Norwegian University
of Life Sciences

Master's Thesis 2022 30 ECTS
Faculty of Science and Technology

Exploration of usability of PLSR for implementation in the RENT feature selection method

Rameesha Asghar Khan
MSc Data Science

Norwegian University
of Life Sciences

Faculty of Science and Technology

Exploration of usability of PLSR for implementation in the RENT feature selection method

Master's Thesis in Data Science
by

Rameesha Asghar Khan

Internal Supervisors

Oliver Tomic

December 15, 2022

"Man's mind once stretched by a new idea, never regains its original dimension."

Oliver Wendell Holmes

Abstract

In machine learning, when the objective is classification or object detection, feature selection provides a significant advantage to obtaining optimal results. Most feature selection heuristics (methods derived from past work done on related problems) available in literature focus mainly on the predictive performance of the algorithms. Stability of the selection of feature selection is an important aspect which is not always on the forefront of feature selection process.

RENT (Repeated Elastic Net Technique) is a fairly new method developed for feature selection which works only for binary classification and regression problems. It uses logistic and linear regression and focuses on stability along with predictive performance. This thesis explores PLSR (Partial Least Squares Regression) algorithm to extend RENT to handle multi-class classification and regression problems.

PLSR-RENT uses cumulative validated explained variance of individual features for feature selection. The selected feature subset is used with different classifiers and the results obtained are compared first with baseline classifiers with original features set and then with feature subsets obtained using other feature selection methods.

The results obtained show that while the performance scores of PLSR-RENT does not exceed other feature selection methods, the difference is but marginal. PLSR-RENT does provide slight improvement in comparison to baseline model for some classification algorithms.

We make the argument that if with slight drop in performance score, the computational cost can drop significantly, a user can make the trade-off between performance scores and number of features (and thus computation cost) while using PLSR-RENT feature selection method.

Acknowledgements

First, I would like to thank my supervisor Oliver Tomic from NMBU for his valuable guidance, feedback and imparted knowledge throughout the whole thesis process. His support and understanding for my circumstances as a mother of young children was essential and highly appreciated.

I would also like to thank my co-supervisor Cecilia Futsæther for her support.

Finally, huge amounts of thanks goes to my family for their patience and understanding, and being the best, free of charge baby sitters while I was working on this thesis.

Contents

Abstract v

Acknowledgements vii

Abbreviations xiii

1 Introduction 1
- 1.1 Motivation . . . 3
- 1.2 Problem Definition . . . 4
- 1.3 Thesis Outline . . . 4

2 Materials and Methods 5
- 2.1 Feature Selection . . . 6
  - 2.1.1 Filter Methods . . . 6
  - 2.1.2 Wrapper Methods . . . 7
  - 2.1.3 Embedded Methods . . . 8
- 2.2 Repeated Elastic Net Technique . . . 9
- 2.3 Partial Least Squares Regression . . . 12
  - 2.3.1 PLSR model equations . . . 12
  - 2.3.2 Explained Variance . . . 13
  - 2.3.3 Cumulative Explained Variance . . . 15
  - 2.3.4 Number of components . . . 16
  - 2.3.5 PLS1/PLS2 . . . 17
  - 2.3.6 PLSR for classification . . . 17
- 2.4 PLSR RENT . . . 18
  - 2.4.1 PLSR-RENT Methodology . . . 18
  - 2.4.2 Comparison methods . . . 20
- 2.5 Model Validation . . . 21
  - 2.5.1 K-fold . . . 22
  - 2.5.2 Stratified K-Fold . . . 22
  - 2.5.3 Repeated Stratified K-Fold . . . 23
- 2.6 Multi-Class Classifiers . . . 24
  - 2.6.1 Multinomial Logistic Regression . . . 24
  - 2.6.2 Support Vector Machines . . . 26
  - 2.6.3 Random Forest . . . 28

2.6.4 K-Nearest Neighbors . . . 29
2.7 Performance Metrics . . . 31
2.7.1 Accuracy . . . 32
2.7.2 Balanced Accuracy . . . 32
2.7.3 Precision . . . 33
2.7.4 Recall . . . 33
2.7.5 F1 score . . . 34
2.7.6 Cohen's Kappa Score . . . 34
2.7.7 Mathews Correlation Coefficient (MCC) . . . 35

3 Experimental Setup **37**
3.1 Software . . . 37
3.2 Workflow . . . 38
3.3 Data . . . 38
3.4 Data Preprocessing . . . 39
3.4.1 Handling Missing and Duplicate Data . . . 39
3.4.2 One Hot Encoding . . . 40
3.4.3 Low variance threshold . . . 40
3.5 Feature Selection . . . 40
3.5.1 PLSR-RENT . . . 41
3.5.2 SelectKBest . . . 41
3.5.3 ExtraTrees Feature Importance . . . 42
3.6 Classification . . . 42
3.6.1 Baseline model pipeline . . . 43
3.6.2 PLSR-RENT model pipeline . . . 43
3.6.3 FS1 model pipeline . . . 43
3.6.4 FS2 model pipeline . . . 43
3.7 Hyperparameter Tuning . . . 44
3.8 Results . . . 45

4 Results and Discussion **47**
4.1 Data Preprocessing . . . 47
4.2 PLSR-RENT . . . 48
4.3 Feature Selection . . . 54
4.4 Hyperparameter Tuning . . . 54
4.5 Classification . . . 55
4.5.1 Comparison with Baseline . . . 55
4.5.2 Comparison with other Feature Selectors . . . 57

5 Conclusion and Future Directions **61**
5.1 Conclusion . . . 61
5.2 Future Work . . . 62

List of Figures **63**

List of Tables **67**

A Complete Results of the Experimental Setup **73**

B Results for Dataset MNIST **79**
B.1 Data 79
B.2 Results 79

Bibliography **81**

Abbreviations

FS	**F**eature **S**election
RENT	**R**epeated **E**lastic **N**et **T**echnique
PLSR	**P**artial **L**east **S**quares **R**egression
PLSR-RENT	**PLSR RENT**
PC	**P**rincipal **C**omponent

Chapter 1

Introduction

Technology... the knack of so arranging the world
that we don't have to experience it.
— Max Frisch

High-dimensional data is when the number of features or variables p is greater than the number of instances or observations n; i.e. $p > n$. The proliferation of big data would have us believe that more data is better; which, while true, comes with its own set of statistical challenges of computations and visualization. "Visual discovery" acquired by plots and diagrams becomes unfeasible to carry out because of the large number of variables [1]. Covariance matrix, which is pertinent in applications in many areas such as financial market, genetic network and climatology, becomes challenging to accurately estimate for high dimensional data [2]. [2] also refers to the problem of selecting statistically significant features based on very small sample size, by giving examples of tumor classification based on hundreds and thousands of gene expressions.

The term *curse of dimensionality* was first introduced by [3]. It refers to a set of problems that are faced when working with high-dimensional data. High-dimensionality gives rise to overfitting. Given a fixed number of samples, if we add more features, the data becomes sparse and the performance of a classifier is going to start to deteriorate after reaching an optimal number of features, as shown in figure 1.1.

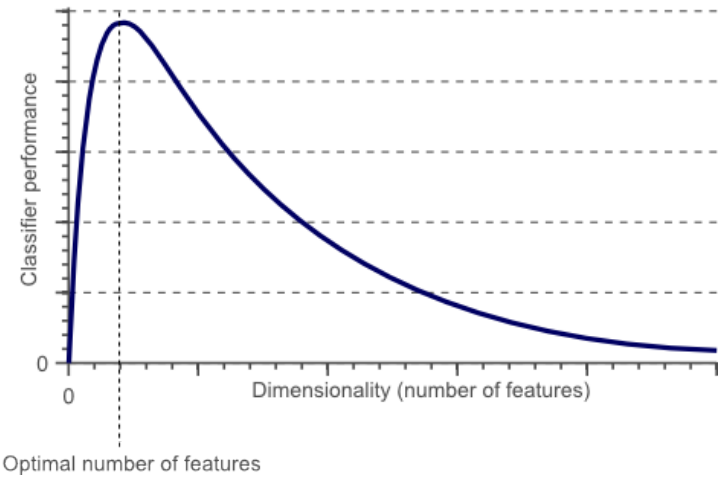


Figure 1.1: Trend of classifiers performance with rising dimensionality. The performance of the classifier increases with increase in dimensionality until it reaches the optimal number of features. After that increasing the dimensionality (without adding samples) only results in decrease in performance of the classifier [4].

These challenges and issues give rise to the need for dimensionality reduction techniques which help us acquire a subset of important features out of the full set of features available.

The knowledge of important features is crucial in optimizing any algorithm that deals with classification or regression issues. The question of why feature selection is an important preprocessing step can be answered precisely by the following three main points:

1. **Computation cost:** The main and most obvious benefit of using feature selection is it decreases the computation time while training the model. The fewer features are present in the data, the quicker the model is trained.

2. **Improves performance:** By getting rid of unnecessary and misleading features the performance of the model may improve. It is ideal to gain improved performance with reduced number of features, but sometimes trading off a bit of performance for significant number of features is also a feasible option.

3. **Decreases over-fitting:** Redundancy may lead to over-fitting. Feature selection handles this problem by getting rid of or reducing the number of redundant features and amount of noise which might lead to over-fitting.

In this thesis we will be exploring a new method for the purpose of feature selection by the use of PLSR algorithm: PLSR-RENT. We will be comparing the performances of our method with the performances of multiple other algorithms capable of handling multi-class problems on full set of features and multiple sets of reduced features.

1.1 Motivation

In the real life applications in the field of bio-metrics, genome study or other scientific studies, the data collected is normally high-dimensional with a higher number of variables than observations.

Feature selection methods are created to help with narrowing down the features to a subset that is relevant for solving a given problem of classification or regression. In the literature there are many heuristics and feature selection methods discussed which are data or application specific, for example [5]; [6]. Conversely, robustness and stability of these feature selection methods is always being questioned. A *stable* process is where a small change in the input does not lead to large change in the output. In the area of feature selection, that means that changes in the input or training data (by adding or removing samples) does not lead to vastly different set of features selected.

Kalousis et al. states the following in his 2005 study: *"We define the stability of a feature selection algorithm as the sensitivity of the feature preferences it produces to differences in training sets drawn from the same generating distribution [..]. Stability quantifies how different training sets affect the feature preferences."*

There are a variety of works that discuss the *stability* of the feature selection methods. [8] review different feature selection techniques, their instability, and their proposed solution to it. [7] evaluates the stability of feature selection methods with use of ranks, weights, scores, etc. Whereas [9] analyses the insatiability problems feature selection algorithms face in high dimensional data.

Recently, Jenul et al. have developed a new method for feature selection called RENT. It is a feature selection technique which provides information on feature selection stability by showing how often a feature is selected when models are trained on subsets of the

training data. This method uses elastic net regularization and an ensemble of linear models for binary classification and regression [10] problems.

1.2 Problem Definition

RENT is a technique which is built to select important features. Although, the method is still relatively new, it is proving to be a competent technique which is also stable. The limitation which it faces is that it only works for binary classification and regression.

Many real world problems are actually multi-class problems; which presents the need to extend RENT so it can handle multi-class problems as well as binary problems. This thesis delves into the next phase of RENT: multi-class classification.

Often times in multivariate analysis we have to work with high-dimensional multi-colinear data which has high covariance between features. PLSR algorithm deals with these issues efficiently and gives us optimal results. This is the reason we will investigate the use of PLSR algorithm to extend RENT to include multi-class classification and regression.

1.3 Thesis Outline

This thesis follows the following structure:

Chapter 1 defines the need for feature selection and the challenges and problems of stability while selecting features; especially in multi-class case.

Chapter 2 outlines the related materials and methods used in the thesis. We delve briefly into the RENT methodology and following it how we develop PLSR-RENT method to extend it.

Chapter 3 will focus on the experimental setup for PLSR-RENT and methodology and workflow of the validation techniques.

Chapter 4 is where the results of our experimental work is presented with figures and tables. The results are also briefly discussed.

Chapter 5 presents the conclusion and points out the areas for future work and development.

Chapter 2

Materials and Methods

If it keeps up, man will atrophy all his limbs
but the push-button finger.
— Frank Lloyd Wright

Dimension reduction techniques are loosely divided into feature extraction or feature selection. There are many different feature extraction and feature selection methods available. Some of the most common techniques are shown in figure 2.1.

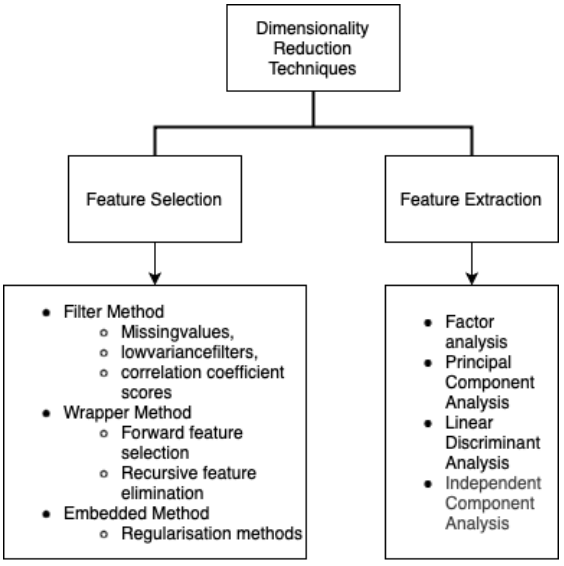


Figure 2.1: Common dimensionality reduction techniques.

Feature extraction methods extract information from the given features and then transforms them into new features with lower dimensions (feature engineering). These new features pack the information of all the features. Whereas, feature selection is simply selection of a subset of informative features from the original set of features.

In this chapter we will outline different feature selection methods, discuss the theory of RENT feature selection method, our approach using PLSR and other related work.

2.1 Feature Selection

Feature selection is an essential step of data pre-processing in machine learning algorithms. An efficient feature selection algorithm can massively improve the performance and computational cost of a machine learning algorithm. Exhaustive research has been done to study the different methodology and techniques for feature selection. A good feature selection method is the one which facilitates decreased complexity, improved performance, and stability of the features selected.

As mentioned in the figure 2.1, there are numerous feature selection techniques. Different authors and papers have divided these techniques based on different criteria. Feature selection techniques are, most commonly, split into three types: (i) filter methods, (ii) wrapper methods, and (iii) embedded methods, as previously shown in figure 2.1.

2.1.1 Filter Methods

Filter methods are supervised feature selection methods which use the simple innate statistical attributes of the data in relation to the target to select features. These methods are also called *univariate* and as can be inferred from the name each feature is assessed and ranked independent of its relation to other features - just to the target. This can be a disadvantage as some features when assessed independently might be weak but in combination with some other feature(s), might prove to be valuable.

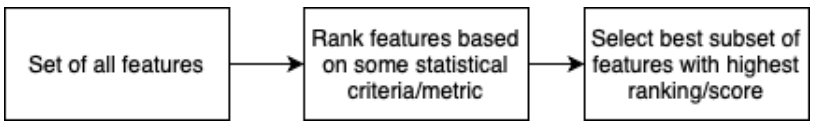


Figure 2.2: Working of filter methods

Statistical measures such as correlation, chi squared, distance, mutual information are used in the process of decision making of filter methods. Filter methods are easy to implement and interpret, and faster to compute.

Missing Value Ratio

When a feature is missing a lot of data points, it does not provide any useful information. In missing value ratio method we set a threshold of the ratio of missing values allowed; then we can remove features accordingly.

Low Variance Filter

Features whose values are either constant or do not change much are said to have low variance and thus do not provide useful information about the attribute which might differentiate it from others. Variance (how much the values change) of each feature is calculated and then based on a given threshold some features are removed in the low variance filter method.

High Correlation Filter

When two independent variables are very similar to each other, they are said to be highly correlated. In this method correlation coefficient is calculated and based on a given threshold, variables with values higher are dropped.

2.1.2 Wrapper Methods

Wrapper methods are also called *greedy* algorithms, as they search through almost every possible combination of features to find the subset of features which will give the best performance score. It is a time-consuming but can give better results.

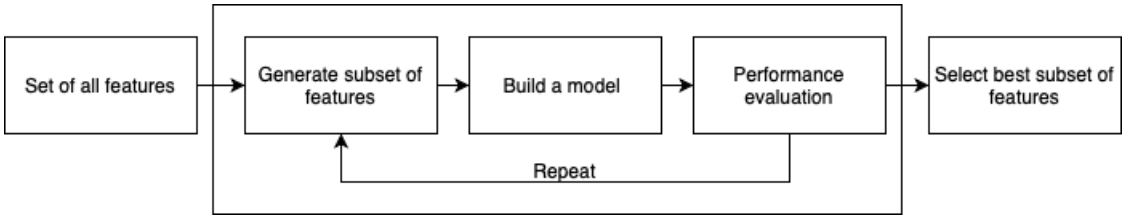


Figure 2.3: Working of wrapper methods

Wrapper methods are based off of a particular classification or regression algorithm as it evaluates the performance and selects features that are best suited to that algorithm. The same features might perform poorly when another algorithm is used.

Forward Feature Selection

In forward feature selection, we start off by selecting one feature and then adding other features one at a time. First, one feature is tested in combination with all other features and the pair with best performance is chosen. Afterwords, we test that pair in combination with the rest of the features and the best is selected. Similarly, the process goes on until the specified number of features is selected.

Backwards Feature Selection

This method starts off by full set of the features and, recursively subtracts all features once in separate subsets. The subset with best performance is selected. From that subset, once again, every remaining feature is subtracted in different subsets and the performance is evaluated. And thus the process repeats until we reach a user specified number of best features.

Recursive Feature Elimination

Recursive Feature Elimination(RFE) is similar to backwards selection. Both start off with full set of features and then eventually subtracts one feature per iteration. The difference is how the next feature that is to be eliminated is selected. In backwards elimination we base our decision on the performance of the model(the best performing subset is selected, discarding the rest). Whereas, in RFE the next feature to be eliminated is based off of feature importance calculated by the model used in the process.

2.1.3 Embedded Methods

Embedded methods find a middle ground between the computationally costly wrapper methods and very basic but computationally cost effective filter methods. These methods incorporate the low computational cost along with the features relations with each other. These methods iteratively search for best features, and select best feature in each iteration. Some of the most common embedded feature selection methods are: regularization methods, and decision trees feature importance method.

Regularization methods

High-dimensional data has more chances of noise in it. Having noise in the data makes the model susceptible to overfitting as we try to accommodate the increased number of features and thus complexity. One solution to it might be simplifying the model; which might lead to underfitting. The ideal situation is to find a balance between overfitting or underfitting a model. Regularization techniques accomplish this balance by introducing penalty. Penalizing different coefficient terms hinders the inclination of some terms to fit to the noise. The higher the penalty, the less chances are for overfitting. Penalty shrinks the coefficients of some terms, sometimes to zero. The terms with zero coefficient values can then be discarded. Regularization methods are lasso(L1 regularization) or elastic nets(L1 plus L2 regularization).

Decision trees feature importance method

Some algorithms, such as Decision Tree, can rank the features based on how much it affects the model prediction while making a decision in the process of training the model. Using these feature importance ranking we can remove the features with low rankings.

Random forests and extra trees are ensembles of decision trees and are popular for feature selection also.

2.2 Repeated Elastic Net Technique

Repeated Elastic Net Technique (RENT) is a feature selection method which focuses on the stability of the features selected, along with the performance of the algorithm. It works by training multiple linear models on different subsets of training data. Figure 2.4 shows the RENT pipeline as explained in [10]. RENT works for binary classification and regularization problems only.

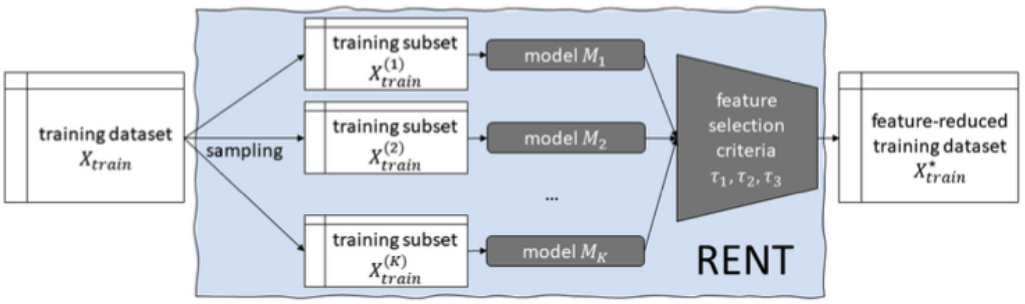


Figure 2.4: RENT pipeline as explained in [10]

RENT is an adaptation of the idea of ensembles of models in [11]. In RENT, we have an ensemble of linear models and the training data is sampled into multiple subsets; one model is trained on each subset of the training data.

RENT uses elastic net regularization. Elastic net is an embedded method for feature selection where features are selected during the model training process. It sets the weight of the features that seem important to non-zero, and the unimportant ones to zero. The results are then saved to a table with the rows representing the weights vectors and the columns representing features. Figure 2.5 visualises the calculation of the results table containing the weights of the features across different data splits.

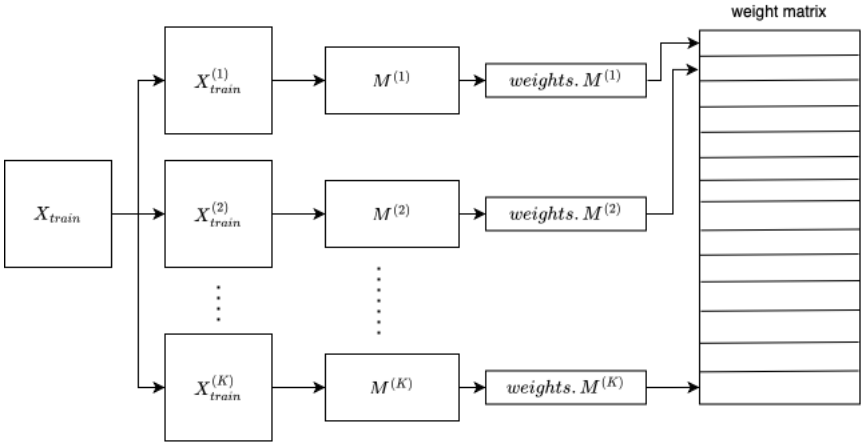
Using the results from the weight table we take three criteria into consideration (τ_1, τ_2, τ_3) when selecting the final features.

1. τ_1 is the frequency of selection of the features. This corresponds to the feature having a non-zero weight. It is calculated by the following equation:

$$\tau_1(\beta_n) = c(\beta_n) = \frac{1}{K} \sum_{k=1}^{K} 1[\beta_{k,n} \neq 0] \tag{2.1}$$

   The information about the feature relevance is stored in $\beta_n = (\beta_{1,n}, ..., \beta_{K,n})$. τ_1 must be relatively high for the feature to get selected.

2. τ_2 calculates the proportion of the feature weights taking the same sign, either positive or negative. It is calculated using equation 2.2. The maximum value τ_2 can take is the value of τ_1.



	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	...	F(N)
K(1)									
K(2)									
K(3)									
⋮									
K(K)									

Figure 2.5: Calculation of weight matrix in RENT with the use of elastic net technique.

When $\tau_2 == \tau_1$ it means that all non-zero weights have the same sign, either all positive or all negative. If τ_2 is smaller than τ_1 it indicates that some of the non-zero weights for this feature have different signs.

$$\tau_2(\beta_n) = \frac{1}{K}|\sum_{k=1}^{K} sign\beta_{k,n}| \tag{2.2}$$

3. τ_3 finds the deviation of the feature weights from zero.

$$\tau_3(\beta_n) = t_{K-1}\left(\frac{|\mu(\beta_n)|}{\sqrt{\frac{\sigma^2(\beta_n)}{K}}}\right) \tag{2.3}$$

t_{K-1} is Student's cumulative distribution function with $K-1$ degrees of freedom. μ and σ are mean and variance, respectively.

All three criteria τ_1, τ_2, and τ_3 must all be above some user defined threshold of t_1, t_2, and t_3 to be selected by the RENT algorithm.

2.3 Partial Least Squares Regression

Partial Least Squares Regression (PLSR) is an algorithm for regression problems for high dimensional data that may contain highly correlated features. PLSR can also be used for classification after the target (response) is one-hot encoded.

PLSR algorithm makes use of the covariance of the features to reduce dimension by building smaller set of uncorrelated components.

2.3.1 PLSR model equations

PLSR is very closely related to PCA (Principal Component Analysis). Both algorithms are capable of reducing the dimensionality of the data but with some pivotal differences. Where PCA uses variance between the independent features, PLSR uses covariance.

PCA only gives weight to the relation between the independent variables X with no regard to relevancy to Y. Performing PCA on X decomposes it into scores matrix T and loadings matrix P, which are both orthogonal to each other. Equation 2.4 shows the decomposition of X; where E is the residuals.

$$X = TP' + E \tag{2.4}$$

PLSR, on the other hand, also weighs in the relation of the independent variables X to the target variable Y. Performing PLSR decomposes both X and Y into equations 2.4 and 2.5. U and Q are orthogonal matrices of scores and loadings of Y, accordingly. F is residuals.

$$Y = UQ' + F \tag{2.5}$$

According to [12] equations 2.4 and 2.5 are called *outer relations* which help to find the *inner relation* as shown in equation 2.6. PLSR finds an optimal solution where T has maximum covariance with U. Figure 2.6 summarizes the conceptual working of PLSR.

$$U = \beta * T \tag{2.6}$$

β is the *regression vector* calculated using the equation 2.7, which can be used to calculate the prediction value y as in equation 2.8.

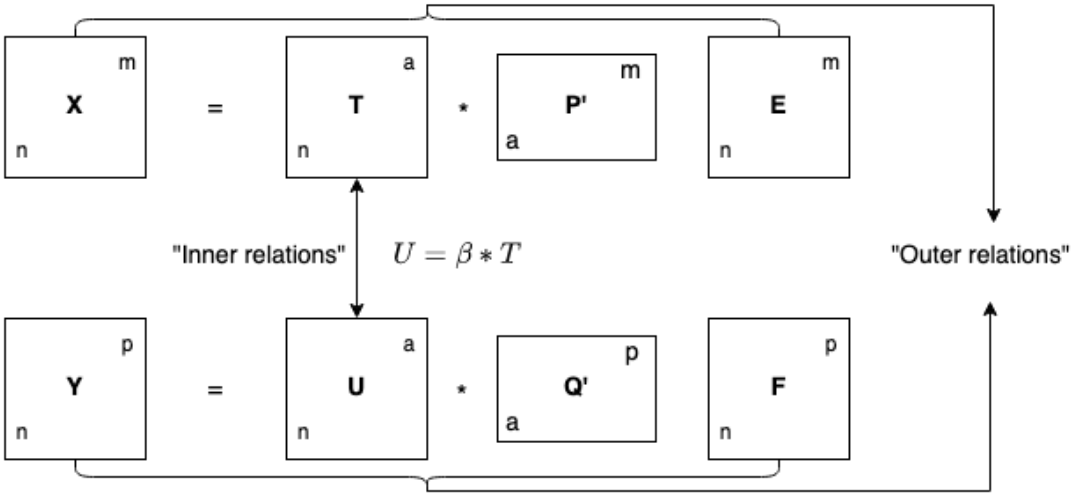


Figure 2.6: The underlying general working of PLSR. The breakdown of X and Y matrices to scores and loadings make up the *outer relations* which are used to calculate *inner relation* that is used to predict Y

$$\beta = (T^T T)^{-1} T^T y \tag{2.7}$$

$$T * \beta = y \tag{2.8}$$

Equation 2.8 shows that in PLSR to predict y we use only the scores instead of the whole X. This makes the model more robust and impervious to small changes in the data X

2.3.2 Explained Variance

PLSR creates principal components(eigenvectors) and each component has a different degree to which they attribute to the variance in the data. **Explained variance** *explains* how much of the total variation is caused by each principal component [13]. Explained variance allows us to rate the components in accordance to their importance. The more important the component is, the higher the explained variance.

Explained variance is calculated as the ratio between a particular eigenvalue and sum of all eigenvalues across all eigenvectors. Given a number of N eigenvectors (principal

components), the explained variance can be represented as in equation 2.9, where λ_i is the eigenvalue.

$$ExpVar = \frac{\lambda_i}{\lambda_1 + \lambda_2 + ... + \lambda_n} \tag{2.9}$$

Consider a dataset with n features $X_j, j = 1, 2, ..., n$ in $\Re^m$. This data set can be represented as a $m \times n$ matrix of $X = [X_1, \ldots, X_n]$, where each X_j is a n-vector.

Explained variance in X for component j for all features is calculated by the equation 2.10

$$ExpVar\_X_j = \frac{SS(\hat{X}_j)}{SS(X)} = \frac{tr(\hat{X_j^T}\hat{X}_j)}{tr(X^T X)} \tag{2.10}$$

where SS is the sum of squares, tr is trace of matrix, and $\hat{X}_j$ is the vector of averages of X_j

Similarly, explained variance in Y for all features for component j is calculated by following equation:

$$ExpVar\_Y_j = \frac{SS(\hat{Y}_j)}{SS(Y)} = \frac{tr(\hat{Y_j^T}\hat{Y}_j)}{tr(Y^T Y)} \tag{2.11}$$

Explained variance for each individual feature in X and Y is calculated by the equations:

$$ExpVarX\_indVar = \frac{diag(\hat{X_j^T}\hat{X}_j)}{diag(X^T X)} \tag{2.12}$$

$$ExpVarY\_indVar = \frac{diag(\hat{Y_j^T}\hat{Y}_j)}{diag(Y^T Y)} \tag{2.13}$$

Consider an example with 10 principal components in figure 2.7. The figure shows the list of explained variance ratio of all principal components. Explained variance or individual explained variance is the variation of each individual component.

```
1 pca.explained_variance_ratio_
```

```
array([3.21371891e-01, 2.37089839e-01, 1.55600315e-01, 1.05351043e-01,
6.92858779e-02, 5.47049848e-02, 3.52056608e-02, 1.45935070e-02,
6.79688170e-03, 4.05252471e-33])
```

Figure 2.7: Explained variance ratio of 10 principal components

2.3.3 Cumulative Explained Variance

Cumulative explained variance is the cumulative sum of the explained variance ratio. In the figure 2.8 the first principal component (PC1) explains 32% of the variation, PC1 and PC2 combined explains 55%, 5 PCs together explain 88% of the data variance. We can see that out of the 10 PCs calculated, using only 7-8 PCs can explain the most of data variance; making the last few PCs almost redundant.

```
1 pca.explained_variance_ratio_.cumsum()
```

```
 array([0.32137189, 0.55846173, 0.71406204, 0.81941309, 0.88869897,
0.94340395, 0.97860961, 0.99320312, 1. , 1. ])
```

Figure 2.8: Cumulative explained variance ratio of 10 principal components

Cumulative explained variance of individual components is the cumulative sum of the individual explained variance across components. In the example in 2.9 the row 1 is individual explained variance of PC1, the row 2 contains the cumulative explained variance of PC1 and PC2, and so on.

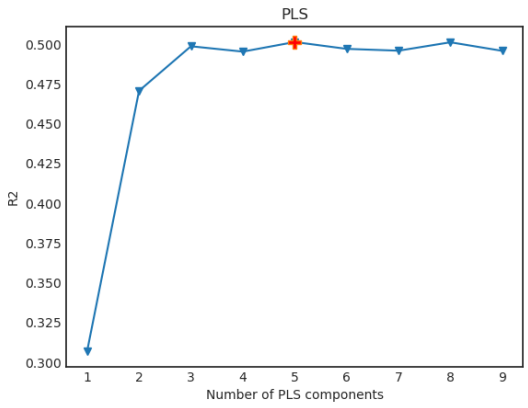
	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	-23.571063	-45.590811	60.715277	70.303332	-0.922528	8.194662	-29.769323	39.626722	-51.961552
2	12.501434	-18.423448	55.935909	86.757854	-15.410352	4.421133	23.186006	32.020137	4.472035
3	61.423435	48.827984	75.563279	87.347264	-13.932984	16.137024	69.776542	27.743225	17.119691
4	68.151156	56.705117	78.186492	88.140066	1.146029	21.493524	78.435899	57.340166	31.303569
5	77.939792	60.618817	80.120469	82.822562	55.223916	21.730589	94.724259	52.812227	57.299265
6	88.122022	73.081127	82.774772	80.761602	52.812248	54.445526	97.927361	70.872623	71.180876
7	96.374560	70.389436	86.231733	98.131495	47.925439	65.540128	99.146017	73.474530	74.540340
8	99.849524	99.455444	99.850854	99.424652	98.287802	97.324628	99.712693	99.655770	99.984927
9	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000

Figure 2.9: Individual cumulative explained variance ratio of 9 principal components

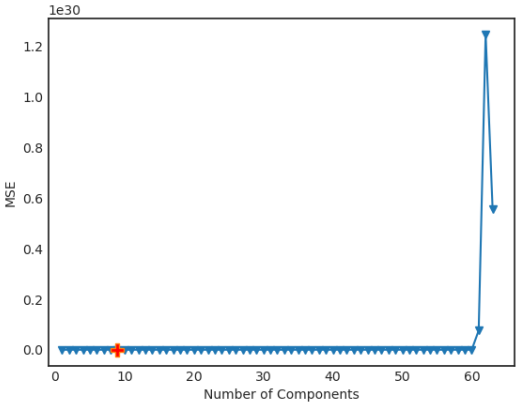
For dimension reduction PCA/PLSR uses explained variance to measure the importance of components. Explained variance can also get rid of redundant features. As in 2.8 we can see that only 9 features can explain 100% of the variance in the original data, leaving one feature redundant.

2.3.4 Number of components

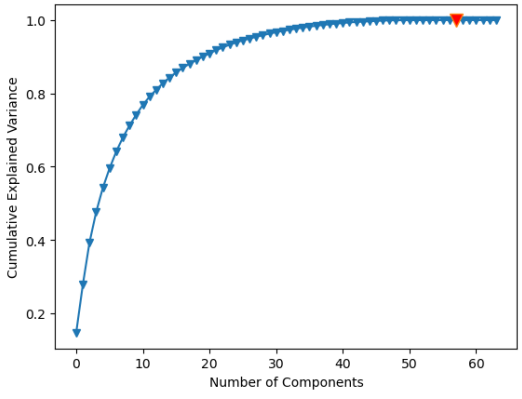
PLSR algorithm outputs new components but the number of components to chose is a choice we have to make. Covariance of the components with Y is the highest for the first component calculated, then the second and so on. We can use validation to decide on the number of components to be selected.



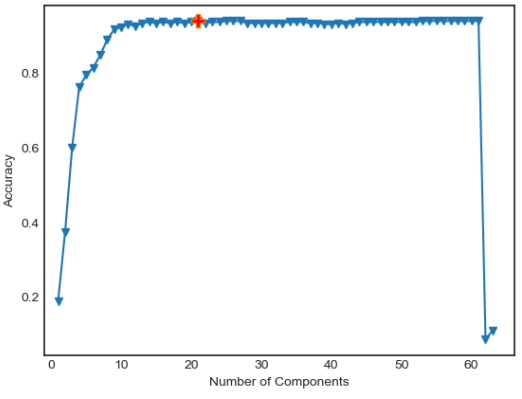
(a) Number of components according to maximum R^2



(b) Number of components according to minimum MSE



(c) Number of Components according to maximum cumulative explained variance



(d) Number of components according to maximum accuracy

Figure 2.10: Choosing number of components according to different criteria

[14] suggested the selection of PCs which will maximise the value of R^2. [15] concluded that mean squared error (MSE) would be potentially be a great criteria for component selection. [16] means that the small values of eigenvalues can improve score predictions. In addition to these we can also choose the number of components based on the maximum number of performance score, which is what we have used in our experimental setup. Figure 2.10 visualizes the different ways of choosing the number of components based on the above mentioned criteria.

2.3.5 PLS1/PLS2

PLSR can be categorized into PLS1 and PLS2. PLS1 is used when handling only one dependent variable. PLS2 is used when we are dealing with more than one dependent variables. In our thesis we are using PLS2 with one-hot encoded target as we are working on multi-class classification case.

2.3.6 PLSR for classification

PLSR is a regression algorithm, as the name suggests. But it can be used for classification also with some alterations. We encode the categorical target values using one-hot encoding and then pass it through the model. In one-hot encoding the Y column is encoded so that we get as many columns as there are classes in Y, with the selected class having the value 1 and the rest having the value 0. This method is called PLS-DA (Principal Least Squares Discriminant Analysis) [17]. The results obtained from the model are not categorical but the numbers are an approximation on which we can apply a threshold to make predictions.

PLSR algorithm has the advantage over other algorithms when dealing with high dimensional data which suffers from multi-colinearity and high correlation resulting in overfitting. This was the reason that in our thesis we have explored PLSR as an option to enhance RENT algorithm to include multi-class classification and regression.

2.4 PLSR RENT

PLSR-RENT is a feature selection method developed as an attempt to extend the scope of RENT to multi-class classification and regression. Given X with a set of features $F \subset \{1, ..., N\}$, the goal of the PLSR-RENT is to find a subset of features $F' \subset \{1, ..., M\}$ such that $M \leq N$. The expected requirement of our technique is to balance performance and stability.

2.4.1 PLSR-RENT Methodology

The performance aspect is measured by some performance metric. In our approach we calculate number of metrics, but must chose one for decision making, such as balanced accuracy. The stability aspect is ensured by using ensemble of PLSR models with various subsets of the data and repeating the process a number of times. Our method uses the cumulative explained variance for each individual feature variable in X as the criterion to select features. The cumulative explained variance of each feature must pass a threshold value for the feature to be selected. The result of PLSR-RENT is a new X' with reduced selected features.

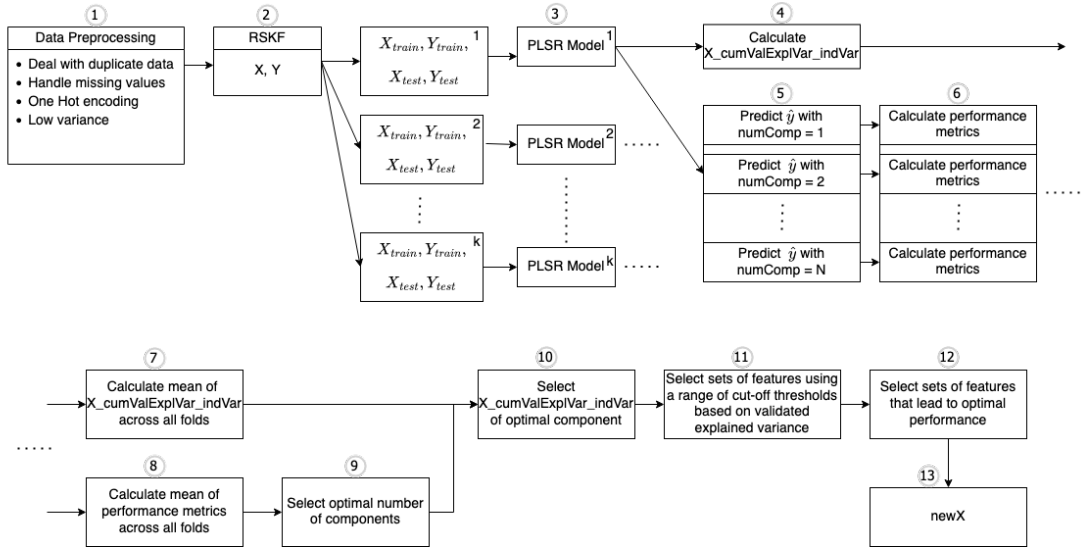


Figure 2.11: Workflow of the PLSR-RENT

Figure 2.11 visualizes the workflow of the PLSR-RENT methodology. The steps of the process are numbered and are used for reference in explanation below.

1. We preprocess the data with N number of features in step 1.

2. The data is split using Repeated Stratified K-Fold (RSKF) (explained in section 2.5.3) in step 2. RSKF is a cross validation technique which returns multiple train and test data splits.

3. Each split of data is used as input for a PLSR model - creating an ensemble of k PLSR models (step 3).

4. For each model we calculate:

   (a) $X\_cumValExplVar\_indVar$: Cumulative calibrated explained variance for each variable in X, i.e. a N length list (step 4).

   (b) Then, in step 5, we iteratively predict X_test using number of components from 0 to N for all the model and save the results as in figure 2.12(a).

   (c) In step 6, we then compute the performance metrics and save them as shown in figure 2.12(b).

	Feature1	Feature2	...	FeatureN
comp0				
comp1				
⋮				
compN				

(a) Example table of $X\_cumValExplVar\_indVar$ for each variable in X

	Accuracy	F1_Score	...	...
comp0				
comp1				
⋮				
compN				

(b) Example table of mean of metrics for each component

Figure 2.12: Tables of mean of $X\_cumValExplVar\_indVar$ and performance metrics across k models

5. After all k models we have k dictionaries with performance metrics and k lists with $X\_cumValExplVar\_indVar$. We find mean of $X\_cumValExplVar\_indVar$ in step 7 and mean of performance metrics dictionaries (element wise mean), in step 8, as explained in figure 2.13

6. Based off on a given performance metric, we choose the component with the maximum performance score from the list performance scores.

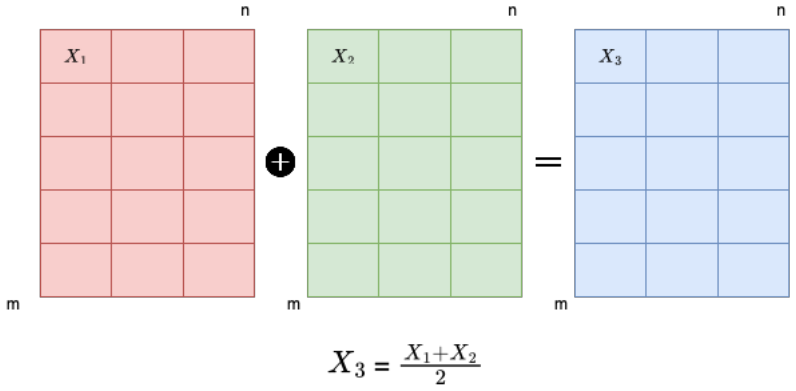


Figure 2.13: Element-wise mean across a dictionary/matrix

7. In step 10, we select the row of optimal number of components from the table of *X_cumValExplVar_indVar*.

8. In step 11, we recursively compare each feature to a cutoff threshold based on validated explained variance. Example if the cumulative explained variance of a feature is greater than a given threshold we keep it, otherwise discard that feature. We are left with a set of features which we process through a model and get results.

9. In the end we will choose the set of features gained from a threshold with maximum score in step 12. We can also manually choose where to cutoff by compromising slightly on the performance score but gaining improved computational efficiency.

2.4.2 Comparison methods

We use hyper-tuned versions of PLSR and multiple other multi-class classifiers (discussed in section 2.6) with X as baseline. We then use the new X' with the same classifiers and hyper-tune them. The results obtained are compared with the baseline results to observe the differences. Figure 2.14 summarizes the process.

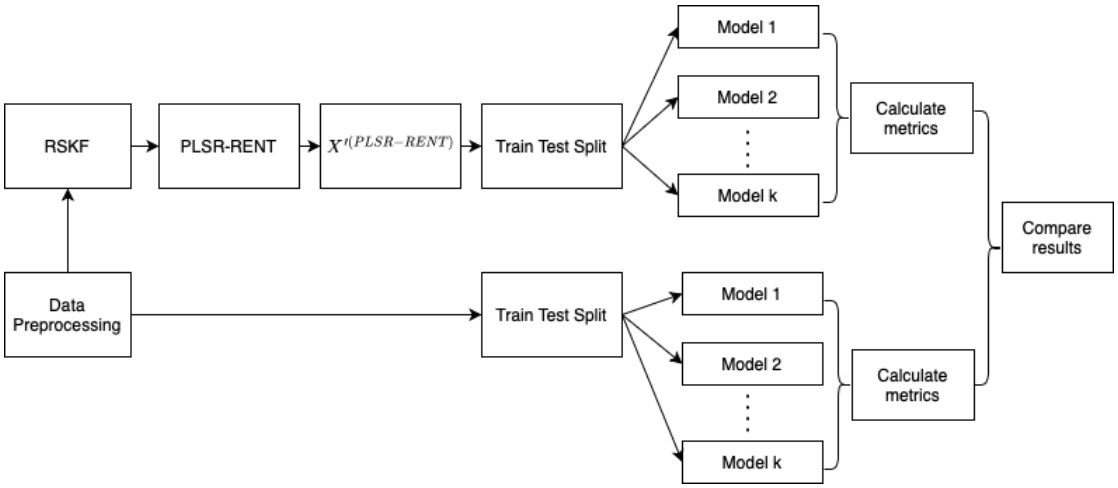


Figure 2.14: Process of comparison of PLSR-RENT with other classifiers

PLSR-RENT is also compared with other established feature selection methods from scikit-learn (discussed in 2.1). We obtain $X'^{(PLSR-RENT)}$, $X'^{(FS1)}$ and $X'^{(FS)2}$ from PLSR-RENT, FS1 and FS2, respectively. Where FS1 and FS2 are two different feature selection methods.

We then run the different hypertuned models with $X'^{PLSR-RENT}$, X'^{FS1} and X'^{FS2} and compare their results as shown in figure 2.15.

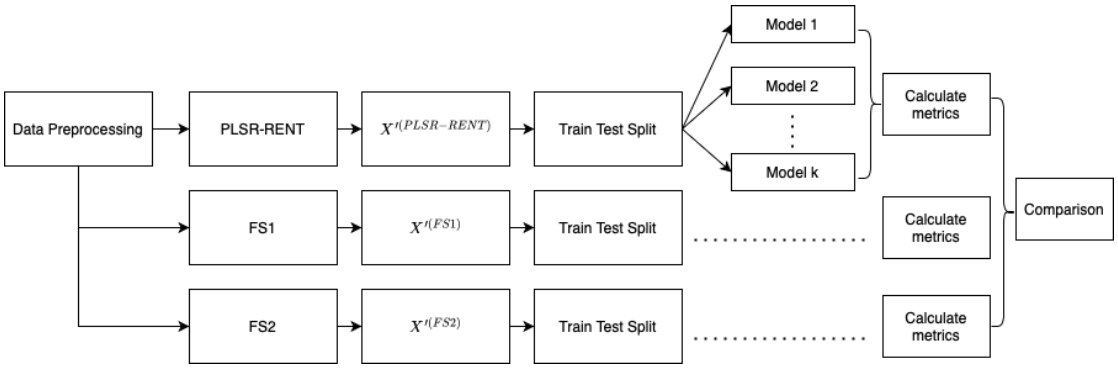


Figure 2.15: Process of comparison of PLSR-RENT with other feature selectors

2.5 Model Validation

In machine learning, splitting the data into simple train and test split using basic *random sampling* presents problems with stability of the results. The performance of the model is prone to change depending on how the data is split randomly into samples, or if the samples are very different from each other, or both. Which samples the model was

trained on and which samples it was tested on may have an influence on the performance of the model.

2.5.1 K-fold

cross validation solves the problem of having different results with different samples by dividing the data into K folds. K models are trained on $K-1$ folds and performance is computed from the last left out fold. The performance across the folds provides a more robust estimate of the overall performance. This technique, while better than just taking a single random sample, still suffers from randomness of the data.

Another method of sampling is *stratified sampling*. In stratified sampling the percentage of *strata* (subgroups or classes) are maintained in the sample taken as in the original data. For example, in original data 80% of the data is class A, and the 20% is class B. In stratified sampling, the sample of the data will maintain the same percentage of the classes.

2.5.2 Stratified K-Fold

is a cross validation technique used to get k-folds(k-samples) of the data with stratified sampling. It splits the original data into K number of folds, maintaining the proportion of the classes in each fold as in the original data. This ensures that each of the K models are trained on subsets of the data that have the same class distributions. This is a useful technique when dealing with class imbalance. In case of great imbalance between classes stratified K-fold ensures that a fold does not contain only the dominant class which might lead to poor predictions for the left-out minority classes.

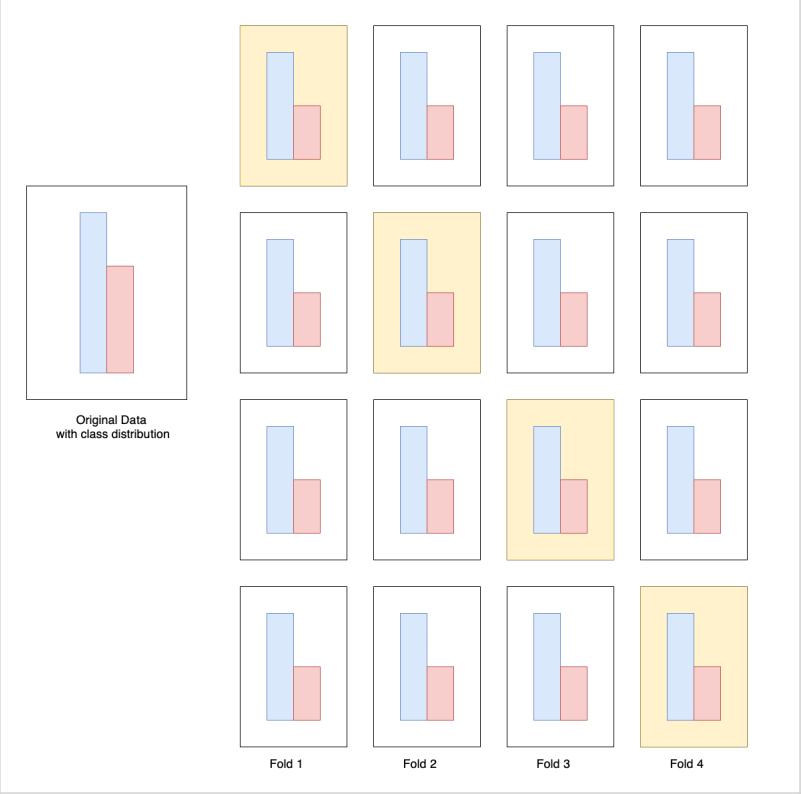


Figure 2.16: Visualization of Stratified K-fold

Figure 2.16 shows the working of stratified K-fold, how it maintains the class distribution in all the splits of the data. Each row represents a single split of the data where it takes one fold of the data for testing (highlighted in yellow box), and the rest for training (in white boxes).

While stratified K-fold might be better than simple K-fold at estimation of classification tasks, there is still some chance of noisy estimates of the performance of the model.

2.5.3 Repeated Stratified K-Fold

Repeated Stratified K-Fold (RSKF) takes stratified K-fold to the next level by repeating the stratified K-fold n number of times. It is expected to have more accurate and stable results of the performance of the model. But this accuracy and stability might come at the cost of it being more computationally expensive than stratified k-fold. For a 10-fold cross validation using RSKF repeated 5 times, it gives us 50 iterations of data samples; i.e. $n\_splits * n\_repeats$. Meaning it is 5 times more computationally expensive. In

each iteration, one fold is reserved for testing, while others are used for training. Also, the rows of data are shuffled at the start of each iteration.

In our experimentation we have used RSKF from the scikit-learn library to ensure model stability and accurate estimation of performance.

```
RepeatedStratifiedKFold(n_splits=10, n_repeats=5,
                        random_state=123, n_jobs=-1)
```

2.6 Multi-Class Classifiers

Multi-class classifiers are algorithms that are capable of classifying more than two classes; i.e $y \in \{1, 2, , 3, ..., k\}$. As opposed it, binary classifiers are capable of classifying only two classes; i.e $y \in \{0, 1\}$.

In binary classification the target classes are often the inverse of each other; for example, classifying whether a patient is sick or healthy, or the fruity is apple or not. Whereas in multi-class classification the target classes are not necessarily inverse. Example of multi-class classification is classifying the fruit as apple, orange or peach. In real life cases most of the classification tasks are multi-class. The applications of multi-class classification range from image classification, medical diagnosis, product classification, facial recognition, content analysis of social media, etc.

There are a number of classifiers built to handle multi-classification tasks. Below we will discuss a few that are used in our thesis.

2.6.1 Multinomial Logistic Regression

Logistic regression algorithm is made for binary classification problems. Multinomial logistic regression model is an extension and generalization of its binary counterpart. It is made to classify more than two classes.

Multinomial logistic regression is also called *softmax regression, multiclass logistic regression*, *maximum entropy classifier*, or *polytomous logistic regression.*

The multinomial logistic regression makes the assumption that the preference of one class over the other does not depend on other choices. For example, if the algorithm gives preference (higher probability) to class A over class B from alternatives class A, class B; the addition of class C to the alternatives should not affect the preference of class A over class B.

The softmax function outputs a vector of probabilities for each alternative and then argmax is applied to give a one hot vector where 1 is the predicted choice and the rest are 0.

Activation function of binary logistic regression is sigmoid function, which is generalized to softmax function for multinomial logistic regression.

An input vector $z = [z_1, z_2, ..., z_k]$ is received by the softmax function, and returns a vector of probability distribution for all the k classes. It can be defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j-1}^{K} e^{z_j}} \tag{2.14}$$

where σ is the softmax function, e^{z_i} is the exponential function for input vector, and e^{z_j} is the exponential function for output vector.

The probabilities of the softmax function lies between $[0, 1]$ and always sums up to 1.

Softmax function for logistic regression can be adapted as in equation 2.15.

$$p(y_k = 1 \mid x) = \frac{e^{(w_k x)} + b_k}{\sum_{j=1}^{K} e^{(w_j x)} + b_j} \tag{2.15}$$

Where x is the input vector, w is the weight vector, and b is bias. The input for softmax is the dot product of the weight vector and input vector.

$$input = [w_1, w_2, ..., w_k].[x_1, x_2, ..., x_k] + b \tag{2.16}$$

The goal of the classifier is to find ideal weights, minimize the cost function and improve the correct predictive probability. Gradient descent algorithm finds the optimal weights. The predictive output can be given as:

$$\hat{y} = \sigma(wx + b) \tag{2.17}$$

2.6.2 Support Vector Machines

Support vector machines (SVM) is a natively binary classification algorithm built on a linear model. It can be used for multi-class classification with some modification. SVM works by finding a boundary between the given classes. This differentiating decision boundary is called *hyperplane*. The goal of SVM is to find the optimal hyperplane in n-dimension that can clearly differentiate between given classes by maximizing the distance between data points.

Kernels are responsible for the computation of separation of data points. SVM uses different kernels for its computations: linear, Radial Basis Function(RBF), polynomial, gaussian and sigmoid. Kernels determine the shape of hyperplanes and how they operate at differentiating classes, shown in figure 2.17. Hypertuning with their parameters result in more efficient class differentiation and thus prediction.

The SVM can be extended to multi-class classification by using either *One-vs-One* or *One-vs-Rest* approach.

1. *One-vs-One* works by treating the multi-class problem as multiple binary classification problems. It divides the classes into pairs and classifies them. Number of SVM models needed for One-vs-One method are expressed as below:

$$n\frac{(n-1)}{2} \tag{2.18}$$

2. *One-vs-Rest* works by building a classifier to separate data points as belonging to a particular class or to the rest of the classes. The number of SVM models needed for One-vs-Rest method is n

In the example taken from [18], we have to classify between three classes: blue, green, and red. In One-vs-One approach, as shown in figure 2.18(a), each hyperplane is for the separation of two classes. The blue-green line separates between blue and green classes. The green-red classifies the points as green or red, and so on. In One-vs-Rest approach,

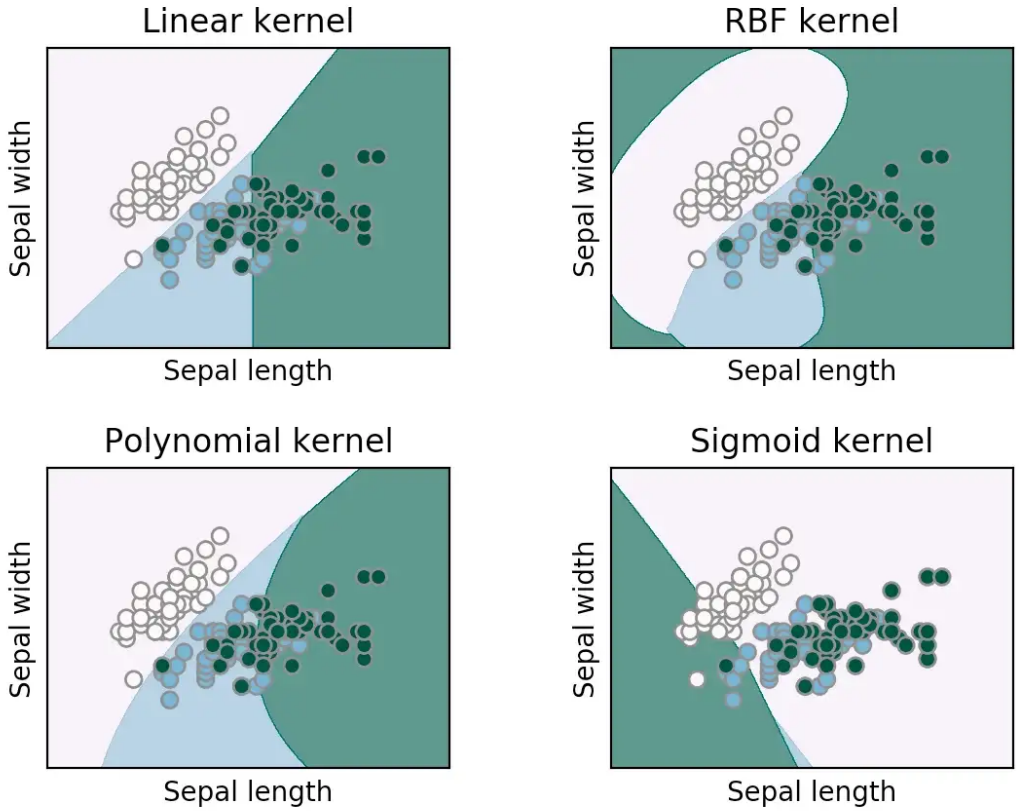
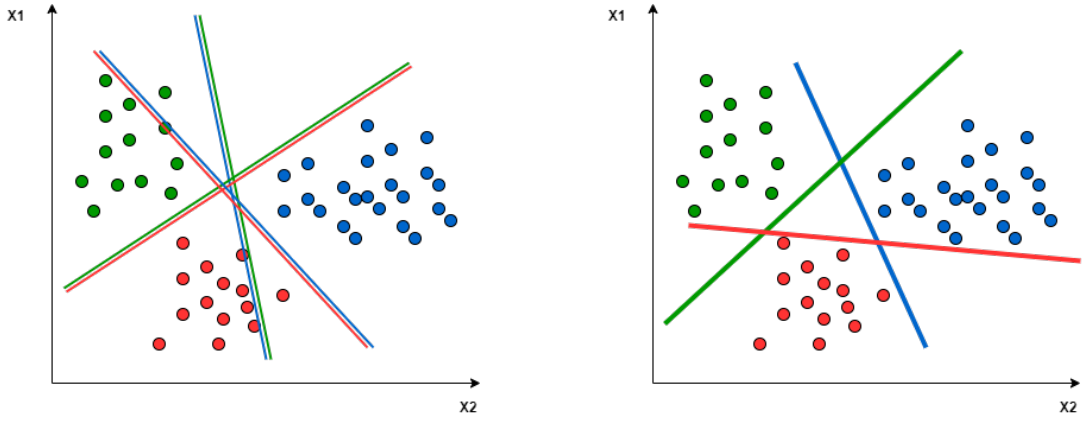


Figure 2.17: Visualization of how different kernel functions create different hyperplanes to differentiate classes

as shown in figure 2.18(b), the green line is classifying the data points as green or not green, the red is classifying as red or not red, and so on.



(a) One-vs-One approach of SVM

(b) One-vs-Rest approach of SVM

Figure 2.18: Different approaches of SVM for multi-class classification[18]

2.6.3 Random Forest

Random Forest is a type of supervised machine learning algorithm which can be used both for classification and regression tasks. Random forest is, at its core, an ensemble of decision trees. Each model tree performs as an individual model and predicts a class for the given classification task. Majority voting is then used to reach a conclusion. This concept can be explained by the figure 2.19 where the Random Forest model has to predict between classes 0 and 1.

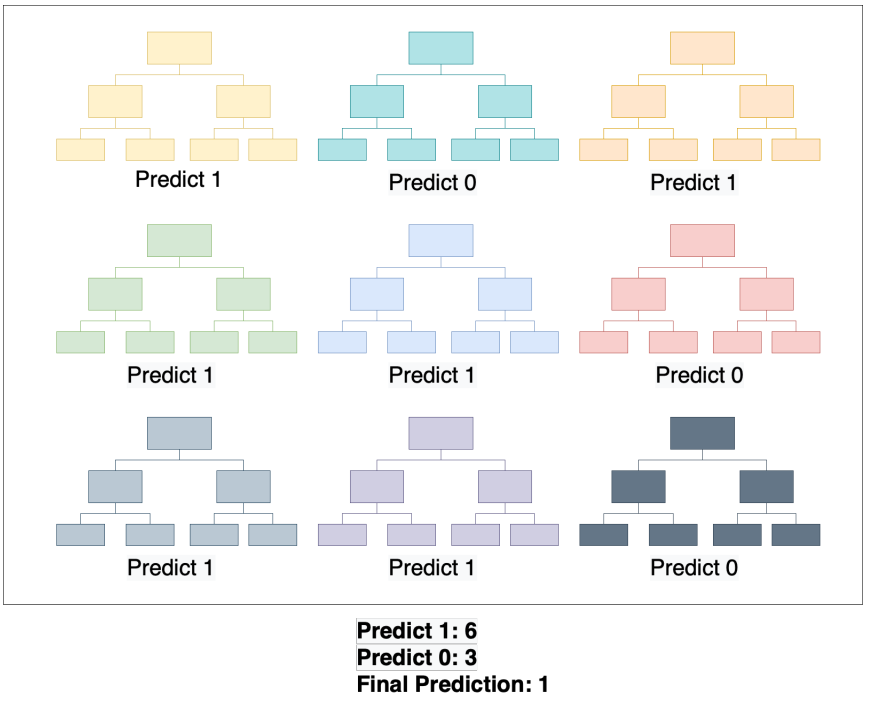


Figure 2.19: Visualization of conceptual working of Random Forest

The criteria to select the attribute to branch out the tree from can be one of the following two methods:

1. **Information gain/Entropy** uses log function and thus is computationally costly. It calculates unpredictability of the data.
2. **Gini-Index** measures *impurity* i.e. probability of a particular attribute being wrongly predicted when chosen at random.

While both methods can have similar results, but the difference is in their performance. Gini is much faster than entropy because, as mentioned, entropy uses log function.

The trees in random forest are uncorrelated to each other or the correlation is very low; and that is the most vital part of Random Forest. This safeguards that the error of one model do not affect the others. In theory, if there is a recognizable pattern in the given data then most of the trees will advance in the same direction.

There are two type of methods used by the Random Forest algorithm to build the individual models with variations:

1. **Bagging/Bootstrap**: Decision tree models are very sensitive to small change in training data. Using that to our advantage we can diversify the models created in Random forest algorithm. Given a training sample of size N, bagging method creates multiple subset samples with replacement, each of same N size. Majority voting decides the final predicted result.

2. **Feature Randomness**: In Random Forest algorithm, for each tree a random subset of features is selected - as opposed to general decision trees, which use the entire features set. This randomness of features ensures that each model is different from the next model.

So, to summarize the random forest ensures that there is low correlation between models by each mode using different training data (subsets) and different features.

In random forest ensemble, the greater the number of the trees are the more accurate the result is going to be and less prone to overfitting. We can also optimise the random forest algorithm by

One more feature of random forest which makes it so favorable is that it calculates the relative feature importance. This helps us to drop features of low importance during the process of feature selection.

2.6.4 K-Nearest Neighbors

K-Nearest neighbours(KNN) is a very simple supervised machine learning algorithm used for both classification ad regression problems. The algorithm is *non-parametric*; which means that it do not use any parameters gained from the training data to make predictions. Rather, non-parametric algorithms use comparison techniques to make

predictions. KNN instead of learning from the training data, saves the whole data for comparision later when asked for prediction. This is the reason why KNN is also known as *lazy algorithm*.

As the name suggests, KNN makes predictions based on the proximity or the distance to its neighbouring data points' classes. It assumes that the data points belonging to the same class must be located near each other in the dimension.

Figure 2.20 visualizes an example where a new data point *?* has to be classified as class plus, circle or triangle. If we choose $k = 5$ then according to the nearest neighbours the class triangle would have the majority and the new data point would be classified as class triangle.

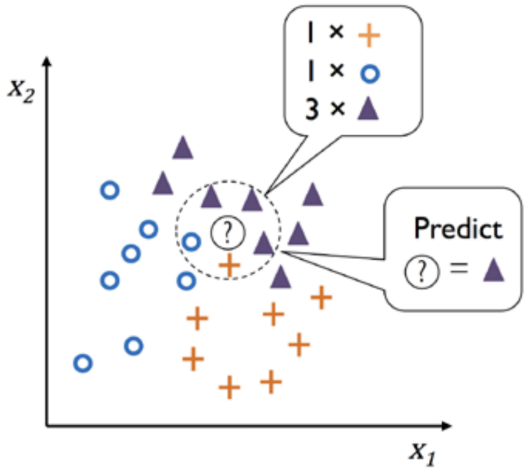


Figure 2.20: KNN classifier example to classify a point as class plus, circle or triangle with $k = 5$ [19]

To predict the class of a given data point, KNN utilizes a distance metric to measure its distance from its neighbours then selects the k closest data points. It makes the prediction based on the class memberships of its neighbours. The predicted class is then the class with most frequency.

The most common distance metrics used by KNN algorithm are Euclidian distance, Manhattan distance, Minkowski distance, among many others.

- Euclidian distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.19}$$

- Manhattan distance:

$$d(x, y) = \sum_{i=1}^{n} \mid x_i - y_i \mid \tag{2.20}$$

- Minkowski distance:

$$d(x, y) = \left(\sum_{i=1}^{n} \mid x_i - y_i \mid^c \right)^{\frac{1}{c}} \tag{2.21}$$

2.7 Performance Metrics

Performance metrics let us know how well our model is performing. Choosing the performance metric for evaluating a machine learning model is dependent on a number of things, some of which might be the outcome we wish to achieve, the balance of the classes involved, and type of data. There is no particular metric which can generalize the performance evaluation of a model. Therefore, it is up to the user to choose the performance metric which fits best to their model and requirements.

For every binary classification metric there are three multiclass counterparts of the particular metric:

- micro-averaged,
- macro-averaged, and
- weighted averaged.

For example, precision in binary classification has three versions for multi-class: macro averaged precision, micro averaged precision and weighted precision. Similarly, there are three versions of other metrics such as recall and F1 scores. Micro-averaging calculates the metric from the combined inputs from all classes and averages them. This way it treats the classes with majority more favorably.

Macro-averaged metrics deals with all classes the same. It calculates the metrics independently per class and then averages the results.

Weighted averaged metrics are the same as the macro averaged metrics, except that it accounts for the weights of each class(samples per class).

Macro metrics are better suited when the classes are balanced in the data. Whereas, for imbalanced data micro or weighted metrics would be more favorable to use.

Below we discuss some of the performance metrics which can be used with multi-class classification problems.

2.7.1 Accuracy

The most simple and most used metric is accuracy. Accuracy is the measure of effectiveness of a classifier. It is the total number of correct prediction divided by the total number of predictions carried out.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.22}$$

For multi-class case with k classes, the average accuracy can be calculated as follows:

$$AverageAccuracy = \frac{\sum_{i=1}^{l} \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}}{k} \tag{2.23}$$

2.7.2 Balanced Accuracy

Balanced accuracy is adaptation of accuracy for imbalanced classes. Instead of the ratio of total number of predictions, balanced accuracy calculates the mean accuracy for each class. Following equation shows the gist of balanced accuracy for N number of classes:

$$BalancedAccuracy = \frac{1}{N}\left(\frac{CorrectPredictions_{class1}}{TotalPredictions_{class1}} + \ldots + \frac{CorrectPredictions_{classN}}{TotalPredictions_{classN}}\right) \tag{2.24}$$

It is the average of *Sensitivity* and *Specificity*. Sensitivity is also called recall (whose formula is given above in equation 2.30), and specificity is calculated by:

$$Specificity = \frac{TN}{(TN + FP)} \tag{2.25}$$

Using the above formulas we can then calculate balanced accuracy as:

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2} \tag{2.26}$$

2.7.3 Precision

Precision is the ratio of correctly classified instances of each class to all the instances classified as that particular class.

For binary classification the precision can be calculated by the formula in equation 2.27.

$$PRE = \frac{TP}{TP + FP} \tag{2.27}$$

[20] has adapted the binary equation for multi-class classification as equation 2.28 for micro averaged precision, and equation 2.29 for macro averaged precision. For k classes C_i is the is the individual class and TP_i, FP_i are true positive and false positive for class C_i, respectively.

$$PRE_{micro} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} (TP_i + FP_i)} \tag{2.28}$$

$$PRE_{macro} = \frac{\sum_{i=1}^{k} \frac{TP_i}{TP_i + FP_i}}{k} \tag{2.29}$$

2.7.4 Recall

Recall is the measure of the ability of the model to correctly classify all instances of each class. Recall is also referred to as *true positive rate (TPR)* or *Sensitivity*.

Equation 2.30 gives the formula for recall of the binary classification task. This equation is modified for multi-class classification task as equation 2.31 and equation 2.32 for micro and macro recall, respectively.

$$REC = TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.30}$$

$$REC_{micro} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} (TP_i + FN_i)} \tag{2.31}$$

$$REC_{macro} = \frac{\sum_{i=1}^{k} \frac{TP_i}{TP_i + FN_i}}{k} \tag{2.32}$$

2.7.5 F1 score

Harmonic mean of precision and recall is called F1 score. It equally makes the use of recall and precision. It is better suited in case of imbalanced data, as opposed to only precision or only recall.

F1 score can range from 0 to 1, where 0 is the worst score and 1 being the perfect score. F1 score can also be referred to as just $Fscore$.

Equation 2.33 gives the general formula of F1 score for binary classification.

$$F1 = 2\frac{PRE.REC}{PRE + REC} \tag{2.33}$$

Using equations calculated previously for micro and macro precision and recall, we can generalize equation 2.33 and calculate F1 micro and macro scores for multi-class below:

$$F1_{micro} = 2\frac{PRE_{micro} REC_{micro}}{PRE_{micro} + REC_{micro}} \tag{2.34}$$

$$F1_{macro} = 2\frac{PRE_{macro} REC_{macro}}{PRE_{macro} + REC_{macro}} \tag{2.35}$$

2.7.6 Cohen's Kappa Score

Cohen's kappa score or Cohen's kappa coefficient is a statistical measure of an agreement between two raters, which, in the case of classification performance, are: predicted and true values.

Cohen's kappa score takes imbalance of the data into consideration during calculation. Therefore, this is a good evaluation metric while dealing with imbalanced data.

It is can be calculated using the following formula:

$$\kappa = \frac{(p_o - p_e)}{(1 - p_e)} = 1 - \frac{1 - p_o}{1 - p_e} \tag{2.36}$$

where p_o is the observed agreement and p_e is the expected agreement.

2.7.7 Mathews Correlation Coefficient (MCC)

MCC calculates the correlation coefficients between the observed and predicted values. It is calculated by the general formula:

$$MCC = \frac{TP.TN - FP.FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.37}$$

Accuracy and F1 score are more popular measures of model evaluation in machine learning but MCC is more advantageous in the sense that it does not inflate the results - which might be the case with accuracy or F1 score. In MCC we gain a good high score only if all across the confusion matrix categories(TP, TN, FP, FN) scores are good.

Chapter 3

Experimental Setup

For a list of all the ways technology has failed to improve the quality of life, please press three.

— *Alice Kahn*

PLSR-RENT is our attempt to extend RENT to the scope of multi-class classification. In this chapter we will describe our methodology and workflow to the PLSR-RENT approach. Section 3.2 outlines the workflow of our experimental setup, section 3.4 details the data preprocessing, the feature selection methods, including PLSR-RENT, is discussed in section 3.5. In section 3.6 we explain the different classification pipelines used. Finally, in section 3.7 and 3.8 hypertuning parameters for classifiers and results are discussed, respectively.

3.1 Software

Programming Language and Tools

- Python 3.9.13
- Anaconda Jupyter Notebook (conda version 22.9.0)
- JetBrains PyCharm IDE

Libraries used for Data handling and machine learning

- Scikit-learn [21]
- Pandas [22]
- Numpy [23]
- Hoggorm [24]
- Hoggormplot [25]
- Matplotlib [26]
- Seaborn [27]

3.2 Workflow

Figure 3.1 briefly outlines the steps in the workflow of our PLSR-RENT methodology for better understanding. We will refer to this figure later while explaining the processes in more detail in later sections.

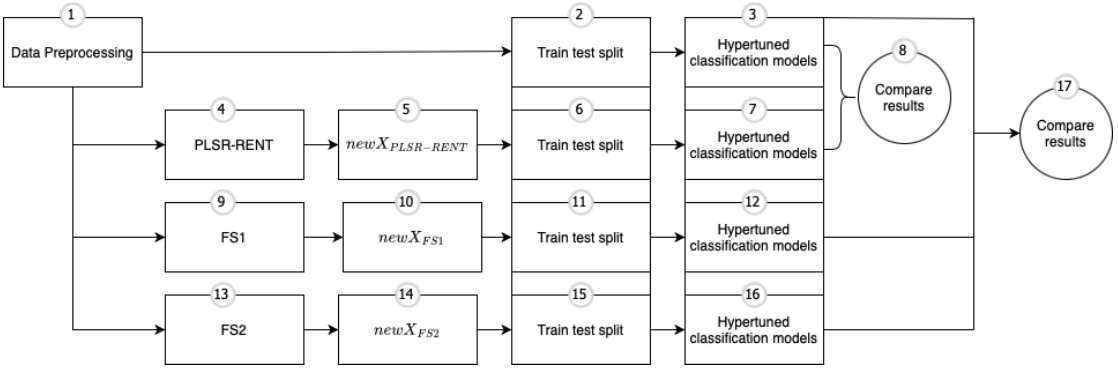


Figure 3.1: Workflow of the experimental setup

3.3 Data

The dataset that we are using is *Multiple Features Data Set* [28], accessed from [29].

It is a dataset of features of hand-written digits (0-9). This dataset is a collection of six sets of data. We are using one of these datasets from the collection. It is a set of

76 fourier coefficients of handwritten digits. It consists of 200 samples for each class of which there are 10 (2000 samples in total).

It has total of 77 columns, with 76 features and 1 target class column.

3.4 Data Preprocessing

When building a good machine learning algorithm preprocessing of the data that goes into the model is equally important than the actual model itself, if not more so. If we put in data with redundancy and noise, our results from the model will reflect that.

Data preprocesing is our first step of the workflow, as described in 3.1.

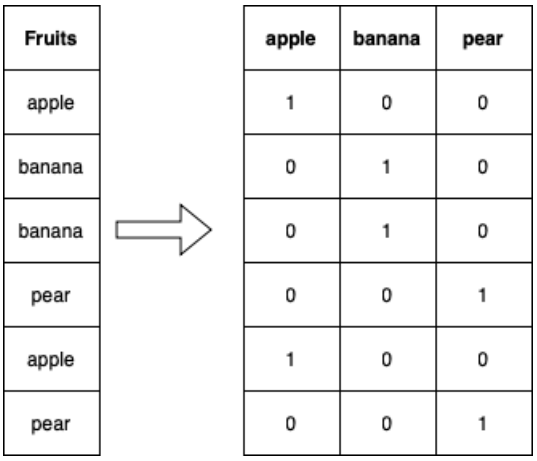
In our thesis we have used a few different data sets and each required tailored data preprocessing. The common steps in data preprocessing are: (1) checking for duplicates, (2) checking and handling missing values, (3) one-hot-encoding the categorical target values, and (4) removing columns with no variance.

3.4.1 Handling Missing and Duplicate Data

First step in data preprocessing is to check for duplicate and missing data and dealing with them accordingly. For handling duplicates, we simply drop all the duplicates as they present redundant information using the built-in function *drop_duplicates* of Pandas.

Missing data is represented by *NaN*, *null* or *None* in the data. To deal with it, there are few different techniques available:

- If the missing data in a column is greater than a given threshold then we simply drop the whole column.
- Similarly, for the rows, if more than a certain number of values are missing, then we drop the row.
- If the missing data is under the threshold limit, then we handle it by imputation of the missing value by KNN-based imputation. There are other interpolation techniques available, but we have using mean to handle the missing data.



Fruits
apple
banana
banana
pear
apple
pear

apple	banana	pear
1	0	0
0	1	0
0	1	0
0	0	1
1	0	0
0	0	1

Figure 3.2: Visualization of how one-hot encoding decomposes a single categorical column into multiple columns

3.4.2 One Hot Encoding

The PLSR method does not take categorical values for its response Y input as it is basically a regression algorithm and not a classification one, and thus only accepts numerical values. To go around that issue we use one-hot encoding to encode the categorical values.

For N number of classes, one-hot encoding converts a single Y column to N columns with every column representing a class/category. For a single sample, only one of the columns with corresponding class would have the value 1 and the rest would be zeros. Example shown in figure 3.2 can visualize the explanation better.

3.4.3 Low variance threshold

If column does not change its values or the change is very minimal, i.e. the values are (almost) constant, then it does not contribute any useful distinguishing information when a model is learning. We remove such columns with constant values ($VarianceThreshold = 0$) in the preprocessing step.

3.5 Feature Selection

After data-preprocessing, we have used three methods for feature selection in our experimentation in steps 4, 9 and 13 in figure 3.1. One method is our PLSR-RENT and then

we use two other established methods provided in scikit library to compare our method with.

3.5.1 PLSR-RENT

We have explained the workflow of PLSR-RENT in section 2.4 and a detailed visualization is shown in figure 2.11. We feed the input data into the PLSR-RENT model and we get an output of selected features $X_{PLSR-RENT}$.

We can choose which performance metric to base off our decision of optimal number of components as done in step 9 of figure 2.11. The cutoff-threshold is automatically selected based on which one results in the highest score of the selected metric. We can manually select the cutoff-threshold for a trade-off between more reduced number of features or slightly more accurate prediction performance.

3.5.2 SelectKBest

SelectKBest univariate feature selection model is used in our experimentation as the first feature selector to compare our method with. It is a filter method for feature selection (as described in section 2.1.1). It selects K best features based on the scores of the features. The two most common scoring functions are: chi-squared or ANOVA F-value.

ANOVA is analysis of variance. It calculates the variance between the samples using the F-statistic.

Chi-squared test builds a contingency table for every feature and class pair to measure their dependency relationship. The features with most dependency are rated higher and features with little to no dependency are rated less. For feature selection n least scored features can be discarded.

As the Chi-square test can only be used when all of the values in the feature are non-negative, we have used ANOVA F-value.

We have used a grid search to decide the number of features K. We choose the hyperparameters with the best score for balanced acccuracy.

3.5.3 ExtraTrees Feature Importance

Tree-based feature selection uses the feature importance calculated in the training of the model to rank the features. Extra Trees classifier is an ensemble of multiple decision trees. It is similar to Random Forest mostly but differs in the construction of the trees.

While building the tree, at each node a mathematical critera is used to decide which is the best feature to split the data (usually the Gini impurity). This mathematical criteria value is called the feature importance. To perform feature selection we use these feature importance to select the features with the most importance.

3.6 Classification

In figure 3.1 in steps 3, 7, 12, and 16 the *hypertuned classification models* referred to are the hypertuned versions of the following five classification models:

1. PLSR
2. Logistic Regression
3. SVC
4. Random Forrest
5. K-Nearest Neighbors

All the classifiers listed above are explained in section 2.6. We use these algorithms with four different model pipelines:

1. Baseline model pipeline
2. PLSR-RENT model pipeline
3. FS1 model pipeline
4. FS2 model pipeline

3.6.1 Baseline model pipeline

Baseline model consist of steps 1, 2, and 3 in figure 3.1. No feature selection is used in this pipeline; i.e. all the features are used. This model simply take the preprocessed data and uses them as inputs for the hypertuned classification algorithms. The results produced are used as a baseline for comparison with our PLSR-RENT model results.

3.6.2 PLSR-RENT model pipeline

In PLSR-RENT model the preprocessed data is passed through the PLSR-RENT algorithm to get a $X_{PLSR-RENT}$. This is a feature reduced version of the original data X. This is used in place of X in train-test-split and then the classification models are hypertuned according to it. The PLSR-RENT model takes the path of steps 1, 4, 5, 6, and 7.

3.6.3 FS1 model pipeline

FS1 model takes the steps 1, 13, 14, 15, and 16. For FS1 model we are using *SelectKBest* univariate feature selection model in step 9 of the workflow figure in 3.1. The number of features are selected using the grid search, and the set of features with the best balanced accuracy score is selected as the new X called X_{FS1}.

3.6.4 FS2 model pipeline

We are using embedded feature selection method using *ExtraTreesClassifier* in *SelectFromModel* method of scikit-learn. 2.1.3 explains the method briefly. It uses the feature importance to rank the best n features. It auto selects the optimal number of features when we do not provide the number of features. The new X with optimal feature subset is given by X_{FS2} The FS2 model takes the following steps: 1, 13, 14, 15, and 16.

All the above mentioned models are cross validated using the same Repeated Stratified K-Fold during the hypertuning of the classification models.

3.7 Hyperparameter Tuning

Hyperparameters can be viewed as the settings of an algorithm which can be fine tuned to gain the best result from it. Hyperparameters msut be provided by the user before training the model. If not explicitly provided by the user then the default values are used.

We have four datasets, one containing all features and three with subsets of the full feature set, $(X, X_{PLSR-RENT}, X_{FS1}, X_{FS2})$ and five classifiers in our setup. Hypertuning a classifier parameters are dependent on the X and Y values given, so we hyptertune each classifier three times for each X.

GridSearchCV is a function found in scikit-learn library which helps to search for the parameters that give the best score for a given model. It takes estimator, parameter grid, scoring options and cv (cross-validation) as input from the user, among few others.

GridSearchCV performs an exhaustive cross-validated search of the parameter grid provided by the user to get the best validation score of the model (given in estimator). We have selected RSKF as the cross-validation technique. RSKF is described in section 2.5.3.

Table 3.1 details the algorithms and their hyperparameters with descriptions and default values as given in [21].

Table 3.1: Hyperparameters used for tuning the classification algorithms used in our experimental setup with descriptions and default values

Algorithm	Hyperparameters	Description	Default value
Logistic Regression	*solver*	Algorithm to choose in optimisation problem	*lbfgs*
	penalty	Penalty to use. Options: l1, l2, none, elasticnet	*l2*
	C	Inverse of regularization strength	1.0
SVC	*kernel*	Specifies the kernel type to be used in the algorithm.	*rbf*
	gamma	Kernel	*scale*
	C	Regularization parameter	1.0
RandomForestClassifier	*n_estimators*	The number of trees in the random forest ensemble	100
	max_features	The number of features to consider when looking for the best split	*sqrt*
KNeighborsClassifier	*n_neighbors*	Number of neighbors to use by default	5
	weights	Weight function used in prediction	*uniform*
	metric	Metric to use for distance computation	*minkowski*
PLSR	*n_components*	Number of components	2

3.8 Results

The final step is computing and comparing results. We compute the following performance metrics for each classifier for each feature selection pipeline.

- Accuracy
- Train accuracy
- Balanced accuracy
- Precision (micro and macro)
- Recall (micro and macro)
- F1 score (micro and macro)
- Matthews Correlation Coefficient (MCC)
- Cohen Kappa Score

After computing the metrics, we first compare our PLSR-RENT results with baseline results in step 8 of figure 3.1, then we compare the results of PLSR-RENT with two other feature selection methods in step 17 of figure 3.1.

Chapter 4

Results and Discussion

The production of too many useful things results in too many useless people.

— Karl Marx

In this chapter we will present the results acquired from the work done in our experimental setup. The discussion of what these results represent will follow along. The results presented in this chapter are from a a single dataset. More experimentation was carried out on multiple other datasets whose results will be presented in the Appendix later.

4.1 Data Preprocessing

There are total of 77 attributes (76 features, 1 target) and 2000 samples in the dataset: 200 samples per class (10 total). After importing the data we checked for and did not find any missing values. There were a total of 6 duplicate rows which were removed. The low variance threshold was applied to check for constant single values in a column. All the feature columns are of the type $float64$ while the target feature is the type *object*. We have total of 10 classes which are almost evenly balanced (completely balanced before dropping duplicates).

The classes are encoded in the form below:

```
array([b'1', b'2', b'3', b'4', b'5', b'6', b'7', b'8', b'9', b'10'],
```

```
dtype=object)
```

We first use *LabelEncoder* to encode the classes as shown in table below:

Table 4.1: Label encoding of the class values

Original Value	Encoded Value
b'1'	0
b'10'	1
b'2'	2
b'3'	3
b'4'	4
b'5'	5
b'6'	6
b'7'	7
b'8'	8
b'9'	9

After label encoding we use *OneHotEncoder* to encode the class attribute.

4.2 PLSR-RENT

The preprocessed data is split with RSKF with $n\_splits = 10$ and $n\_repeats = 5$, giving us 50 data splits and models. The output we get from the ensemble of models is the mean of validated explained variance of each feature and the mean of performance metrics for the ensemble of models in step 7 and 8, respectively, from the figure 2.11. Mean of different performance metrics across number of components is visualized in figure 4.1. We can see in the figure that the performance is sharply increasing starting from 1 component to approximately 10 components. The performance appears to be stabilising around 20 components.

We can also see the spike in precision macro at 0 to 1 components. This is caused by divide-by-zero error encountered when true positives, false positives and false negatives are all 0. This usually occurs in case of no results. For these special cases scikit-learn handles it by replacing it with 1. These "edge cases" results should be avoided.

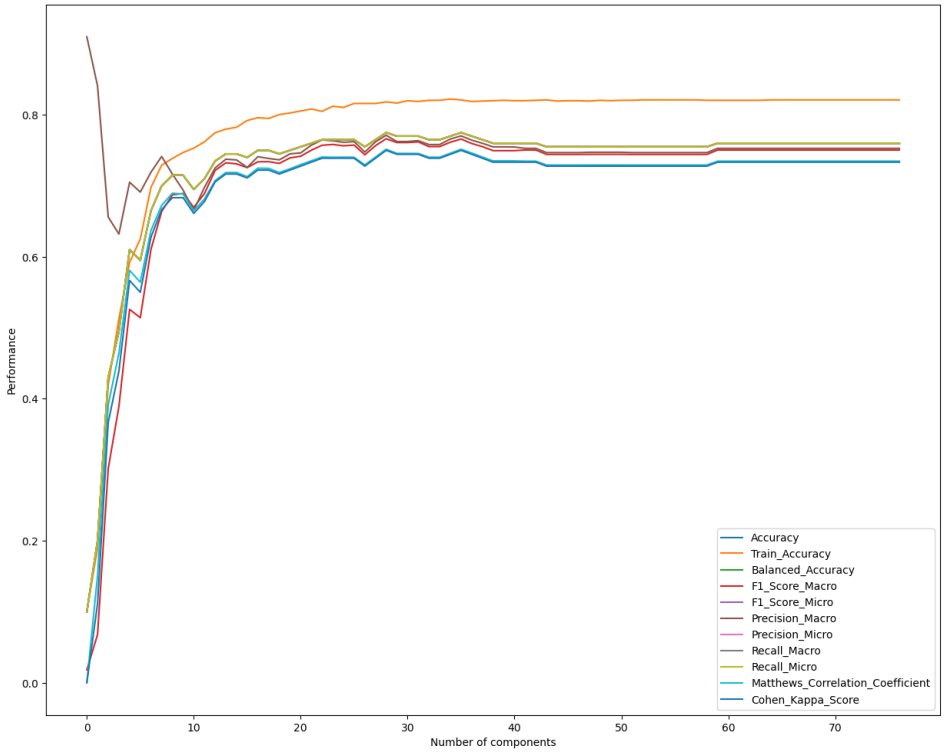


Figure 4.1: Mean of performance metrics across based on 50 models.

Figure 4.2 shows the cumulative validated explained variance of each feature across the 1st component. We can see that features do not have very high validated explained variances - some scoring negative and highest being less than 40.

The cumulative validated explained variance increases as we add more components till a certain number of components before stabilising. We need to find the number of components where we have best results. This is done by choosing the optimal number of components. We have chosen the number of components as 28 (in step 9 of 3.1) based on highest balanced accuracy score (gained from performance metric scores acquired in step 8 of 3.1).

We select the values of cumulative validated explained variance of individual features for 28th component (optimal component). This is visualized in figure 4.3. We can see that all the features have higher scores than those calculated for just 1 component - ranging to almost 100. By sorting the features by their cumulative explained variance we can rank them and discard them for feature selection accordingly, starting from the feature with lowest cumulative explained variance.

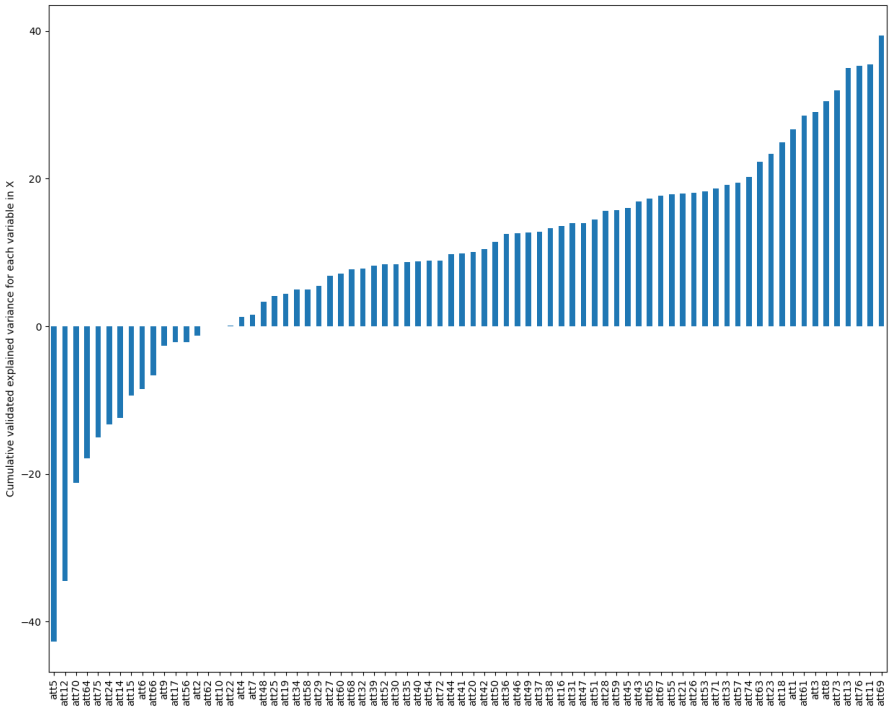


Figure 4.2: Mean of cumulative validated explained variance of individual variables for 1st component. 1st component does not have very high explained variance for all the features. Some features have negative values of cumulative explained variance.

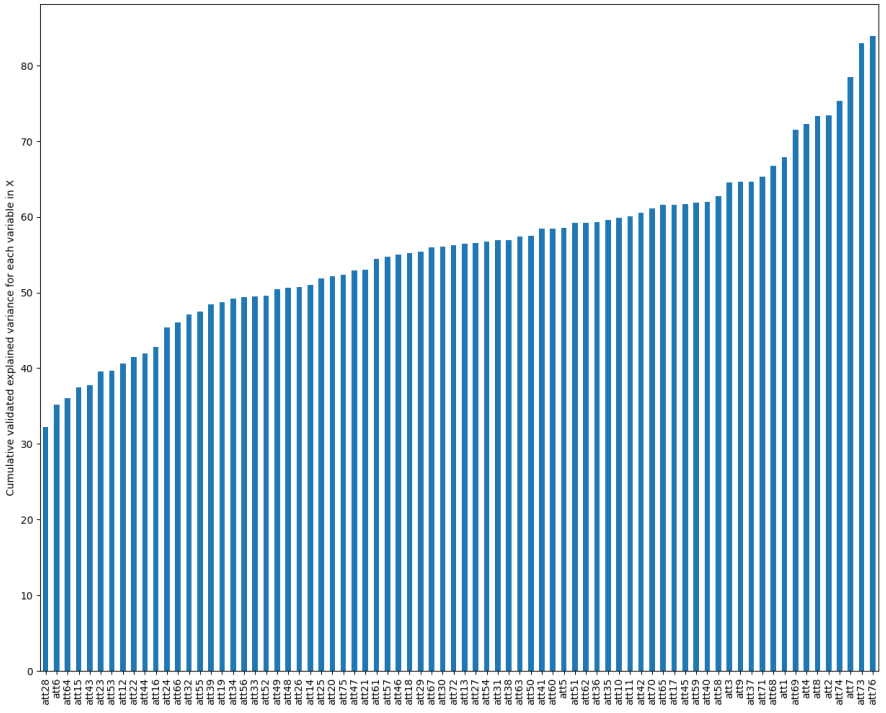


Figure 4.3: Mean of cumulative validated explained variance of individual variables for 28 components. 28 components are the optimal number of components with regards to the performance metrics.

With our chosen cumulative validated explained variance of individual features we can now start discarding features for feature selection based on a threshold. Threshold is set as values ranging from 0 to maximum value of validated explained variance across all features (as the range of explained variance ranges from 0 to 100). We cut off the features that have a validated explained variance lower than the selected threshold, and use the rest as new X to PLSR model and calculate the performance metrics.

Figure 4.4 shows performance metric curves (along left y-axis) and number of features (along right y-axis in black) against cutoff threshold. The figure shows a stable result up until cutoff threshold reaches 33 before it starts to decline. This is because we do not drop any features before reaching threshold of 33 as all the values of cumulative validated explained variance of individual features are above that number. We can see how the drop in number of features is related to the drop in performance scores. Also observed in the figure is the spike in precision macro curve after about 80 cutoff value. This is probably due to divide-by-zero case mentioned above as edge case. Studying the figure we can also conclude the amount of features to select. We can either automatically choose the number of features (cutoff threshold) which gives the maximum score for given performance metric. Or, we can analyze the graph and choose a trade-off: slight drop in performance with dropping off significant amount of features, and thus reduce the computation cost.

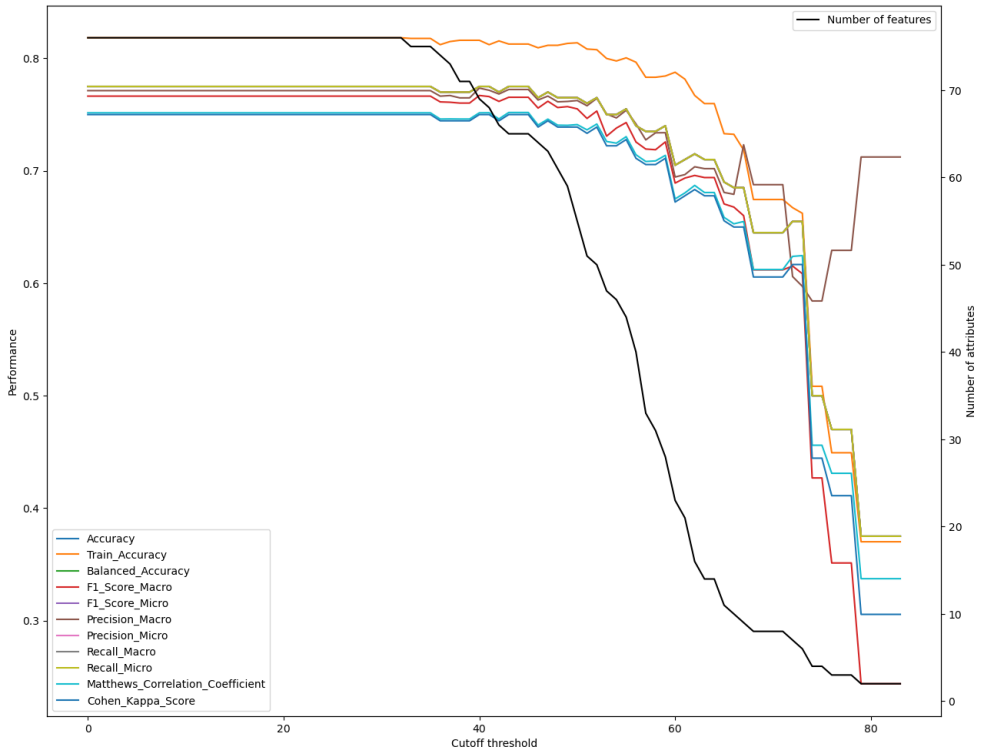


Figure 4.4: Performance metrics curves vs cutoff threshold. The performance metrics have the vertical axis to the left, while the number of features (black line) is defined by the vertical axis to the right

Choosing the cutoff threshold by the maximum balanced accuracy score of 0.775, we get cutoff at 43 with 11 features dropped with 65 selected out of 76 (14.47% feature reduction).

If we manually choose cutoff threshold at 52, we can drop 26 features with 34.21% reduction and the trade-off of score is only 0.01 less. Similarly, by choosing cutoff at 59, we can drop off 48 features (63.15% reduction rate) and the balanced accuracy score is only dropped by 0.035, with score 0.740.

Therefore, choosing the cutoff manually can be a useful tool when we can bend a little on the score for favor of lower computational cost. Table 4.2 shows some sample of the data with cutoff values and the related number of features and scores. This overview can make it easier to see the exact drop in scores and choose the cutoff manually.

Table 4.2: Sample data of cutoff thresholds and associated number of features, scores and feature reduction rates

Cutoff Threshold	Numbers of Features	Balanced Accuracy	Reduction Rate (in %)
42	66	0.770	13.16
43	65	0.775	14.47
46	64	0.765	15.79
47	63	0.770	17.11
48	61	0.765	19.74
49	59	0.765	22.37
50	55	0.765	27.63
51	51	0.760	32.89
52	50	0.765	34.21
53	47	0.750	38.16
54	46	0.750	39.47
55	44	0.755	42.11
56	40	0.740	47.37
57	33	0.735	56.58
58	31	0.735	59.21
59	28	0.740	63.16
60	23	0.705	69.74
61	21	0.710	72.37
62	16	0.715	78.95
63	14	0.710	81.58
65	11	0.690	85.53

We have selected cutoff at 59 manually, as the cutoff selected based on maximum score points to 0 cutoff. Using this cutoff value we gain reduced features subset $X_{PLSR-RENT}$ which we will process through different classifiers and calculate the performance metrics.

4.3 Feature Selection

In addition to PLSR-RENT method to gain new reduced set of features, we have also used two other feature selection methods for comparison. Table 4.3 shows the reduction rates for different methods of feature selections.

The reduction rate of PLSR-RENT is not the highest among the feature selectors, but that is not an indication of its prediction performance.

Additionally, further tests need to be conducted to check and compare the stability of the feature selectors.

Table 4.3: Reduction rates of the different feature selectors

Feature Selector	Reduction Rate
PLSR-RENT	63.16
SelectKBest	55.26
ExtraTrees Feature Importance	69.73

4.4 Hyperparameter Tuning

After gaining new sets of inputs from three different feature selection methods, we use them as input to hypertuned classifiers. All the classifiers are hypertuned to output the best results using *GridSearchCV* as mention in section 3.7. The table 4.4 gives the hyperparameters obtained for different inputs of X.

Table 4.4: Hyperparameters values gained from GridSearchCV

Algorithm	Hyperparameters	Baseline	PLSR-RENT	SelectKBest	ExtraTrees
LogisticRegression	*solver*	*lbfgs*	*lbfgs*	*lbfgs*	*lbfgs*
	penalty	*l2*	*l2*	*l2*	*l2*
	C	0.1	1.0	1.0	1.0
SVC	*kernel*	*rbf*	*rbf*	*rbf*	*rbf*
	gamma	*scale*	*scale*	*scale*	*scale*
	C	50	10	10	10
RandomForestClassifier	*n_estimators*	1000	1000	1000	1000
	max_features	*log2*	*log2*	*log2*	*log2*
KNeighborsClassifier	*n_neighbors*	7	9	7	5
	weights	*distance*	*distance*	*distance*	*distance*
	metric	*euclidean*	*manhattan*	*euclidean*	*euclidean*
PLSR	*n_components*	16	28	16	18

The hyperparameters for *LogisticRegression* and *RandomForestClassifier* do not change across the different subsets of X. The value of C for *SVC*, the $n_neighbors$ for *KNeighborsClassifiers* and the $n_components$ for *PLSR* are varied for different features subsets.

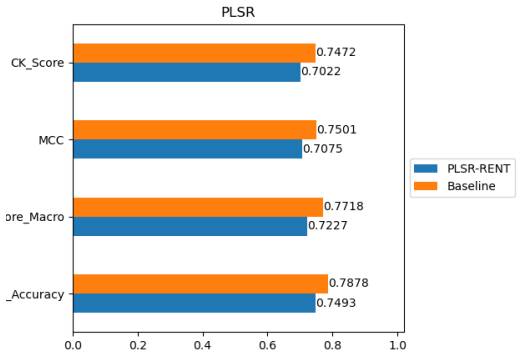
4.5 Classification

We have four sets of X: the original full features set and three reduced subsets. Processing them through the different pipelines for different classifiers is carried out and the results are compared and studied.

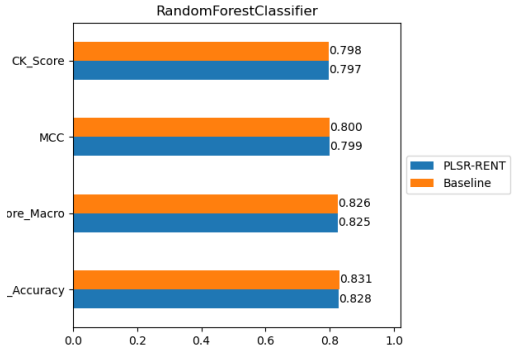
Comparison of PLSR-RENT results with baseline classifier results is done in section 4.5.1 and then with other feature selectors is done in section 4.5.2, respectively.

4.5.1 Comparison with Baseline

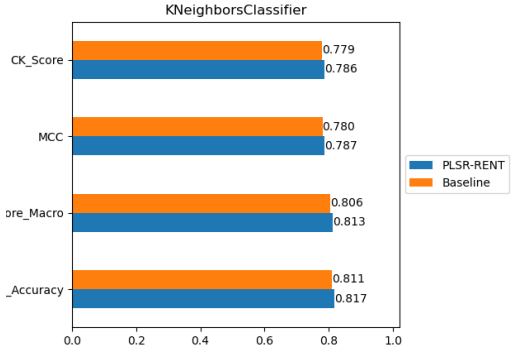
The results gained from the PLSR-RENT pipeline and the baseline pipeline are compared and presented in figure 4.5 below. The figure provides the comparison between the feature subsets for five different classifiers.



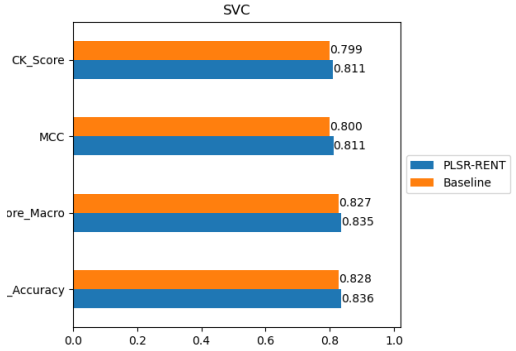
(a) Baseline vs. PLSR-RENT for PLSR



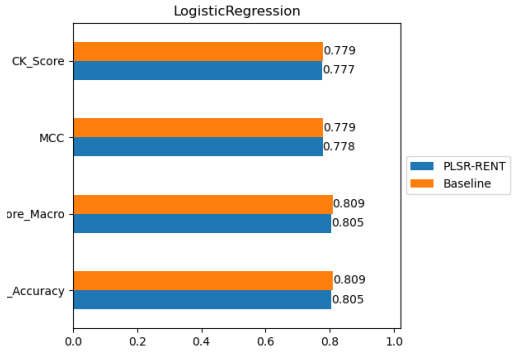
(b) Baseline vs. PLSR-RENT for Random Forest



(c) Baseline vs. PLSR-RENT for KNeighbors



(d) Baseline vs. PLSR-RENT for SVC



(e) Baseline vs. PLSR-RENT for LogisticRegression

Figure 4.5: Comparison results of four different performance metrics for simple preprocessed X and $newX_{PLSR-RENT}$ obtained from PLSR-RENT

For *PLSR* classification model in figure 4.5(a) we observe that the performance of PLSR-RENT is actually not exceeding that of the baseline. This might be due to the manual

selection of the cutoff where we trade-off slight performance for the sake of computation cost. At a reduction rate of 63.16%, the computation cost should be reduced.

Random Forest Classifier comparison in 4.5(b) shows that PLSR-RENT and baseline are almost neck to neck in performance metrics score with only very minuscule difference being in favor of baseline.

In *KNeighborsClassifier* in 4.5(c), the results are also almost similar with slight favor being observed for PLSR-RENT.

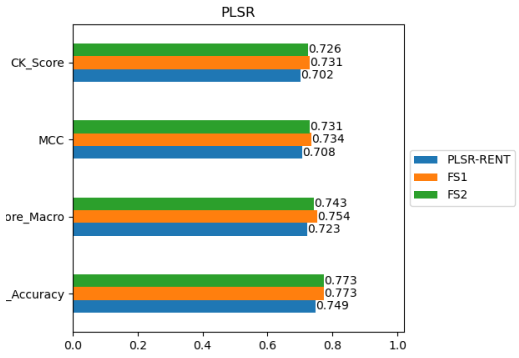
Figure 4.5(d) shows the result comparisons for *SVC* classifier, and we see that PLSR-RENT is also performing slightly better than baseline.

Logistic Regression comparison in figure 4.5(e) also shows almost identical results.

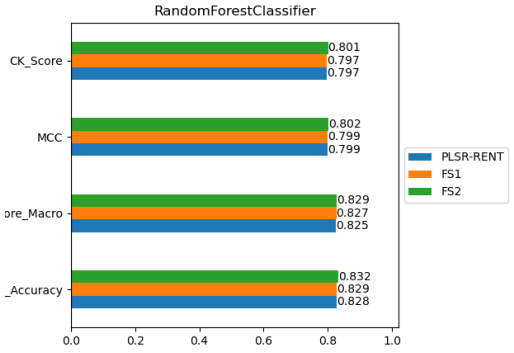
As observed from the results above, we can conclude that when considering just baseline or PLSR-RENT, PLSR-RENT should be considered because while the results might be almost similar, but we have significantly cut down on the number of features.

4.5.2 Comparison with other Feature Selectors

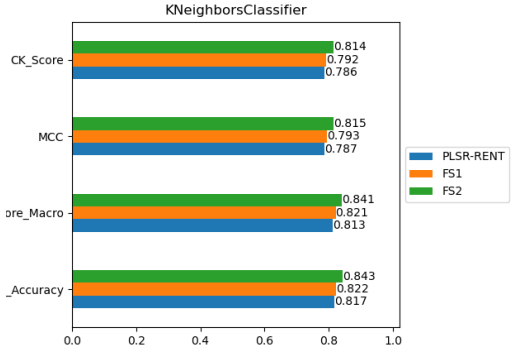
The results obtained from the different subset of features are compared in the figure 4.6. The comparison is carried out for five different classifiers.



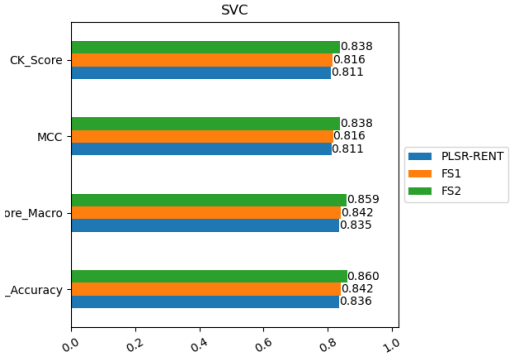
(a) Feature selectors comparison for PLSR



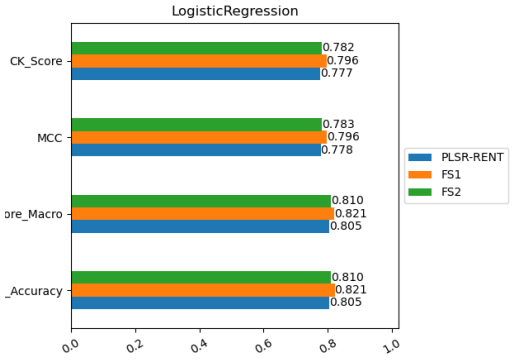
(b) Feature selectors comparison for Random Forest



(c) Feature selectors comparison for KNeighbors



(d) Feature selectors comparison for SVC



(e) Feature selectors comparison for LogisticRegression

Figure 4.6: Comparison results of four different performance metrics for feature selectors: PLSR-RENT, FS1(SelectKBest), FS2(ExtraTrees feature importance)

Figure 4.6(a) shows the comparison for *PLSR* model. The other feature selectors perform better than PLSR-RENT but the difference is just fractional.

Figure 4.6(b) shows the comparison of feature selectors for the model *Random Forest Classifier*. All the feature selectors perform almost the same.

Comparisons for *KNNeighbors Classifier* and *SVC* are visualized in figures 4.6(c) and 4.6(d), respectively. The results show that for both models ExtraTrees feature selector perform the best, while PLSR-RENT and SelectKBest are almost equal and follow ExtraTrees closely.

In figure 4.6(e) the comparison of the results of *Logistic Regression* is shown. SelectKBest feature selector comes out on top with PLSR-RENT and Extra Trees following closely behind.

The results obtained from comparisons of classifiers and feature selectors show that while PLSR-RENT might not be out-performing the other feature selectors, it does come close to them in prediction performance. The difference in the results is not significant.

Chapter 5

Conclusion and Future Directions

Lo! Men have become the tools of their tools.

— Henry David Thoreau

5.1 Conclusion

RENT is a feature selection technique which currently only supports binary classification and regression tasks. The purpose of the thesis was to explore new methodology to extend RENT to be able to handle multi-class classification and regression problems. To this regard we explored the usage of PLSR algorithm as a candidate to extend RENT, as it is capable of handling multi-class classification and regression. PLSR algorithm is also very efficient at handling data which is high-dimensional, multi-colinear and/or having high covariance.

Using cumulative validated explained variance for individual features of PLSR, we created a feature selection method which give us a reduced set of features. We call this method PLSR-RENT.

We can conclude from the results presented in the 4.5.1 that PLSR-RENT does equal or even slightly better than most baseline simple hypertuned classifier models with

significant amount of features reduced. This makes PLSR-RENT seem like a good option regarding low computational cost.

When comparing PLSR-RENT with other feature selectors in section 4.6 we see that though PLSR-RENT does not outperform other feature selectors, the difference is not very significant. This can place PLSR-RENT in the same league as the other feature selectors.

Overall, PLSR-RENT does not outperforms the other feature selection methods already available to us. But tests still need to be conducted to check if PLSR-RENT performs better in cases where PLSR is said to be more efficient, i.e. high-dimensional data. We can also test the candidacy of other algorithms to extend RENT to handle multi-class cases. The direction of the further work and exploration is discussed in the next section.

5.2 Future Work

The current work done in the PLSR-RENT can be extended and/or improved in many different ways. Some of the options for future direction of work is given below:

Data

The results presented in our work are from one dataset which is balanced. There are few other datasets also used for the experimentation whose results are given in Appendix.

It would be insightful to test PLSR-RENT with a dataset which is very high dimensional with multi-colinearity, because that is where the PLSR supposedly performs superior to other models.

Binary and Regression Problems

Conducting tests on binary and regression problems should provide information if PLSR-RENT can give equal or comparable results to multi-class data.

Computation Cost

One of the purpose of feature selection is that it reduces the computation time. Computation cost can be calculated and be a factor in decision making in the future works.

Feature Selection by Number of Features

In most established feature selection methods, user can specify the number of features it needs. We can also implement such functionality in the step 10 of PLSR-RENT workflow shown in figure 2.11.

Optimal Number of Components for PLSR

In the workflow of PLSR-RENT, in step 9 of figure 2.11 we choose the optimal number of components based on the given performance scores. We can choose the number of components based off on the other criteria mentioned in section 2.3.4 to study the effect on the results.

Other Classifiers

The goal of PLSR-RENT was to explore its candidacy for extending RENT to multi-class classification, and eventually regression. We should test other classifiers to conclude if they work better for extending RENT to multi-class cases.

Test of Stability

Although the process carried out in the experimentation with the help of RSKF should, theoretically, result in stable feature selection. But tests need to be conducted to ensure that and then comparison with stability of other feature selectors should be studied.

List of Figures

1.1 Trend of classifiers performance with rising dimensionality. The performance of the classifier increases with increase in dimensionality until it reaches the optimal number of features. After that increasing the dimensionality (without adding samples) only results in decrease in performance of the classifier [4]. 2

2.1 Common dimensionality reduction techniques. 5
2.2 Working of filter methods 6
2.3 Working of wrapper methods 7
2.4 RENT pipeline as explained in [10] 10
2.5 Calculation of weight matrix in RENT with the use of elastic net technique. 11
2.6 The underlying general working of PLSR. The breakdown of X and Y matrices to scores and loadings make up the *outer relations* which are used to calculate *inner relation* that is used to predict Y 13
2.7 Explained variance ratio of 10 principal components 15
2.8 Cumulative explained variance ratio of 10 principal components 15
2.9 Individual cumulative explained variance ratio of 9 principal components . 15
2.10 Choosing number of components according to different criteria 16
2.11 Workflow of the PLSR-RENT 18
2.12 Tables of mean of $X\_cumValExplVar\_indVar$ and performance metrics across k models 19
2.13 Element-wise mean across a dictionary/matrix 20
2.14 Process of comparison of PLSR-RENT with other classifiers 21
2.15 Process of comparison of PLSR-RENT with other feature selectors 21
2.16 Visualization of Stratified K-fold 23
2.17 Visualization of how different kernel functions create different hyperplanes to differentiate classes 27
2.18 Different approaches of SVM for multi-class classification[18] 27
2.19 Visualization of conceptual working of Random Forest 28
2.20 KNN classifier example to classify a point as class plus, circle or triangle with $k = 5$ [19] 30

3.1 Workflow of the experimental setup 38
3.2 Visualization of how one-hot encoding decomposes a single categorical column into multiple columns 40

4.1 Mean of performance metrics across based on 50 models. 49

4.2 Mean of cumulative validated explained variance of individual variables for 1st component. 1st component does not have very high explained variance for all the features. Some features have negative values of cumulative explained variance. 50
4.3 Mean of cumulative validated explained variance of individual variables for 28 components. 28 components are the optimal number of components with regards to the performance metrics. 50
4.4 Performance metrics curves vs cutoff threshold. The performance metrics have the vertical axis to the left, while the number of features (black line) is defined by the vertical axis to the right 52
4.5 Comparison results of four different performance metrics for simple pre-processed X and $newX_{PLSR-RENT}$ obtained from PLSR-RENT 56
4.6 Comparison results of four different performance metrics for feature selectors: PLSR-RENT, FS1(SelectKBest), FS2(ExtraTrees feature importance) 58

A.1 Results for hypertuned classifiers using no feature selection 74
A.2 Results for hypertuned classifiers using PLSR-RENT 75
A.3 Results for hypertuned classifiers using FS1 76
A.4 Results for hypertuned classifiers using FS2 77

B.1 Performance and number of attributes against cutoff threshold 80

List of Tables

3.1 Hyperparameters used for tuning the classification algorithms used in our experimental setup with descriptions and default values 44

4.1 Label encoding of the class values 48
4.2 Sample data of cutoff thresholds and associated number of features, scores and feature reduction rates 53
4.3 Reduction rates of the different feature selectors 54
4.4 Hyperparameters values gained from GridSearchCV 55

B.1 Results of different feature selectors and classifiers 80

Bibliography

[1] Iain M Johnstone and D Michael Titterington. Statistical challenges of high-dimensional data, 2009.

[2] Jianqing Fan and Runze Li. Statistical challenges with high dimensionality: Feature selection in knowledge discovery, 2006. URL https://arxiv.org/abs/math/0602133.

[3] Richard Bellman. *Adaptive Control Processes: A guided tour*. Princeton University Press, 1972.

[4] Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. Feature dimensionality reduction: a review. *Complex & Intelligent Systems*, pages 1–31, 2022.

[5] A. Jović, K. Brkić, and N. Bogunović. A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1200–1205, 2015. doi: 10.1109/MIPRO.2015.7160458.

[6] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research*, 13:27–66, 02 2012.

[7] Alexandros Kalousis, Julien Prados, and Melanie Hilario. Stability of feature selection algorithms. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[8] Utkarsh Mahadeo Khaire and R Dhanalakshmi. Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences*, 2019.

[9] David Dernoncourt, Blaise Hanczar, and Jean-Daniel Zucker. Analysis of feature selection stability on high dimension and small sample data. *Computational Statistics and Data Analysis*, 71:681–693, 2014. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2013.07.012. URL https://www.sciencedirect.com/science/article/pii/S0167947313002570.

[10] Anna Jenul, Stefan Schrunner, Kristian Hovde Liland, Ulf Geir Indahl, Cecilia Marie Futsæther, and Oliver Tomic. Rent—repeated elastic net technique for feature selection. *IEEE Access*, 9:152333–152346, 2021. doi: 10.1109/ACCESS.2021.3126429.

[11] Nicolai Meinshausen and Peter Bühlmann. Stability selection. *Journal of the Royal Statistical Society Series B*, 72:417–473, 09 2010. doi: 10.2307/40802220.

[12] H. Lohninger. Pls - partial least squares regression. URL http://www.statistics4u.com/fundstat_eng/dd_pls.html.

[13] Ajitesh Kumar. Pca explained variance concepts with python example, Aug 2022. URL https://vitalflux.com/pca-explained-variance-concept-python-example/.

[14] William F. Lott. The optimal set of principal component restrictions on a least-squares regression. *Communications in Statistics*, 2(5):449–464, 1973. doi: 10.1080/03610927308827089. URL https://doi.org/10.1080/03610927308827089.

[15] Richard F. Gunst and Robert L. Mason. Biased estimation in regression: An evaluation using mean squared error. *Journal of the American Statistical Association*, 72(359):616–628, 1977. doi: 10.1080/01621459.1977.10480625. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1977.10480625.

[16] Harold Hotelling. The relation of the newer multivariate statistical methods to factor analysis. *British Journal of Statistical Psychology*, 10:69–79, 01 2011. doi: 10.1111/j.2044-8317.1957.tb00179.x.

[17] Loong Chuen Lee, Choong-Yeun Liong, and Abdul Aziz Jemain. Partial least squares-discriminant analysis (pls-da) for classification of high-dimensional (hd) data: a review of contemporary practice strategies and knowledge gaps. *Analyst*, 143:3526–3539, 2018. doi: 10.1039/C8AN00599K. URL http://dx.doi.org/10.1039/C8AN00599K.

[18] Baeldung. Multiclass classification using support vector machines, Nov 2022. URL https://www.baeldung.com/cs/svm-multiclass-classification.

[19] Sebastian Raschka. *Python machine learning*. Packt publishing ltd, 2015.

[20] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45:427–437, 07 2009. doi: 10.1016/j.ipm.2009.03.002.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[24] Oliver Tomic, Thomas Graff, Kristian Hovde Liland, and Tormod Næs. hoggorm: a python library for explorative multivariate statistics. *The Journal of Open Source Software*, 4(39), 2019. doi: 10.21105/joss.00980. URL http://joss.theoj.org/papers/10.21105/joss.00980.

[25] Oliver Tomic. Olivertomic/hoggormplot: Plotting functions for visualisation of data analysis results from the hoggorm package. URL https://github.com/olivertomic/hoggormPlot.

[26] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[27] Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, and Adel Qalieh. mwaskom/seaborn: v0.8.1 (september 2017), September 2017. URL https://doi.org/10.5281/zenodo.883859.

[28] Robert P.W. Duin. Multiple features data set, 1998. URL https://archive.ics.uci.edu/ml/datasets/Multiple+Features.

[29] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Appendix A

Complete Results of the Experimental Setup

In this appendix we will share the complete set of results of our experimental setup. There are a few extra classifiers and performance metrics used.

	Model	Accuracy	Train_Accuracy	Balanced_Accuracy	F1_Score_Macro	F1_Score_Micro	Precision_Macro	Precision_Micro	Recall_Macro	Recall_Micro	MCC	Cohen_Kappa_Score
0	PLSR	0.775000	0.812709	0.775000	0.765366	0.775000	0.772290	0.775000	0.775000	0.775000	0.751840	0.750000
1	LogisticRegression	0.801214	0.905618	0.808972	0.808643	0.801214	0.811088	0.801214	0.808972	0.801214	0.779230	0.778792
2	SVC	0.819423	1.000000	0.827682	0.826768	0.819423	0.830161	0.819423	0.827682	0.819423	0.799867	0.799076
3	DecisionTreeClassifier	0.735964	0.899625	0.739878	0.738162	0.735964	0.737205	0.735964	0.739878	0.735964	0.706153	0.706070
4	RandomForestClassifier	0.819423	1.000000	0.831150	0.828459	0.819423	0.833885	0.819423	0.831150	0.819423	0.800796	0.799169
5	KNeighborsClassifier	0.801214	1.000000	0.810601	0.805791	0.801214	0.808257	0.801214	0.810601	0.801214	0.780178	0.778913
6	AdaBoostClassifier	0.608498	0.658427	0.618444	0.596531	0.608498	0.659166	0.608498	0.618444	0.608498	0.577532	0.565225
7	GaussianNB	0.758725	0.812734	0.762161	0.762951	0.758725	0.769328	0.758725	0.762161	0.758725	0.732161	0.731314

Figure A.1: Results for hypertuned classifiers using no feature selection

	Model	Accuracy	Train_Accuracy	Balanced_Accuracy	F1_Score_Macro	F1_Score_Micro	Precision_Macro	Precision_Micro	Recall_Macro	Recall_Micro	MCC	Cohen_Kappa_Score
0	PLSR	0.775000	0.812709	0.775000	0.765366	0.775000	0.772290	0.775000	0.775000	0.775000	0.751840	0.750000
1	LogisticRegression	0.808801	0.903371	0.814964	0.814323	0.808801	0.816752	0.808801	0.814964	0.808801	0.787719	0.787216
2	SVC	0.819423	1.000000	0.827536	0.827596	0.819423	0.831191	0.819423	0.827536	0.819423	0.799720	0.799035
3	DecisionTreeClassifier	0.746586	0.898876	0.745531	0.743397	0.746586	0.745080	0.746586	0.745531	0.746586	0.718097	0.717549
4	RandomForestClassifier	0.814871	1.000000	0.827903	0.823312	0.814871	0.828589	0.814871	0.827903	0.814871	0.796130	0.794157
5	KNeighborsClassifier	0.805766	1.000000	0.817534	0.811417	0.805766	0.816356	0.805766	0.817534	0.805766	0.786000	0.784056
6	AdaBoostClassifier	0.640364	0.678652	0.642662	0.649935	0.640364	0.694605	0.640364	0.642662	0.640364	0.602408	0.599042
7	GaussianNB	0.763278	0.809738	0.769112	0.768653	0.763278	0.774234	0.763278	0.769112	0.763278	0.737503	0.736514

Figure A.2: Results for hypertuned classifiers using PLSR-RENT

	Model	Accuracy	Train_Accuracy	Balanced_Accuracy	F1_Score_Macro	F1_Score_Micro	Precision_Macro	Precision_Micro	Recall_Macro	Recall_Micro	MCC	Cohen_Kappa_Score
0	PLSR	0.775000	0.804905	0.775000	0.765964	0.775000	0.775040	0.775000	0.775000	0.775000	0.751966	0.750000
1	LogisticRegression	0.819423	0.904120	0.824547	0.824832	0.819423	0.827698	0.819423	0.824547	0.819423	0.799435	0.799053
2	SVC	0.834598	0.991011	0.840598	0.840989	0.834598	0.842911	0.834598	0.840598	0.834598	0.816093	0.815851
3	DecisionTreeClassifier	0.754173	0.896629	0.758511	0.756029	0.754173	0.755288	0.754173	0.758511	0.754173	0.726478	0.726311
4	RandomForestClassifier	0.820941	1.000000	0.831276	0.830356	0.820941	0.834722	0.820941	0.831276	0.820941	0.801845	0.800781
5	KNeighborsClassifier	0.814871	1.000000	0.823929	0.823152	0.814871	0.826773	0.814871	0.823929	0.814871	0.794860	0.794007
6	AdaBoostClassifier	0.608498	0.658427	0.618444	0.596531	0.608498	0.659166	0.608498	0.618444	0.608498	0.577532	0.565225
7	GaussianNB	0.781487	0.812734	0.786169	0.788399	0.781487	0.794081	0.781487	0.786169	0.781487	0.757162	0.756647

Figure A.3: Results for hypertuned classifiers using FS1

	Model	Accuracy	Train_Accuracy	Balanced_Accuracy	F1_Score_Macro	F1_Score_Micro	Precision_Macro	Precision_Micro	Recall_Macro	Recall_Micro	MCC	Cohen_Kappa_Score
0	PLSR	0.745000	0.787625	0.745000	0.725687	0.745000	0.740293	0.745000	0.745000	0.745000	0.720419	0.716667
1	LogisticRegression	0.813354	0.886891	0.818450	0.817900	0.813354	0.819735	0.813354	0.818450	0.813354	0.792643	0.792284
2	SVC	0.854325	0.970037	0.860163	0.859154	0.854325	0.859945	0.854325	0.860163	0.854325	0.838174	0.837864
3	DecisionTreeClassifier	0.755690	0.922846	0.762132	0.758294	0.755690	0.756071	0.755690	0.762132	0.755690	0.728272	0.728072
4	RandomForestClassifier	0.825493	1.000000	0.835829	0.834292	0.825493	0.839262	0.825493	0.835829	0.825493	0.807146	0.805881
5	KNeighborsClassifier	0.813354	0.910112	0.821813	0.821031	0.813354	0.826111	0.813354	0.821813	0.813354	0.793378	0.792317
6	AdaBoostClassifier	0.608498	0.658427	0.618444	0.596531	0.608498	0.659166	0.608498	0.618444	0.608498	0.577532	0.565225
7	GaussianNB	0.769347	0.816479	0.774348	0.775575	0.769347	0.779731	0.769347	0.774348	0.769347	0.743585	0.743156

Figure A.4: Results for hypertuned classifiers using FS2

Appendix B

Results for Dataset MNIST

B.1 Data

The dataset used in this example is **MNIST** dataset available from the scikit-learn library. It is a dataset of handwritten digits. It has 64 features and 1 target column. There are 10 different classes available (0-9). It is an almost balanced dataset. After preprocessing 3 columns were dropped due to low variance threshold, leaving us with 61 feature columns.

B.2 Results

The figure B.1 shows the PLSR-RENT performance metrics and number of attributes against cutoff threshold. We choose cutoff value manually at 90, resulting in 15 features being subtracted, and thus we are left with 46 features out of 61.

Table B.1 presents the results of the experimental setup run with MNIST dataset.

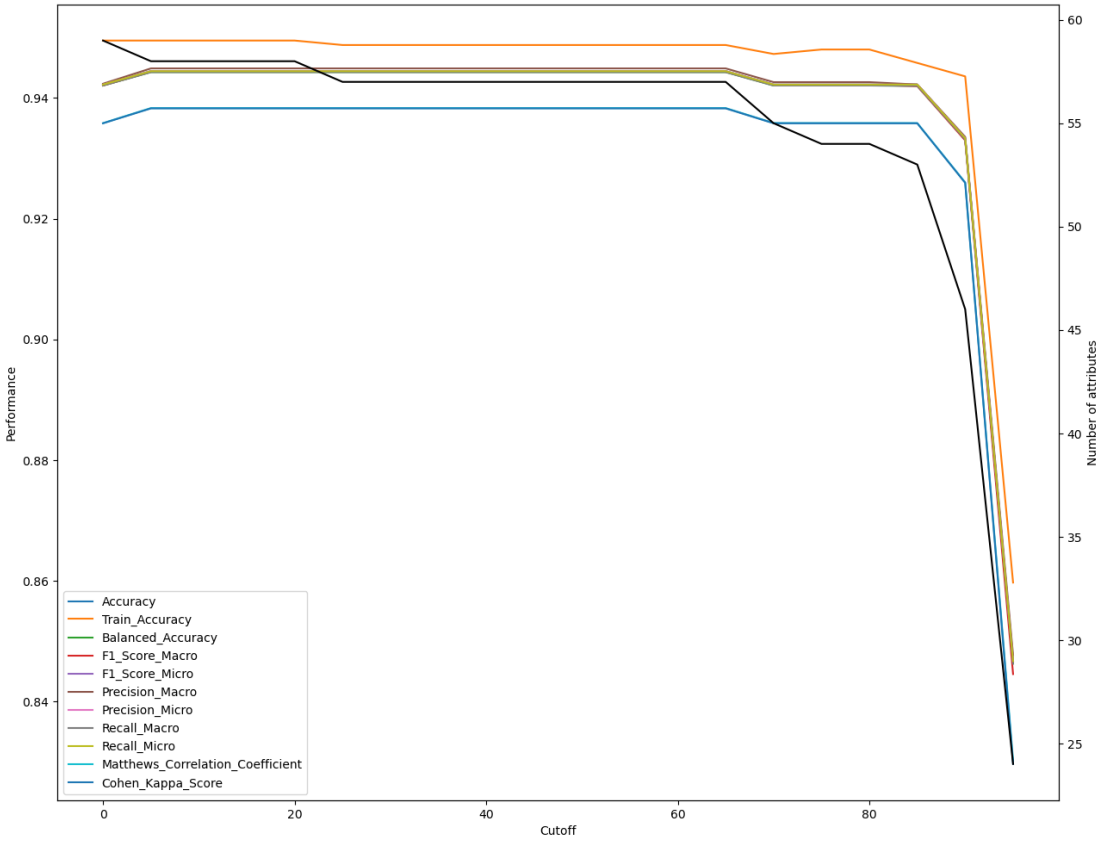
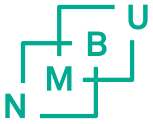


Figure B.1: Performance and number of attributes against cutoff threshold

Table B.1: Results of different feature selectors and classifiers

Feature Selection	Classifier	Reduction Rate	Balanced Accuracy	F1 Score	Cohen's Kappa Score	MCC
Baseline	PLSR	0	0.944286	0.944343	0.938272	0.938328
	Logistic Regression		0.973810	0.973364	0.970032	0.970065
	SVC		0.980109	0.979574	0.977524	0.977555
	RandomForest		0.975167	0.974978	0.971903	0.971933
	KNeighbors		0.973646	0.972566	0.970037	0.970089
PLSR-RENT	PLSR	24.6	0.933123	0.932974	0.925925	0.926006
	Logistic Regression		0.970622	0.970389	0.966284	0.966324
	SVC		0.988389	0.988316	0.986887	0.986903
	RandomForest		0.972016	0.971673	0.968158	0.968216
	KNeighbors		0.986873	0.986868	0.985014	0.985036
SelectKBest	PLSR	3.27	0.942064	0.942052	0.935802	0.935849
	Logistic Regression		0.973842	0.973398	0.970032	0.970084
	SVC		0.982032	0.981499	0.979397	0.979418
	RandomForest		0.975167	0.975187	0.971901	0.971938
	KNeighbors		0.975569	0.974490	0.971910	0.971959
ExtraTrees	PLSR	47.54	0.926458	0.926134	0.918517	0.918724
	Logistic Regression		0.954230	0.953825	0.949429	0.949465
	SVC		0.988632	0.988436	0.986888	0.986907
	RandomForest		0.973572	0.973259	0.970031	0.970062
	KNeighbors		0.989140	0.988609	0.986889	0.986911



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway