_RMSE,
        fold_200_1_bc$fb_RMSE_bc,
        horizontal = T,
        main = "200-Fold CV RMSPE",
        xlab = "RMSPE",
        pch = 20,
        col = "cyan",
        names = c("FB",
                  "FB with BCE"
                  ),
        ylim = c(0.1, 1.1))

##############################
#    CV Result Summaries     #
##############################

```

```r
## Summary of performance metrics across 3 candidate models
model_vec <- list(forward_model,
               fb_model,
               icm_model)
pred_num_vec <- c()
AIC_vec <- c()
BIC_vec <- c()
l0_vec <- c()
R_s_adj_vec <- c()
for (model in model_vec){
  local_pred_num = length(model[["coefficients"]])
  local_AIC = AIC(model)
  local_BIC = BIC(model)
  local_l0 = AIC(model,
                 k = 0.5 * log(nrow(protein)))
  local_R = summary(model)$adj.r.squared
  pred_num_vec <- c(pred_num_vec,
                    local_pred_num)
  AIC_vec <- c(AIC_vec,
               local_AIC)
  BIC_vec <- c(BIC_vec,
               local_BIC)
  l0_vec <- c(l0_vec,
              local_l0)
  R_s_adj_vec <- c(R_s_adj_vec,
                   local_R)
}
model_metric_df = data.frame(
  num_pred = pred_num_vec,
  AIC = AIC_vec,
  BIC =  BIC_vec,
  l0 = l0_vec,
  R_squared_adj = R_s_adj_vec
)
rownames(model_metric_df) <-
  c("Forward", "FB", "ICM")

## Generate model summary LateX table for FB model
fb_model.format = format(coef(summary(fb_model))[,c(1,2,4)],
                              digit = 3)
colnames(fb_model.format) = c("Estimate", "Standard Error", "p - value")
rownames(fb_model.format)[1] = "Intercept"
print(xtable(fb_model.format),
      booktabs=TRUE,
      title = "Final Model Parameter Estimates, Standard Error and p-values")
```

```

## Get features with 5 largest manitudes of coefficients estimate
top_five_coe = names(sort(abs(fb_model$coefficients), decreasing = T)[2:6])
fb_model$coefficients[names(fb_model$coefficients)%in% top_five_coe]

## FB model compared to FB model with angles
fold_2_100_angle = cv_angle_run(2, 100)
fold_200_1_angle = cv_angle_run(200, 1)
par(mfrow = c(2,1),
    mar = c(4,4,4,4),
    cex = 0.6)
boxplot(fold_2_100_angle$fb_RMSE,
        fold_2_100_angle$fb_RMSE_bc,
        horizontal = T,
        main = "2-Fold CV 100 Resamples RMSPE",
        xlab = "RMSPE",
        pch = 20,
        col = "orange",
        names = c("FB",
                  "FB with angle"
                  ),
        ylim = c(0.1, 1.1))
boxplot(fold_200_1_angle$fb_RMSE,
        fold_200_1_angle$fb_RMSE_bc,
        horizontal = T,
        main = "200-Fold CV RMSPE",
        xlab = "RMSPE",
        pch = 20,
        col = "cyan",
        names = c("FB",
                  "FB with angle"
                  ),
        ylim = c(0.1, 1.1))

## 95% confidence interval for estimated RMSPE
a = qt(0.975, 199)
rbar = mean(fold_2_100_angle$fb_RMSE)
sd = sd(fold_2_100_angle$fb_RMSE)
print(c(rbar - a * sd * sqrt(1+ (1 / 200)),
        rbar + a * sd * sqrt(1+ (1 / 200))))

## Leverage vs. Cook's distance plot
# Sigma
fb_sigma = summary(fb_model)$sigma

```

```r
# Residual
fb_res = resid(fb_model)

# Get Hat value
fb_hat = hatvalues(fb_model)

# Studentized residual
fb_stu_res = fb_res / fb_sigma / sqrt(1-fb_hat)

# Calculated the mean of the hat value
fb_hbar= mean (fb_hat)
# Calculated the cook's distance of both model and the mean
fb_cd = cooks.distance(fb_model)
fb_cd_mean = mean(fb_cd)

# Store the index where studentized distance is greater than 3
stu_ind = fb_stu_res > 3

# Colored yellow for those high leverage point
color_vec = rep("black",len = nrow(protein_vif))
color_vec[stu_ind] = "yellow"

# Store the index where hat value is greater than 2 time average of hat value
lev_ind = fb_hat > 2 * fb_hbar

# Colored blue for those high leverage point
color_vec[lev_ind] = "blue"

# Store the index where cook's distance is greater than 0.5
infl_ind = (fb_cd > 0.5)

# Colored red for those high cook's distance
color_vec[infl_ind] = "red"

# Plot the cook's distance vs leverage for both model
# with vertical line indicate 2*leverage average
plot(fb_hat,
     fb_cd,
     yaxt = "n",
     pch = 21,
     bg = color_vec,
     main = "Cook's Distance vs. Leverage Plot",
     xlab = "Leverage",
     ylab = "Cook's Distance")
axis(2, at=seq(0, 0.06, 0.02))
```

```

legend("topleft",
       legend = c("Cook's Distance > 0.5",
                  "Leverage > 2 * mean(Leverage)",
                  "Studentized Residual > 3",
                  "2 * mean(Leverage)"),
       col = c("red","blue","yellow",
               "grey60"),
       cex = 0.9,
       pch = c(19,19,19,NA),
       lty = c(NA,NA,NA,2),
       bty ="n")

abline(v = 2*fb_hbar,col = "grey60", lty =2)

## Occurrence count of atom/length types
name_list = names(fb_model$coefficients)
total_list <- c()
for (name in name_list){
    total_list <- c(total_list,
                    strsplit(name, "_")[[1]])
}
summ_table = table(total_list)
barplot(sort(summ_table),
        main = "Occurrence Count of Atom/Length Types",
        las=2,
        ylim = c(0, 50),
        ylab = "Frequency")

##############################
#      Final Prediction      #
##############################
test_set = read.csv("protein-test.csv")
final_pred = predict(fb_model,
                     newdata = test_set)
writeLines(as.character(final_pred), "final_pred.txt")
```

## REFERENCE

[1]R. Rosipal, M. Girolami, L. J. Trejo, and A. Cichocki, “Kernel pca for feature extraction and de-noising in nonlinear regression,” Neural Computing & Applications **10**, 231–243 (2001).