Multiple Regression Techniques and Unsupervised Learning for Predicting Urban Land Covers

Jonas Wallnstein and Joshua Chen

December 21, 2023

Contents

1 Introduction

The way in which the surface of a city is distributed has a huge impact on human life in the city. Cities with a large share of impervious surfaces have higher risk of floods as water cannot sink into the soil (Feng et al. 2021), and temperatures are more extreme (Kalnay and Cai, 2003). An important aspect for public policy is therefore knowing how the area in a city is distributed. This is especially important in light of the climate crisis which will increase the frequency and severity of extreme weather events such as floods and high temperature. The distribution of surfaces is referred to as the land cover and is defined as "the composition and characteristics of land surface elements" (Cihlar, 2000). The land cover in an urban area typically includes objects such as asphalt cars and trees. Classification of these objects is vital to assess the share of impervious surfaces in a city and is typically determined by analyzing aerial images. Thus, our outcome variable is the class of an object, which can be one of nine types of urban land cover, for example trees or buildings. The input variables are measures from an aerial image such as the brightness, color or shape of an object. The input data is preprocessed from the pixel level to different "segments", which are explained in the data section.

The goal of this thesis is to classify the urban land cover objects in a setting with a high dimensional feature space. Thus, we use logistic and multinomial regressions with LASSO and Bayesian model selection. Further, we assess the performance of different clustering methods in differentiating the actual number of clusters that correspond to our nine types of land cover. This thesis proceeds with a summary of related work and the data used. Then we use regression and unsupervised learning and finally discuss our methodology and findings.

2 Related Works

Urban land cover classification has been primarily done using machine learning methods such as Support Vector Machines (Johnson, 2013), Extreme Gradient Boosting (Georganos, 2018) and in recent years increasingly Deep Learning Models (Zhang et al., 2018; Helber et al., 2019). The latter can be considered state of the art for land cover classification. While such models typically produce better predictions than our regression framework, they lack any availability for interpretability. An approach similar to ours is done using multinomial logistic regression to classify soil depth in Taiwan by Chan et al. (2019).

3 Dataset

3.1 Overview:

The dataset utilized for the project was found on the UCI Machine Learning Repository. This dataset was donated by Dr. Brian Johnson on March 26th, 2014. Dr. Johnson, a deputy director at the Institute for Global Environmental Strategies, whose projects include mapping urban land covers. This repository contains data from high resolution aerial imagery for classification into nine types of urban land covers with the purpose of "assist[ing] sustainable urban planning efforts".

The data provided is already split into a training and testing set. The testing set was created by random sampling. The designated testing set contains 507 samples and the designated training set contains 168 samples. Because the split is arbitrary, the roles of the training set and testing set were switched to have more data for the models to train on. This results in a 75.11%, 24.89%

train-test split.

The dataset contains no missing values.

The chart on the right contains the breakdown of the training set and testing set by classification as well as further exploratory variables. As will be discussed later in the paper, immediately the issue of class imbalance becomes readily apparent.

	Training	Testing	% Train Set	% of Class
Asphalt	45	14	8.88	76.27
Trees	89	17	17.55	83.97
Grass	83	29	16.37	74.11
Soil	20	14	3.94	58.82
Concrete	93	23	18.34	80.17
Buildings	97	25	19.13	79.51
Cars	21	15	4.14	58.33
Pool	14	15	2.76	48.28
Shadows	45	16	8.88	73.77

Table 1: Breakdown of Data by Class

3.2 Features:

This data comes with a significant amount of preprocessing already. The high resolution aerial imaging has already been converted from pixels to 148 interpretable features. Each feature is repeated seven times - each reflecting a different resolution. The features capture objects' shape, spectral (color), and size. Area is measured in square meters but the spectral and size are broken down into many elements.

Below is the feature legend for the dataset. Suffixes 40, 60, 80, 100, 120, 140 are references to the aforementioned resolution. The descriptions and legend are pulled from the UCI website.

Variable	Description
Class	Land cover class (nominal)
BrdIndx	Border Index (shape variable)
Area	Area in m^2 (size variable)
Round	Roundness (shape variable)
Bright	Brightness (spectral variable)
Compact	Compactness (shape variable)
ShpIndx	Shape Index (shape variable)
Mean_G	Green (spectral variable)
Mean_R	Red (spectral variable)
Mean_NIR	Near Infrared (spectral variable)
SD_G	Standard deviation of Green (texture variable)
SD_R	Standard deviation of Red (texture variable)
SD_NIR	Standard deviation of Near Infrared (texture variable)
LW	Length/Width (shape variable)
GLCM1	Gray-Level Co-occurrence Matrix (texture variable)
Rect	Rectangularity (shape variable)
GLCM2	Another Gray-Level Co-occurrence Matrix attribute (texture variable)
Dens	Density (shape variable)
Assym	Assymetry (shape variable)
NDVI	Normalized Difference Vegetation Index (spectral variable)
BordLngth	Border Length (shape variable)
GLCM3	Another Gray-Level Co-occurrence Matrix attribute (texture variable)

Table 2: Description of Variables

4 Regression

This multi-class regression problem has a heavy emphasis on model selection as very apparent in the ordinary logistic regression section. In his paper, Dr. Brian Johnson addressed this through a blended model, averaging the scores of models trained on the segmented levels of resolution. This seems particularly advantageous towards reducing the high correlation that inevitably arises between corresponding features - such as all the mean values of green. However, for our model selection strategies, we decided to utilize LASSO-BIC, 20-fold LASSO, and Bayesian Model Averaging. As our goal here is purely prediction, explainability and interpretability are secondary concerns.

We considered numerous metrics to assess model performance. Our first consideration was overall accuracy across all classes. However, we also considered application cases where false positives or false positives are particularly egregious. Therefore, we also considered precision, recall, and F1 across each class.

4.1 One vs Rest Regression:

For multi-class classification, we started with an One-vs-Rest approach towards the multi-classification problem. For each class, a model was fitted to create binary classification on predicting for the respective class. For each test sample, all nine models were utilized to predict on it. The highest probability across the nine predictions was then selected as the classification.

	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree
Asphalt	13	0	1	1	0	6	1	0	0
Building	0	22	1	7	0	2	2	2	0
Car	0	0	12	1	0	1	0	1	0
Concrete	0	2	0	11	0	1	1	3	1
Grass	0	1	0	0	22	0	1	3	10
Pool	0	0	0	0	0	5	0	0	0
Shadow	1	0	0	0	2	0	11	0	0
Soil	0	0	1	0	0	0	0	5	0
Tree	0	0	0	0	1	0	0	0	6

Figure 1: Confusion Matrix for Logistic Regression
Overall Accuracy: 0.637 - (0.5593, 0.7096)

Metric	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree	Average
Precision	0.565	0.579	0.800	0.550	0.564	1.00	0.786	0.714	0.857	0.713
Recall	0.929	0.880	0.800	0.478	0.759	0.333	0.688	0.357	0.353	0.620
F1	0.703	0.698	0.800	0.512	0.647	0.500	0.733	0.476	0.500	0.619

Figure 2: Precision, Recall, and F1 Scores for Logistic Regression

4.1.1 Ordinary Logistic Regression:

The first method utilized was with simple logistic regression. The results are seen in the figures above.

This has an overall accuracy of 0.6369 with a 95% confidence interval of (0.5593, 0.7096). Precision, a better metric for indicator of model performance in this unbalanced dataset, indicates poor performance at classification.

Furthermore, glm.fit in R, used for the logistic regressions, returned the warning: "... algorithm did not converge... fitted probabilities numerically 0 or 1 occurred". This indicates extreme overfitting and, quickly predicting on in-sample data indicates this to be the case. Quickly predicting in-sample data returns perfect predictions (1.000 accuracy).

4.1.2 Lasso AIC:

Our first step for addressing this issue was using LASSO BIC and EBIC as a strong model selection tool. However, as typical, AIC was found to be a much more effective metric for purely prediction. We modified the formula from the BIC function (provided to us in seminar one) to produce our AIC OvR models and predictions on the testing set resulted in an overall accuracy of 0.7976 with 95% CI of (0.7288,.8556)

There is clear improvement in performance from simple logistic regression. However, the in-sample prediction was at near-perfect 99.41%, still indicating there to be a significant amount of overfitting. Apparently there is still much room for improvement.

Metric	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree	Average
Precision	0.867	0.657	0.933	0.833	0.676	1.000	0.938	0.818	0.900	0.847
Recall	0.929	0.920	0.933	0.652	0.862	0.733	0.938	0.643	0.529	0.793
F1	0.897	0.767	0.933	0.732	0.758	0.846	0.938	0.720	0.667	0.806

Figure 3: Precision, Recall, and F1 Scoresfor LASSO-AIC

Metric	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree	Average
Precision	1.000	0.733	0.933	0.667	0.806	1.000	0.941	1.000	0.875	0.884
Recall	0.929	0.880	0.933	0.783	0.862	1.000	1.000	0.286	0.824	0.833
F1	0.963	0.800	0.933	0.720	0.833	1.000	0.970	0.444	0.848	0.835

Figure 4: Precision, Recall, and F1 Scores for LASSO-CV

4.1.3 20-Fold Lasso:

With clear room for improvement, we tried LASSO again - this time set by 20 fold cross validation. We see a significant jump in performance across all metrics (see figure 4). Overall accuracy jumped to 0.8393 with 95% CI (0.7749, 0.8913).

4.1.4 Bayesian Model Averaging:

Finally, we employed Bayesian Model Averaging. With the conservative assumption that we do not have good prior knowledge, we chose a Zellner's prior with g=1. We then chose Beta-Binomial(1,1) as our prior on γ. Clearly, from the metrics as seen in figure 5, this was by far the best predictor. The overall average from this model was 0.8571 with 95% CI (0.7949, 0.9063).

While Bayesian Model Averaging produced the best results, it was far too computationally expensive relative to the other models. For reference, to train 9 models and predict both testing and in-sample data took LASSO-AIC an elapsed time of 1.86 seconds and LASSO-CV an elapsed time of 31.80 seconds. To train the singular model for predicting "Asphalt" using BMA took an elapsed time of 581.33 seconds. Training all 9 models would often take upwards of 100 minutes.

4.2 One vs. One

To juxtapose our one vs rest models so far, we implemented a one vs one approach using LASSO-CV. As this approach requires 36 models, BMA would be far too computationally expensive and LASSO-CV has already proven to be similarly effective. Each model will then output one of the

Metric	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree	Average
Precision	1.000	0.733	0.933	0.667	0.806	1.000	0.941	1.000	0.875	0.892
Recall	0.929	0.880	0.933	0.783	0.862	1.000	1.000	0.286	0.824	0.852
F1	0.963	0.800	0.933	0.720	0.833	1.000	0.970	0.444	0.848	0.858

Figure 5: Precision, Recall, and F1 Scores for BMA

	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree
Asphalt	14	1	1	0	0	5	1	0	0
Building	0	23	4	5	0	9	0	6	0
Car	0	0	10	0	0	1	0	0	0
Concrete	0	1	0	17	0	0	0	2	1
Grass	0	0	0	0	26	0	0	2	1
Pool	0	0	0	0	0	0	0	0	0
Shadow	0	0	0	0	0	0	15	0	0
Soil	0	0	0	1	0	0	0	4	0
Tree	0	0	0	0	3	0	0	0	15

Figure 6: Confusion Matrix for OvO with LASSO-CV
Overall Accuracy: 0.7381 - (0.6648, 0.8028)

	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree
Asphalt	13	0	0	0	0	0	1	0	0
Building	0	22	0	6	0	0	0	1	0
Car	0	1	15	0	0	0	0	0	0
Concrete	0	0	0	16	1	0	0	1	1
Grass	0	1	0	0	27	0	0	2	6
Pool	0	0	0	0	0	14	0	0	0
Shadow	1	0	0	0	0	1	15	0	0
Soil	0	1	0	1	0	0	0	10	0
Tree	0	0	0	0	1	0	0	0	10

Figure 7: Confusion Matrix for LASSO-CV/OVR/SMOTE Oversampling
Overall Accuracy: 0.8452 - (0.7815, 0.8963)

two predictions. The final prediction will be the majority agreement of the predictors.

One problem with this methodology is its inability to decide between ties. In R, this will, by default, go to the earlier class. For example, if there are 7 models that predict 'car' and 7 that predict 'grass', the final prediction would be 'car'. This is clearly an issue as seen in the confusion matrix. I did remove all 14 instances with ties from the predictions. While this did boost the accuracy to 0.7922 with a 95% CI (0.7195, 0.8533), this is still considerably worse the OvR LASSO-CV.
Surprisingly, while we expected OvO to be better at dealing with class imbalance, it apparently really struggled with pool and soil - two of the smaller classes.

4.3 SMOTE Oversampling with 20-Fold LASSO CV:

As touched upon previously, class imbalance can be an issue with multi-class problems. One technique we used to address this issue is with SMOTE oversampling. We once again did a One vs Rest regression using a 20 fold Lasso logistic regression. The number of samples was arbitrarily set 900 to achieve around a 50% target classification.

SMOTE Oversampling performed better with an accuracy of 0.8452 and 95% CI (0.7815, 0.8963).

Metric	Asphalt	Building	Car	Concrete	Grass	Pool	Shadow	Soil	Tree	Average
Precision	0.929	0.759	0.938	0.842	0.750	1.000	0.882	0.833	0.909	0.871
Recall	0.929	0.880	1.000	0.696	0.931	0.933	0.938	0.714	0.588	0.845
F1	0.929	0.815	0.968	0.762	0.831	0.966	0.909	0.769	0.714	0.851

Figure 8: Precision, Recall, and F1 Scores for LASSO-CV with SMOTE Oversampling

The mean precision was marginally worse and the mean recall was marginally better. However, the more important metric is the variance of these metrics. Precision had a variance of 0.00694, recall is 0.0201641, and F1 is 0.008895928. For LASSO CV: Precision is 0.0153, recall is 0.0475, and F1 is 0.0298. What this represents is that, without Oversampling, LASSO-CV clearly does a good job with some classes and really poorly with others. However, with Oversampling, the ability of the model to predict any class is much more balanced.

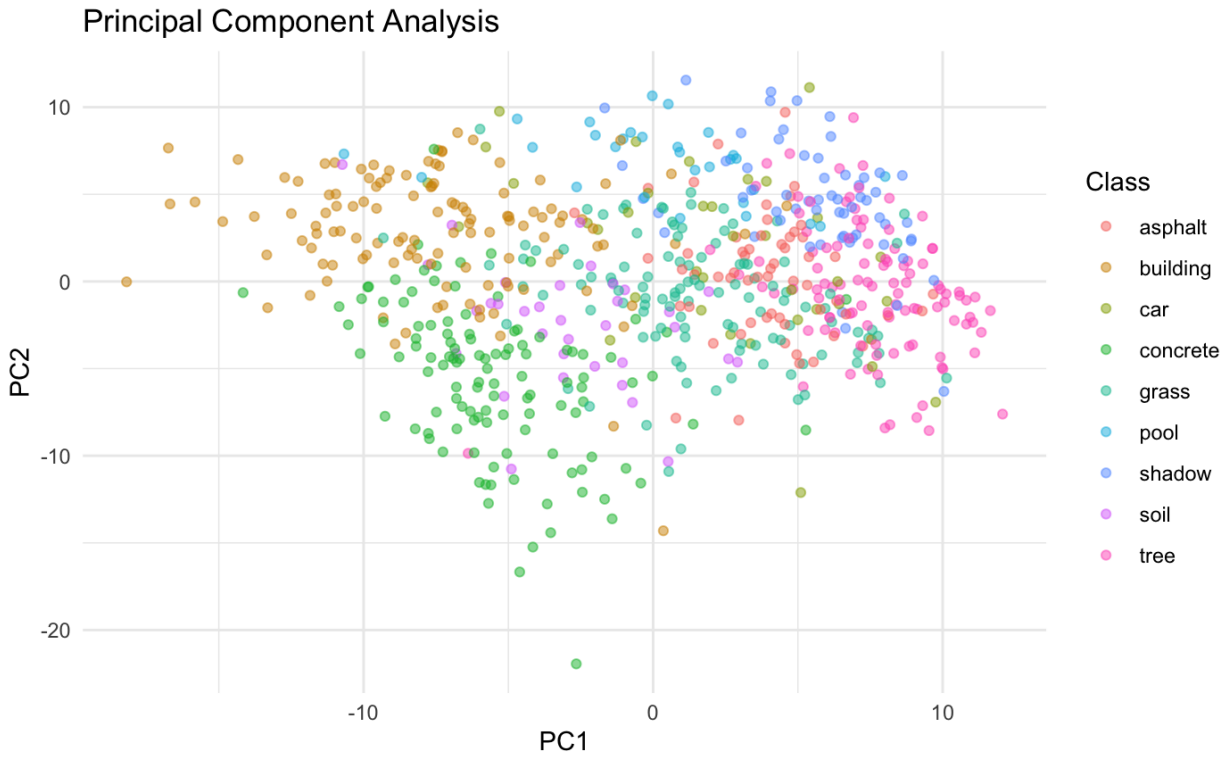
5 Unsupervised Learning

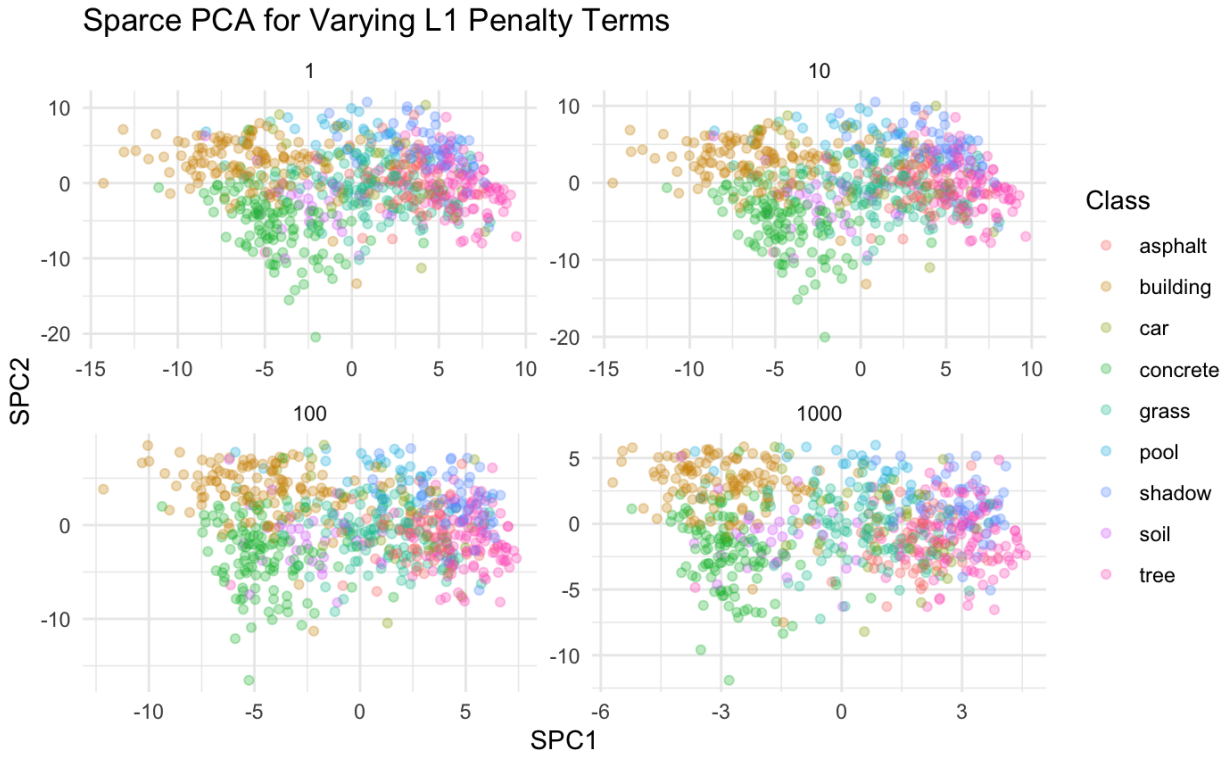
5.1 Overview:

Our goal in this section is to assess whether different clustering algorithms can detect the various types of urban land cover in the data and correctly form the corresponding clusters. However, clustering methods such as Gaussian Mixture Models (GMM) and K-Means tend to perform poorly in high-dimensional settings – often described as the "curse of dimensionality," a term coined by Bellman (1957). This is because, as the number of dimensions increases, distances between points become less distinct, and the relative distance between any two points tends to converge. Moreover, relevant clusters may exist only within certain feature subspaces.
When we attempted to cluster our original high-dimensional dataframe, we encountered these issues. Neither K-means nor GMMs provided a clear delineation of cluster numbers, indicating these algorithms' inability to distinguish between different clusters effectively. Consequently, we proceeded to reduce the data's dimensionality using various methods and compared the results prior to clustering.

5.2 Principal Component Analysis:

A common approach to reduce the dimensionality of data is Principal Component Analysis (PCA). It transforms a set of possibly correlated variables into a smaller number of uncorrelated variables called principal components while retaining the maximum variability of the original data. The first principal component accounts for the greatest possible variance in the data set, with each subsequent component having the highest variance possible under the constraint that it is orthogonal to the preceding components. Plotting the first two principal components, we can observe the general direction in which the different classes differ. However, there is still significant overlap between the classes, which could cause our clustering methods to miss distinctions between the clusters, at least in this two-dimensional representation. Furthermore, PCA may underperform in high-dimensional settings like ours.


Principal Component Analysis
PC2
PC1
10
0
-10
-20
-10
0
10
Class
asphalt
building
car
concrete
grass
pool
shadow
soil
tree


Sparce PCA for Varying L1 Penalty Terms
1
10
100
1000
SPC2
SPC1
Class
asphalt
building
car
concrete
grass
pool
shadow
soil
tree

5.3 Sparse Principal Component Analysis

Our goal with dimensionality reduction is not just to preserve the maximum amount of variance or to minimize the error. Instead, we aim to find a lower-dimensional representation of our data that enhances our ability to discriminate between different types of urban land cover. Therefore, we prioritize identifying principal components that are linear combinations of features crucial for distinguishing between classes, while still preserving as much variance as possible. Unlike standard PCA, which generally utilizes all variables in the dataset to construct each principal component (PC), Sparse PCA constructs PCs from a subset of the original variables. This results in each PC being a linear combination of only a select few variables, achieved through a penalty that enforces sparsity. We employ L1 regularization (similar to Lasso regression), which penalizes the absolute value of the coefficients, introducing sparsity. A penalty strength of 100 has proven effective in separating the clusters in our case, although ideally, this should be confirmed by hyperparameter tuning, such as cross-validation.

5.4 Contrastive Principal Component Analysis

Contrastive PCA extends traditional PCA by focusing on identifying patterns that are particularly prominent in a target dataset when contrasted with a background dataset (Abid et al. 2018). The target dataset is expected to contain distinct patterns or features, whereas the background dataset typically represents general, noisy variability. This background data may be similar to the target but lacks the specific features of interest. In our study, the target dataset consists of our actual data, while the background dataset could be represented by random land cover that does not include our nine classes. Due to the absence of a suitable background dataset, we simulated it using two alternative methods. Firstly, we contrasted features at scales 20 to 60 against those at scales 100 to 140. This approach is based on Johnson's (2013) findings, which indicate that scales 40 to 60 yield the highest classification accuracy, likely because these scales most effectively capture the objects of interest. Conversely, scales 100 to 140, being too coarse, probably represent a mixture of objects and thus can be considered as background data. Secondly, we simulated a background dataset by sampling values from a normal distribution for each column, using the same mean and standard deviation as the corresponding column in the target dataset. However, neither method provided a significant improvement in cluster differentiation over PCA and SPCA. We further investigated whether introducing sparsity, namely applying the sparse contrastive PCA method as outlined by Boileau et al. (2020), would enhance the results, but this too did not lead to better outcomes.

5.5 Clustering

Based on the above results, we found PCA and SPCA to be the most promising candidates for dimensionality reduction and proceeded to cluster. Since the SPCA function in R allows a maximum of nine principal components, we chose to use this number for both PCA and SPCA to enable a fair comparison. With K-means clustering, the optimal number of clusters according to the criterion in Tibshirani and Hastie (2001) is 6 for PCA and 3 for SPCA, which are both below the actual 9 classes. Using GMM, the optimal number of clusters is 8 for PCA and 9 for SPCA, very close to the actual number of clusters. However, comparing the number of clusters with the true number of classes is not a sufficient measure of goodness, as the classes are somewhat artificial and depend on explicit choices. For example, the classes asphalt and concrete are very similar and could be combined into one.

Since we know the true labels, it is insightful to compare the assigned clusters to those obtained using unsupervised learning. To assess the performance of these assignments, we are using two metrics. The Corrected Rand Index (CRI) is an adjusted measure of similarity between two clusterings, correcting the Rand Index for chance agreement, with a range from -1 (no agreement) to 1 (perfect match). The Variation of Information (VI) index quantifies the amount of information lost or gained between two clusterings, with 0 indicating no difference and higher values indicating less similarity. According to both measures, GMM in combination with SPCA dimension reduction performs the best, achieving a score of 0.47 for CRI and 1.74 for VI. The corresponding clusters are not pure in the sense that they perfectly match the nine classes, but they do detect some patterns. For example, one cluster is almost entirely composed of trees and grass, which are indeed similar.

5.6 Discussion:

While we used PCA to reduce dimensionality of our data before clustering, there are more approaches to do clustering in high-dimensionality settings that would have been interesting to explore with more time. While PCA and SPCA globally reduce dimensionality in the dataset, subspace clustering is able to find clusters in different subspaces of a dataset. This would be beneficial for our application as different objects can be identified with different measures. The feature measuring green for example will be more relevant for the trees compared to cars. Further, it would be interesting to investigate how the contrastive PCA methods would perform in this setting with actual background data instead of simulations.

6 Conclusion

This project provided us with the opportunity to employ various model selection tools for a multi-class problem. Effectively addressing overfitting was achieved through model selection using Bayesian Model Averaging (BMA) and LASSO-CV. However, due to class imbalance, many models exhibited inconsistency in predicting classes. The application of SMOTE Oversampling successfully mitigated this issue, yielding more consistent results across classes. Moreover, the One vs. Rest approach proved more effective than One vs. One in this scenario.

Given more time, several additional methodologies could be explored for their effectiveness. Softmax Regression stands out as a potential solution to eliminate the need for multiple models. Bootstrapping and random oversampling could be alternatives to SMOTE Oversampling. Bayesian Model Averaging, when combined with either SMOTE Oversampling or a One vs. One approach, may yield insightful results with sufficient computational resources.

Furthermore, our models operated in a context-independent manner, necessitating the use of F1, Precision, and Recall for evaluation. In hypothetical situations where a false positive carries more significant consequences (emphasizing precision), or when it's preferable to make no prediction rather than an incorrect one, incorporating multiple methods (such as setting a minimum threshold) for probability assessment becomes crucial. Additionally, the paper did not delve into the variables selected by the various methods.

Concerning unsupervised learning we investigated different methods for dimensionality reduction including PCA, sparse PCA, contrastive PCA and sparse contrastive PCA. We then used the reduced dimensionality data to find clusters using the K-means clustering and Gaussian Mixture Models. While we used PCA to reduce dimensionality of our data before clustering, there are

more approaches to do clustering in high-dimensionality settings that would have been interesting to explore with more time. While PCA and SPCA globally reduce dimensionality in the dataset, subspace clustering methods such as CLIQUE are able to find clusters in different subspaces of a dataset. This would be beneficial for our application as different objects can be identified with different measures. The feature measuring green for example will be more relevant for the trees compared to cars. Further, it would be interesting to investigate how the contrastive PCA methods would perform in this setting with actual background data instead of simulations.

7 Bibliography

Abid, A., Zhang, M.J., Bagaria, V.K., Zou, J., 2018. Exploring patterns enriched in a dataset with contrastive principal component analysis. Nat Commun 9, 2134. https://doi.org/10.1038/s41467-018-04608-8

Bellman, R., 1966. Dynamic programming. Science, 153(3731), pp.34-37.

Boileau, P., Hejazi, N.S., Dudoit, S., 2020. Exploring high-dimensional biological data with sparse contrastive principal component analysis. Bioinformatics 36, 3422–3430. https://doi.org/10.1093/bioinformatics/btaa176

Chan, H.C., Chang, C.C., Chen, P.A., Lee, J.T., 2019. Using multinomial logistic regression for prediction of soil depth in an area of complex topography in Taiwan. CATENA 176, 419–429. https://doi.org/10.1016/j.catena.2019.01.030

Cihlar, J., 2000. Land cover mapping of large areas from satellites: status and research priorities. International journal of remote sensing, 21(6-7), pp.1093-1114.

Feng, B., Zhang, Y., Bourke, R., 2021. Urbanization impacts on flood risks based on urban growth data and coupled flood models. Nat Hazards 106, 613–627. https://doi.org/10.1007/s11069-020-04480-0

Georganos, S., Grippa, T., Vanhuysse, S., Lennert, M., Shimoni, M., Wolff, E., 2018. Very High Resolution Object-Based Land Use–Land Cover Urban Classification Using Extreme Gradient Boosting. IEEE Geosci. Remote Sensing Lett. 15, 607–611. https://doi.org/10.1109/LGRS.2018.2803259

Helber, P., Bischke, B., Dengel, A., Borth, D., 2019. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. IEEE J. Sel. Top. Appl. Earth Observations Remote Sensing 12, 2217–2226. https://doi.org/10.1109/JSTARS.2019.2918242

Johnson, B.A., 2013. High-resolution urban land-cover classification using a competitive multi-scale object-based approach. Remote Sensing Letters 4, 131–140. https://doi.org/10.1080/2150704X.2012.705440

Kalnay, E., Cai, M., 2003. Impact of urbanization and land-use change on climate. Nature 423, 528–531. https://doi.org/10.1038/nature01675

Tibshirani, R., Walther, G., Hastie, T., 2001. Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63,

411–423. https://doi.org/10.1111/1467-9868.00293

Zhang, P., Ke, Y., Zhang, Z., Wang, M., Li, P. and Zhang, S., 2018. Urban land use and land cover classification using novel deep learning models based on high spatial resolution satellite imagery. Sensors, 18(11), p.3717.

8 Appendix

Code for Regression Learning

Contents

Loading packages

1 Setup and Exploration:

```
softmax <- function(x) {
    exp_x <- exp(x - max(x))  # Subtracting max(x) for numerical stability
    return(exp_x / sum(exp_x))
}
```

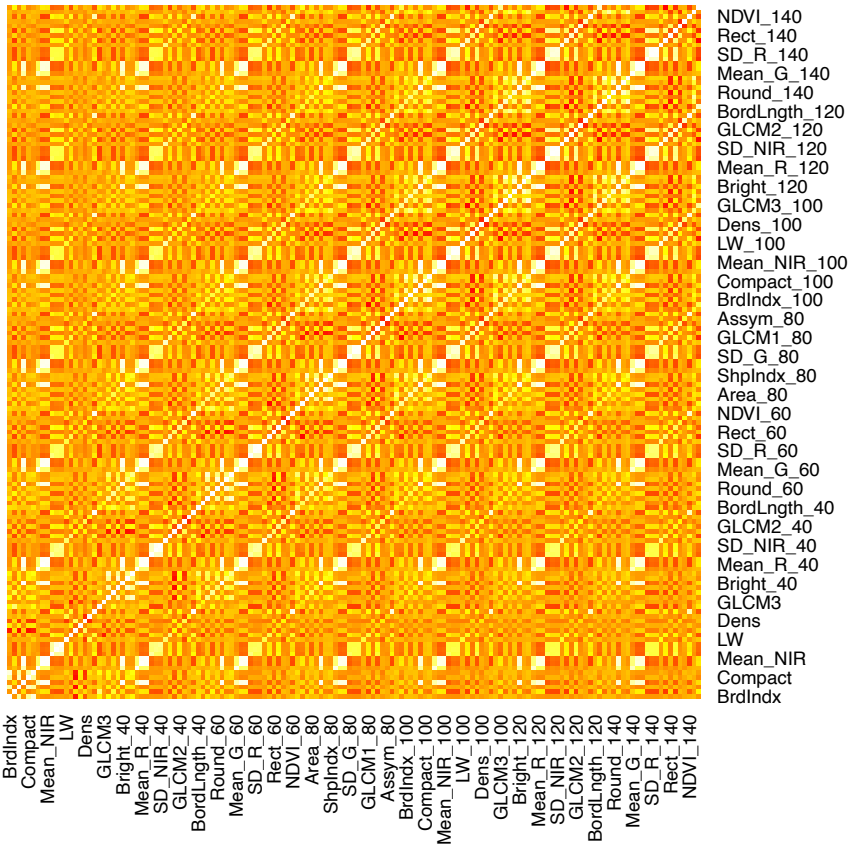
```
scaling_model <- preProcess(X_train, method = c("center", "scale"))
X_train <- predict(scaling_model, newdata = X_train)
X_test <- predict(scaling_model, newdata = X_test)
```

```
classification_percentage <- colSums(y_train_dum)/(colSums(y_test_dum)+colSums(y_train_dum))
classification_percentage
```

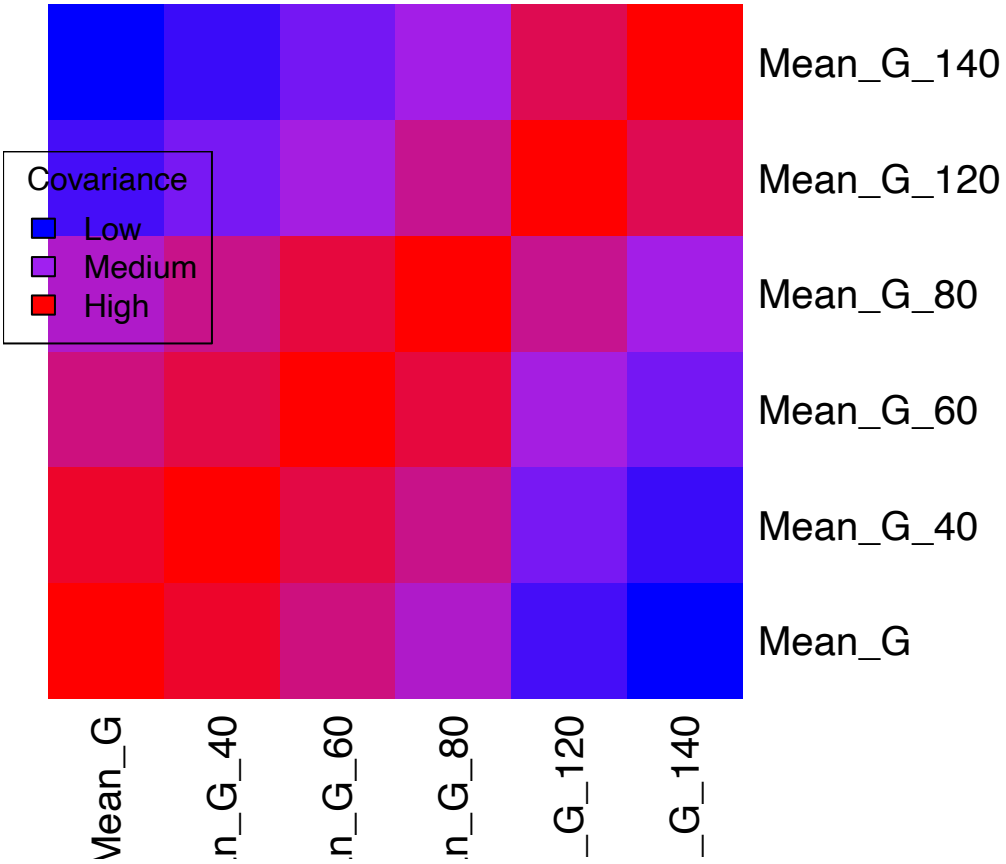
```
##   asphalt  building       car  concrete     grass      pool    shadow      soil
## 0.7627119 0.7950820 0.5833333 0.8017241 0.7410714 0.4827586 0.7377049 0.5882353
##      tree
## 0.8396226
```

Covariance Matrix

```
cov_matrix <- cov(X_train)
heatmap(cov_matrix,
        Rowv = NA,
        Colv = NA,
        col = heat.colors(256),
        scale = "none",
        margins = c(5, 5))
```




```
my_palette <- colorRampPalette(c("blue", "purple", "red"))(256)
cov_matrix <- cov(X_train[,c('Mean_G','Mean_G_40','Mean_G_60','Mean_G_80','Mean_G_120','Mean_G_140')])
heatmap(cov_matrix,
        Rowv = NA,
        Colv = NA,
        col = my_palette,
        scale = "none",
        margins = c(5, 5))
legend("topleft", legend = c("Low", "Medium", "High"), fill = c('blue','purple','red'), title = "Covaria
```



2 One vs Rest Regression

2.1 Simple Logistic Regression

One vs Rest Regression with every single class:

```
set.seed(31415)
class = colnames(y_test_dum)
lr_result = list()

for (c in class) {
  y_train_class <- y_train_dum[[c]]

  model_name <- paste('lr_', c, sep = "")
  X_train_matrix <- as.matrix(X_train)
  assign(model_name, glm(y_train_class ~ ., data = data.frame(cbind(y_train_class, X_train_matrix)), fa

  fit.lr <- get(model_name)
  pred <- predict(fit.lr, newdata = as.data.frame(X_test),  type = "response")

  lr_result[[c]] <- pred
}
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Translation to binary data

```
lr_df <- as.data.frame(lr_result)
colnames(lr_df) = class

lr_df_binary <- apply(lr_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lr_df)
  return(binary_row)
})

lr_df_binary <- as.data.frame(t(lr_df_binary))
head(lr_df_binary)
```

```
##   asphalt building car concrete grass pool shadow soil tree
## 1       0        0   1        0     0    0      0    0    0
## 2       0        0   0        1     0    0      0    0    0
## 3       0        1   0        0     0    0      0    0    0
## 4       0        1   0        0     0    0      0    0    0
## 5       1        0   0        0     0    1      0    1    0
## 6       0        0   0        0     1    0      0    0    0
```

Confusion Matrix

```
max_pred_names <- as.factor(colnames(lr_df_binary)[max.col(lr_df_binary, 'first')])
max_test_names <- as.factor(colnames(y_test_dum)[max.col(y_test_dum, 'first')])
conf_matrix = confusionMatrix(max_pred_names, max_test_names)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       13        0   1        1     1    6      1    0    0
##   building       0       22   1        7     2    2      2    2    0
##   car            0        0  12        1     0    1      0    1    0
##   concrete       0        2   0       11     1    1      1    3    1
##   grass          0        1   0        2    22    0      1    3   10
##   pool           0        0   0        0     0    5      0    0    0
##   shadow         1        0   0        0     2    0     11    0    0
##   soil           0        0   1        1     0    0      0    5    0
##   tree           0        0   0        0     1    0      0    0    6
##
## Overall Statistics
##
##                Accuracy : 0.6369
##                  95% CI : (0.5593, 0.7096)
##     No Information Rate : 0.1726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5838
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 0.92857          0.8800    0.80000          0.47826
## Specificity                 0.93506          0.8881    0.98039          0.93793
## Pos Pred Value              0.56522          0.5789    0.80000          0.55000
## Neg Pred Value              0.99310          0.9769    0.98039          0.91892
## Prevalence                  0.08333          0.1488    0.08929          0.13690
## Detection Rate              0.07738          0.1310    0.07143          0.06548
## Detection Prevalence        0.13690          0.2262    0.08929          0.11905
## Balanced Accuracy           0.93182          0.8841    0.89020          0.70810
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.7586     0.33333       0.68750     0.35714
## Specificity                0.8777     1.00000       0.98026     0.98701
## Pos Pred Value             0.5641     1.00000       0.78571     0.71429
## Neg Pred Value             0.9457     0.93865       0.96753     0.94410
## Prevalence                 0.1726     0.08929       0.09524     0.08333
## Detection Rate             0.1310     0.02976       0.06548     0.02976
## Detection Prevalence       0.2321     0.02976       0.08333     0.04167
## Balanced Accuracy          0.8182     0.66667       0.83388     0.67208
##                      Class: tree
## Sensitivity              0.35294
## Specificity              0.99338
## Pos Pred Value           0.85714
## Neg Pred Value           0.93168
## Prevalence               0.10119
## Detection Rate           0.03571
## Detection Prevalence     0.04167
## Balanced Accuracy        0.67316
```

```
accuracy <- conf_matrix$byClass[, "Balanced Accuracy"]
precision <- conf_matrix$byClass[, "Pos Pred Value"]  # Precision
```

```
recall <- conf_matrix$byClass[, "Sensitivity"]        # Recall
f1_score <- conf_matrix$byClass[, "F1"]
mean(precision)
```

```
## [1] 0.7128234
```

```
mean(recall)
```

```
## [1] 0.6195967
```

```
mean(f1_score)
```

```
## [1] 0.618814
```

2.1.1 In-Sample Predfictions

```
set.seed(31415)
class = colnames(y_test_dum)
lr_insample = list()

for (c in class) {
  y_train_class <- y_train_dum[[c]]

  model_name <- paste('lr_', c, sep = "")
  lr_model <-  get(model_name)
  pred <- predict(lr_model, newdata = as.data.frame(X_train),  type = "response")

  lr_insample[[c]] <- pred
}
```

```
lr_df <- as.data.frame(lr_insample)
colnames(lr_df) = class

lr_df_binary <- apply(lr_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lr_df)
  return(binary_row)
})

lr_in_binary <- as.data.frame(t(lr_df_binary))
head(lr_in_binary)
```

```
##   asphalt building car concrete grass pool shadow soil tree
## 1       0        0   0        1     0    0      0    0    0
## 2       0        0   0        0     0    0      1    0    0
## 3       0        0   0        0     0    0      1    0    0
## 4       0        0   0        0     0    0      0    0    1
## 5       1        0   0        0     0    0      0    0    0
## 6       0        1   0        0     0    0      0    0    0
```

```
max_pred_names <- as.factor(colnames(lr_in_binary)[max.col(lr_in_binary, 'first')])
max_test_names <- as.factor(colnames(y_train_dum)[max.col(y_train_dum, 'first')])
conf_matrix_insample = confusionMatrix(max_pred_names, max_test_names)
conf_matrix_insample
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       45        0   0        0     0    0      0    0    0
##   building       0       97   0        0     0    0      0    0    0
##   car            0        0  21        0     0    0      0    0    0
##   concrete       0        0   0       93     0    0      0    0    0
##   grass          0        0   0        0    83    0      0    0    0
##   pool           0        0   0        0     0   14      0    0    0
##   shadow         0        0   0        0     0    0     45    0    0
##   soil           0        0   0        0     0    0      0   20    0
##   tree           0        0   0        0     0    0      0    0   89
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9928, 1)
##     No Information Rate : 0.1913
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 1.00000          1.0000    1.00000          1.0000
## Specificity                 1.00000          1.0000    1.00000          1.0000
## Pos Pred Value              1.00000          1.0000    1.00000          1.0000
## Neg Pred Value              1.00000          1.0000    1.00000          1.0000
## Prevalence                  0.08876          0.1913    0.04142          0.1834
## Detection Rate              0.08876          0.1913    0.04142          0.1834
## Detection Prevalence        0.08876          0.1913    0.04142          0.1834
## Balanced Accuracy           1.00000          1.0000    1.00000          1.0000
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                1.0000     1.00000       1.00000     1.00000
## Specificity                1.0000     1.00000       1.00000     1.00000
## Pos Pred Value             1.0000     1.00000       1.00000     1.00000
## Neg Pred Value             1.0000     1.00000       1.00000     1.00000
## Prevalence                 0.1637     0.02761       0.08876     0.03945
## Detection Rate             0.1637     0.02761       0.08876     0.03945
## Detection Prevalence       0.1637     0.02761       0.08876     0.03945
## Balanced Accuracy          1.0000     1.00000       1.00000     1.00000
##                      Class: tree
## Sensitivity               1.0000
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            1.0000
## Prevalence                0.1755
## Detection Rate            0.1755
## Detection Prevalence      0.1755
## Balanced Accuracy         1.0000
```

2.2 LASSO-AIC

```
lasso.aic.logistic <- function(y,x,extended=FALSE) {
  require(glmnet)
  fit <- glmnet(x=x,y=y,family='binomial',alpha=1)
  pred <- predict(fit,newx=x,type='response')
  n <- length(y)
  p <- colSums(fit$beta!=0) + 1
  aic <- -2* colSums(y*log(pred)+(1-y)*log(1-pred)) + 2*p
  sel <- which.min(aic)
  beta <- c(fit$a0[sel],fit$beta[,sel]); names(beta)[1]= 'Intercept'
  ypred <- pred[,sel]
  ans <- list(model=fit,coef=beta,ypred=ypred,lambda.opt=fit$lambda[sel],lambda=data.frame(lambda=fit$la
  return(ans)
}
```

```
set.seed(31415)
class = colnames(y_test_dum)
aic_result = list()
aic_insamp = list()

system.time(for (c in class) {
  y_train_class <- y_train_dum[[c]]

  model_name <- paste('lassoaic_', c, sep = "")
  assign(model_name, lasso.aic.logistic(y=y_train_class,x=as.matrix(X_train), extended = FALSE))

  fit.lassoaic <- get(model_name)
  best_lambda <- fit.lassoaic$lambda.opt
  cat("Optimal Lambda for class", c, ":", best_lambda, "\n")

  predictions <- predict(fit.lassoaic$model, newx = X_test, s = best_lambda, type = 'response')
  aicinsample <- predict(fit.lassoaic$model, newx = as.matrix(X_train), s = best_lambda, type = 'respons

  aic_result[[c]] <- predictions
  aic_insamp[[c]] <- aicinsample
})
```

```
## Optimal Lambda for class asphalt : 0.000359828
## Optimal Lambda for class building : 0.0006646005
## Optimal Lambda for class car : 0.0005873631
## Optimal Lambda for class concrete : 0.0002867618
## Optimal Lambda for class grass : 0.0001520269
## Optimal Lambda for class pool : 0.008345434
## Optimal Lambda for class shadow : 0.000607259
## Optimal Lambda for class soil : 8.037127e-05
## Optimal Lambda for class tree : 4.709458e-05
```

```
##    user  system elapsed
##   2.361   0.011   2.375
```

```
aic_df <- as.data.frame(aic_result)
colnames(aic_df) = class

aic_df_binary <- apply(aic_df, 1, function(row) {
```

```
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(aic_df)
  return(binary_row)
})

aic_df_binary <- as.data.frame(t(aic_df_binary))
head(aic_df_binary)
```

```
##   asphalt building car concrete grass pool shadow soil tree
## 1       0        0   1        0     0    0      0    0    0
## 2       0        0   0        1     0    0      0    0    0
## 3       0        1   0        0     0    0      0    0    0
## 4       0        0   0        1     0    0      0    0    0
## 5       0        0   0        0     0    0      0    1    0
## 6       0        0   0        0     1    0      0    0    0
```

```
aicmax_pred_names <- as.factor(colnames(aic_df_binary)[max.col(aic_df_binary, 'first')])
aicmax_test_names <- as.factor(colnames(y_test_dum)[max.col(y_test_dum, 'first')])
aicconf_matrix = confusionMatrix(aicmax_pred_names, aicmax_test_names)
aicconf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       13        0   0        0     0    1      1    0    0
##   building       0       23   1        5     2    3      0    1    0
##   car            0        1  14        0     0    0      0    0    0
##   concrete       0        0   0       15     1    0      0    1    1
##   grass          0        1   0        1    25    0      0    3    7
##   pool           0        0   0        0     0   11      0    0    0
##   shadow         1        0   0        0     0    0     15    0    0
##   soil           0        0   0        2     0    0      0    9    0
##   tree           0        0   0        0     1    0      0    0    9
##
## Overall Statistics
##
##                Accuracy : 0.7976
##                  95% CI : (0.7288, 0.8556)
##     No Information Rate : 0.1726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7686
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 0.92857          0.9200    0.93333         0.65217
## Specificity                 0.98701          0.9161    0.99346         0.97931
## Pos Pred Value              0.86667          0.6571    0.93333         0.83333
## Neg Pred Value              0.99346          0.9850    0.99346         0.94667
## Prevalence                  0.08333          0.1488    0.08929         0.13690
```

```
## Detection Rate              0.07738          0.1369    0.08333         0.08929
## Detection Prevalence        0.08929          0.2083    0.08929         0.10714
## Balanced Accuracy           0.95779          0.9180    0.96340         0.81574
##                       Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                 0.8621     0.73333       0.93750     0.64286
## Specificity                 0.9137     1.00000       0.99342     0.98701
## Pos Pred Value              0.6757     1.00000       0.93750     0.81818
## Neg Pred Value              0.9695     0.97452       0.99342     0.96815
## Prevalence                  0.1726     0.08929       0.09524     0.08333
## Detection Rate              0.1488     0.06548       0.08929     0.05357
## Detection Prevalence        0.2202     0.06548       0.09524     0.06548
## Balanced Accuracy           0.8879     0.86667       0.96546     0.81494
##                       Class: tree
## Sensitivity               0.52941
## Specificity               0.99338
## Pos Pred Value            0.90000
## Neg Pred Value            0.94937
## Prevalence                0.10119
## Detection Rate            0.05357
## Detection Prevalence      0.05952
## Balanced Accuracy         0.76139
```

2.2.1 In-sample prediction

```
accuracy <- aicconf_matrix$byClass[, "Balanced Accuracy"]
precision <- aicconf_matrix$byClass[, "Pos Pred Value"]  # Precision
recall <- aicconf_matrix$byClass[, "Sensitivity"]        # Recall
f1_score <- aicconf_matrix$byClass[, "F1"]
mean(precision)
```

```
## [1] 0.8468704
```

```
mean(recall)
```

```
## [1] 0.79325
```

```
mean(f1_score)
```

```
## [1] 0.8062395
```

```
lassocv_df <- as.data.frame(aic_insamp)
colnames(lassocv_df) = class

lassocv_df_binary <- apply(lassocv_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lassocv_df)
  return(binary_row)
})

lassocv_in_binary <- as.data.frame(t(lassocv_df_binary))

lassomax_pred_names <- as.factor(colnames(lassocv_in_binary)[max.col(lassocv_in_binary, 'first')])
lassomax_test_names <- as.factor(colnames(y_train_dum)[max.col(y_train_dum, 'first')])
insamp_aic_matrix = confusionMatrix(lassomax_pred_names, lassomax_test_names)
insamp_aic_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       45        0   0        0     0    0      0    0    0
##   building       0       97   0        0     1    1      0    0    0
##   car            0        0  21        0     0    0      0    0    0
##   concrete       0        0   0       93     0    0      0    0    0
##   grass          0        0   0        0    82    1      0    0    0
##   pool           0        0   0        0     0   12      0    0    0
##   shadow         0        0   0        0     0    0     45    0    0
##   soil           0        0   0        0     0    0      0   20    0
##   tree           0        0   0        0     0    0      0    0   89
##
## Overall Statistics
##
##                Accuracy : 0.9941
##                  95% CI : (0.9828, 0.9988)
##     No Information Rate : 0.1913
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9931
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 1.00000          1.0000    1.00000          1.0000
## Specificity                 1.00000          0.9951    1.00000          1.0000
## Pos Pred Value              1.00000          0.9798    1.00000          1.0000
## Neg Pred Value              1.00000          1.0000    1.00000          1.0000
## Prevalence                  0.08876          0.1913    0.04142          0.1834
## Detection Rate              0.08876          0.1913    0.04142          0.1834
## Detection Prevalence        0.08876          0.1953    0.04142          0.1834
## Balanced Accuracy           1.00000          0.9976    1.00000          1.0000
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.9880     0.85714       1.00000     1.00000
## Specificity                0.9976     1.00000       1.00000     1.00000
## Pos Pred Value             0.9880     1.00000       1.00000     1.00000
## Neg Pred Value             0.9976     0.99596       1.00000     1.00000
## Prevalence                 0.1637     0.02761       0.08876     0.03945
## Detection Rate             0.1617     0.02367       0.08876     0.03945
## Detection Prevalence       0.1637     0.02367       0.08876     0.03945
## Balanced Accuracy          0.9928     0.92857       1.00000     1.00000
##                      Class: tree
## Sensitivity               1.0000
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            1.0000
## Prevalence                0.1755
## Detection Rate            0.1755
## Detection Prevalence      0.1755
## Balanced Accuracy         1.0000
```

2.3 20 Fold LASSO CV

```
set.seed(31415)
class = colnames(y_test_dum)
lasso_result = list()
lasso_insample = list()

system.time(for (c in class) {
  y_train_class <- y_train_dum[[c]]

  model_name <- paste('lasso_', c, sep = "")
  assign(model_name, cv.glmnet(x = as.matrix(X_train), y = y_train_class, family = "binomial", alpha = 1

  fit.lasso <- get(model_name)
  best_lambda <- fit.lasso$lambda.min
  cat("Optimal Lambda for class", c, ":", best_lambda, "\n")

  predictions <- predict(fit.lasso, newx = as.matrix(X_test), s = best_lambda, type = "response")
  insample_pred <- predict(fit.lasso, newx = as.matrix(X_train), s = best_lambda, type = "response")

  lasso_result[[c]] <- predictions
  lasso_insample[[c]] <- insample_pred
})
```

```
## Optimal Lambda for class asphalt : 0.004436128
## Optimal Lambda for class building : 0.006198086
## Optimal Lambda for class car : 0.002160545
## Optimal Lambda for class concrete : 0.00896336
## Optimal Lambda for class grass : 0.003594659
## Optimal Lambda for class pool : 0.002732754
## Optimal Lambda for class shadow : 0.002952859
## Optimal Lambda for class soil : 0.003644739
## Optimal Lambda for class tree : 0.003098509
```

```
##    user  system elapsed
##  41.606   0.275  41.981
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
lasso_df <- as.data.frame(lasso_result)
colnames(lasso_df) = class

lasso_df_binary <- apply(lasso_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lasso_df)
  return(binary_row)
})

lasso_df_binary <- as.data.frame(t(lasso_df_binary))

lassomax_pred_names <- as.factor(colnames(lasso_df_binary)[max.col(lasso_df_binary, 'first')])
lassomax_test_names <- as.factor(colnames(y_test_dum)[max.col(y_test_dum, 'first')])
lassoconf_matrix = confusionMatrix(lassomax_pred_names, lassomax_test_names)
lassoconf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       13        0   0        0     0    0      0    0    0
##   building       0       22   1        5     0    0      0    2    0
##   car            0        1  14        0     0    0      0    0    0
##   concrete       0        1   0       18     2    0      0    5    1
##   grass          0        1   0        0    25    0      0    3    2
##   pool           0        0   0        0     0   15      0    0    0
##   shadow         1        0   0        0     0    0     16    0    0
##   soil           0        0   0        0     0    0      0    4    0
##   tree           0        0   0        0     2    0      0    0   14
##
## Overall Statistics
##
##                Accuracy : 0.8393
##                  95% CI : (0.7749, 0.8913)
##     No Information Rate : 0.1726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8165
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 0.92857          0.8800    0.93333          0.7826
## Specificity                 1.00000          0.9441    0.99346          0.9379
## Pos Pred Value              1.00000          0.7333    0.93333          0.6667
## Neg Pred Value              0.99355          0.9783    0.99346          0.9645
## Prevalence                  0.08333          0.1488    0.08929          0.1369
## Detection Rate              0.07738          0.1310    0.08333          0.1071
## Detection Prevalence        0.07738          0.1786    0.08929          0.1607
## Balanced Accuracy           0.96429          0.9120    0.96340          0.8603
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.8621     1.00000       1.00000     0.28571
## Specificity                0.9568     1.00000       0.99342     1.00000
## Pos Pred Value             0.8065     1.00000       0.94118     1.00000
## Neg Pred Value             0.9708     1.00000       1.00000     0.93902
## Prevalence                 0.1726     0.08929       0.09524     0.08333
## Detection Rate             0.1488     0.08929       0.09524     0.02381
## Detection Prevalence       0.1845     0.08929       0.10119     0.02381
## Balanced Accuracy          0.9095     1.00000       0.99671     0.64286
##                      Class: tree
## Sensitivity              0.82353
## Specificity              0.98675
## Pos Pred Value           0.87500
## Neg Pred Value           0.98026
## Prevalence               0.10119
## Detection Rate           0.08333
## Detection Prevalence     0.09524
## Balanced Accuracy        0.90514
```

Summary Statistics:

```
accuracy <- lassoconf_matrix$byClass[, "Balanced Accuracy"]
precision <- lassoconf_matrix$byClass[, "Pos Pred Value"]
recall <- lassoconf_matrix$byClass[, "Sensitivity"]
f1_score <- lassoconf_matrix$byClass[, "F1"]
var(precision)
```

```
## [1] 0.01526249
```

```
var(recall)
```

```
## [1] 0.04752289
```

```
var(f1_score)
```

```
## [1] 0.02982246
```

2.3.1 LASSO In-sample

```
lassocv_df <- as.data.frame(lasso_insample)
colnames(lassocv_df) = class

lassocv_df_binary <- apply(lassocv_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lassocv_df)
  return(binary_row)
})

lassocv_in_binary <- as.data.frame(t(lassocv_df_binary))

lassomax_pred_names <- as.factor(colnames(lassocv_in_binary)[max.col(lassocv_in_binary, 'first')])
lassomax_test_names <- as.factor(colnames(y_train_dum)[max.col(y_train_dum, 'first')])
insamp_lasso_matrix = confusionMatrix(lassomax_pred_names, lassomax_test_names)
insamp_lasso_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       43        2   0        0     0    0      2    0    0
##   building       0       89   0        8     1    1      0    2    0
##   car            0        0  20        0     0    0      0    0    0
##   concrete       0        6   1       84     0    0      0    2    0
##   grass          0        0   0        1    74    1      0    4    2
##   pool           0        0   0        0     0   12      0    0    0
##   shadow         2        0   0        0     0    0     43    0    1
##   soil           0        0   0        0     0    0      0   12    0
##   tree           0        0   0        0     8    0      0    0   86
##
## Overall Statistics
##
##                Accuracy : 0.9132
##                  95% CI : (0.8852, 0.9362)
##     No Information Rate : 0.1913
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.8979
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 0.95556          0.9175    0.95238          0.9032
## Specificity                 0.99134          0.9707    1.00000          0.9783
## Pos Pred Value              0.91489          0.8812    1.00000          0.9032
## Neg Pred Value              0.99565          0.9803    0.99795          0.9783
## Prevalence                  0.08876          0.1913    0.04142          0.1834
## Detection Rate              0.08481          0.1755    0.03945          0.1657
## Detection Prevalence        0.09270          0.1992    0.03945          0.1834
## Balanced Accuracy           0.97345          0.9441    0.97619          0.9407
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.8916     0.85714       0.95556     0.60000
## Specificity                0.9811     1.00000       0.99351     1.00000
## Pos Pred Value             0.9024     1.00000       0.93478     1.00000
## Neg Pred Value             0.9788     0.99596       0.99566     0.98384
## Prevalence                 0.1637     0.02761       0.08876     0.03945
## Detection Rate             0.1460     0.02367       0.08481     0.02367
## Detection Prevalence       0.1617     0.02367       0.09073     0.02367
## Balanced Accuracy          0.9363     0.92857       0.97453     0.80000
##                      Class: tree
## Sensitivity               0.9663
## Specificity               0.9809
## Pos Pred Value            0.9149
## Neg Pred Value            0.9927
## Prevalence                0.1755
## Detection Rate            0.1696
## Detection Prevalence      0.1854
## Balanced Accuracy         0.9736
```

2.4 Bayesian Model Selection

```
bayes_results = list()

system.time(for (c in class) {
  y_train_class <- y_train_dum[[c]]

  model_name <- paste('bayes_', c, sep = "")
  assign(model_name, modelSelection(y_train_class ~ ., data=X_train, priorCoef = zellnerprior(taustd =
  print(paste(c,":", " model fitted", sep = ""))
  fit.bayes <- get(model_name)

  predictions <- predict(fit.bayes, newdata = X_test,data = X_train, type= 'response')
  #predictions <- 1/(1 + e^-predictions)

  bayes_results[[c]] <- predictions
})
```

2.4.1 Convergence Plots:

```
if (!file.exists("conv_plots")) {
  dir.create("conv_plots")
}
```

```
margppest= matrix(NA,nrow=nrow(bayes_asphalt$postSample),ncol=ncol(bayes_asphalt$postSample))
for (j in 1:ncol(bayes_asphalt$postSample)) {
  margppest[,j]= cumsum(bayes_asphalt$postSample[,j])/(1:nrow(bayes_asphalt$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest = matrix(NA,nrow=nrow(bayes_building$postSample),ncol=ncol(bayes_building$postSample))
for (j in 1:ncol(bayes_building$postSample)) {
  margppest[,j]= cumsum(bayes_building$postSample[,j])/(1:nrow(bayes_building$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_car$postSample),ncol=ncol(bayes_car$postSample))
for (j in 1:ncol(bayes_car$postSample)) {
  margppest[,j]= cumsum(bayes_car$postSample[,j])/(1:nrow(bayes_car$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_concrete$postSample),ncol=ncol(bayes_concrete$postSample))
for (j in 1:ncol(bayes_concrete$postSample)) {
  margppest[,j]= cumsum(bayes_concrete$postSample[,j])/(1:nrow(bayes_concrete$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_grass$postSample),ncol=ncol(bayes_grass$postSample))
for (j in 1:ncol(bayes_grass$postSample)) {
  margppest[,j]= cumsum(bayes_grass$postSample[,j])/(1:nrow(bayes_grass$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_pool$postSample),ncol=ncol(bayes_pool$postSample))
for (j in 1:ncol(bayes_pool$postSample)) {
  margppest[,j]= cumsum(bayes_pool$postSample[,j])/(1:nrow(bayes_pool$postSample))
}
```

```
par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_tree$postSample),ncol=ncol(bayes_tree$postSample))
for (j in 1:ncol(bayes_tree$postSample)) {
  margppest[,j]= cumsum(bayes_tree$postSample[,j])/(1:nrow(bayes_tree$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(bayes_soil$postSample),ncol=ncol(bayes_soil$postSample))
for (j in 1:ncol(bayes_soil$postSample)) {
  margppest[,j]= cumsum(bayes_soil$postSample[,j])/(1:nrow(bayes_soil$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
margppest= matrix(NA,nrow=nrow(hi$postSample),ncol=ncol(hi$postSample))
for (j in 1:ncol(hi$postSample)) {
  margppest[,j]= cumsum(hi$postSample[,j])/(1:nrow(hi$postSample))
}

par(mar=c(4,5,1,1), cex.lab=1, cex.axis=1)
plot(margppest[,1], type='l', ylim=c(0,1), xlab='Gibbs iteration', ylab='Estimated P(gamma_j=1 | y)', ma
for (j in 2:ncol(margppest)) lines(margppest[,j])
```

```
bayes_means <- lapply(bayes_results, function(df) df[, 1])
bayes_results_df <- data.frame(bayes_means)
head(bayes_results_df)
```

```
bayes_df_binary <- apply(bayes_results_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(bayes_results_df)
  return(binary_row)
})

bayes_df_binary <- as.data.frame(t(bayes_df_binary))
head(bayes_df_binary)
```

Confusion Matrix:

```
bayesmax_pred_names <- as.factor(colnames(lasso_df_binary)[max.col(bayes_df_binary, 'first')])
bayesmax_test_names <- as.factor(colnames(y_test_dum)[max.col(y_test_dum, 'first')])
bayes_conf_matrix = confusionMatrix(bayesmax_pred_names, bayesmax_test_names)
bayes_conf_matrix
```

```
accuracy <- bayes_conf_matrix$byClass[, "Balanced Accuracy"]
precision <- bayes_conf_matrix$byClass[, "Pos Pred Value"]
recall <- bayes_conf_matrix$byClass[, "Sensitivity"]
f1_score <- bayes_conf_matrix$byClass[, "F1"]
```

```
mean(precision)
mean(recall)
mean(f1_score)
```

3 One-vs-One Regression

```
y_train <- training[,1]
y_test <- testing[,1]
```

```
models <- list()

class_pairs <- combn(class, 2, simplify = TRUE)

for (i in seq_len(ncol(class_pairs))) {
  pair <- class_pairs[, i]

  first_class <- pair[1]
  second_class <- pair[2]
  wanted_classes <- y_train_dum[, first_class] | y_train_dum[, second_class]
  train_subset <- X_train[wanted_classes, ]
  y_train_subset <- y_train_dum[wanted_classes, first_class]


  model <- cv.glmnet(x = as.matrix(train_subset),
                     y = as.numeric(as.matrix(y_train_subset)),
                     family = "binomial",
                     alpha = 1,
                     nfolds = 20)

  models[[paste(pair, collapse = "_vs_")]] <- model
}
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
predictions <- data.frame(matrix(NA, nrow = nrow(X_test), ncol = 0))

for (i in seq_len(length(models))) {
  pair <- names(models)[i]
  model <- models[[i]]
```

```
  predictions[pair] <- predict(model, newx = as.matrix(X_test), type = 'response')
  classes  <- strsplit(pair, "_vs_")[[1]]
  predictions[pair] <- ifelse(predictions[pair]>0.5,classes[1],classes[2])
}
```

```
final_predictions <- apply(predictions, 1, function(row) {
  names(which.max(table(unlist(row))))
})

ovo_conf_mat <- confusionMatrix(data = as.factor(final_predictions), as.factor(y_test$class))
```

```
## Warning in levels(reference) != levels(data): longer object length is not a
## multiple of shorter object length
```

```
## Warning in confusionMatrix.default(data = as.factor(final_predictions), :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
ovo_conf_mat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       14        1   1        0     0    5      1    0    0
##   building       0       23   4        5     0    9      0    6    0
##   car            0        0  10        0     0    1      0    0    0
##   concrete       0        1   0       17     0    0      0    2    1
##   grass          0        0   0        0    26    0      0    2    1
##   pool           0        0   0        0     0    0      0    0    0
##   shadow         0        0   0        0     0    0     15    0    0
##   soil           0        0   0        1     0    0      0    4    0
##   tree           0        0   0        0     3    0      0    0   15
##
## Overall Statistics
##
##                Accuracy : 0.7381
##                  95% CI : (0.6648, 0.8028)
##     No Information Rate : 0.1726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 1.00000          0.9200    0.66667          0.7391
## Specificity                 0.94805          0.8322    0.99346          0.9724
## Pos Pred Value              0.63636          0.4894    0.90909          0.8095
## Neg Pred Value              1.00000          0.9835    0.96815          0.9592
## Prevalence                  0.08333          0.1488    0.08929          0.1369
## Detection Rate              0.08333          0.1369    0.05952          0.1012
## Detection Prevalence        0.13095          0.2798    0.06548          0.1250
```

```
## Balanced Accuracy           0.97403          0.8761    0.83007          0.8558
##                       Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                 0.8966     0.00000       0.93750     0.28571
## Specificity                 0.9784     1.00000       1.00000     0.99351
## Pos Pred Value              0.8966         NaN       1.00000     0.80000
## Neg Pred Value              0.9784     0.91071       0.99346     0.93865
## Prevalence                  0.1726     0.08929       0.09524     0.08333
## Detection Rate              0.1548     0.00000       0.08929     0.02381
## Detection Prevalence        0.1726     0.00000       0.08929     0.02976
## Balanced Accuracy           0.9375     0.50000       0.96875     0.63961
##                       Class: tree
## Sensitivity               0.88235
## Specificity               0.98013
## Pos Pred Value            0.83333
## Neg Pred Value            0.98667
## Prevalence                0.10119
## Detection Rate            0.08929
## Detection Prevalence      0.10714
## Balanced Accuracy         0.93124
```

Without ties:

```
tied_rows <- list()

for (i in 1:nrow(predictions)) {
  predictions_row <- table(unlist(predictions[i,]))
  top_predict <- order(-predictions_row)
  if (as.integer(predictions_row[top_predict[1]]) == as.integer(predictions_row[top_predict[2]])) {
    tied_rows <- c(tied_rows, list(i))
  }
}
```

```
fp_noties <- final_predictions[-unlist(tied_rows)]
ref_noties <- y_test$class[-unlist(tied_rows)]

ovo_conf_mat <- confusionMatrix(data = as.factor(fp_noties), as.factor(ref_noties))
```

```
## Warning in levels(reference) != levels(data): longer object length is not a
## multiple of shorter object length
```

```
## Warning in confusionMatrix.default(data = as.factor(fp_noties),
## as.factor(ref_noties)): Levels are not in the same order for reference and
## data. Refactoring data to match.
```

```
ovo_conf_mat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       14        1   0        0     0    3      1    0    0
##   building       0       23   2        5     0    7      0    2    0
##   car            0        0   8        0     0    1      0    0    0
##   concrete       0        1   0       17     0    0      0    1    1
##   grass          0        0   0        0    26    0      0    2    1
##   pool           0        0   0        0     0    0      0    0    0
##   shadow         0        0   0        0     0    0     15    0    0
```

```
##   soil           0        0   0        1     0    0      0    4    0
##   tree           0        0   0        0     3    0      0    0   15
##
## Overall Statistics
##
##                Accuracy : 0.7922
##                  95% CI : (0.7195, 0.8533)
##     No Information Rate : 0.1883
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7595
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 1.00000          0.9200    0.80000          0.7391
## Specificity                 0.96429          0.8760    0.99306          0.9771
## Pos Pred Value              0.73684          0.5897    0.88889          0.8500
## Neg Pred Value              1.00000          0.9826    0.98621          0.9552
## Prevalence                  0.09091          0.1623    0.06494          0.1494
## Detection Rate              0.09091          0.1494    0.05195          0.1104
## Detection Prevalence        0.12338          0.2532    0.05844          0.1299
## Balanced Accuracy           0.98214          0.8980    0.89653          0.8581
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.8966     0.00000        0.9375     0.44444
## Specificity                0.9760     1.00000        1.0000     0.99310
## Pos Pred Value             0.8966         NaN        1.0000     0.80000
## Neg Pred Value             0.9760     0.92857        0.9928     0.96644
## Prevalence                 0.1883     0.07143        0.1039     0.05844
## Detection Rate             0.1688     0.00000        0.0974     0.02597
## Detection Prevalence       0.1883     0.00000        0.0974     0.03247
## Balanced Accuracy          0.9363     0.50000        0.9688     0.71877
##                      Class: tree
## Sensitivity               0.8824
## Specificity               0.9781
## Pos Pred Value            0.8333
## Neg Pred Value            0.9853
## Prevalence                0.1104
## Detection Rate            0.0974
## Detection Prevalence      0.1169
## Balanced Accuracy         0.9302
```

4 SMOTE Oversampling:

```r
for (c in class) {
  y_train_class <- y_train_dum[[c]]
  oversampled_data <- ovun.sample(y_train_class ~ ., data = cbind(X_train, y_train_class), method = "ov

  X_train_oversampled <- oversampled_data[, -which(names(oversampled_data) == "y_train_class")]
  y_train_oversampled <- oversampled_data$y_train_class
```

```
  model_name <- paste('lasso_', c, '_oversampled', sep = "")
  assign(model_name, cv.glmnet(x = as.matrix(X_train_oversampled), y = y_train_oversampled, family = "bi

  fit_lasso_oversampled <- get(model_name)
  best_lambda_oversampled <- fit_lasso_oversampled$lambda.min
  cat("Optimal Lambda for class", c, "with SMOTE oversampling:", best_lambda_oversampled, "\n")

  predictions <- predict(fit_lasso_oversampled, newx = as.matrix(X_test), s = best_lambda_oversampled, t

  lasso_result[[c]] <- predictions
}
```

```
## Optimal Lambda for class asphalt with SMOTE oversampling: 0.0007257957
## Optimal Lambda for class building with SMOTE oversampling: 0.001289831
## Optimal Lambda for class car with SMOTE oversampling: 0.0001348818
## Optimal Lambda for class concrete with SMOTE oversampling: 0.001398017
## Optimal Lambda for class grass with SMOTE oversampling: 0.002417456
## Optimal Lambda for class pool with SMOTE oversampling: 0.000456565
## Optimal Lambda for class shadow with SMOTE oversampling: 0.0004461372
## Optimal Lambda for class soil with SMOTE oversampling: 0.0003669641
## Optimal Lambda for class tree with SMOTE oversampling: 0.001304967
```

```
lasso_df <- as.data.frame(lasso_result)
colnames(lasso_df) = class

lasso_df_binary <- apply(lasso_df, 1, function(row) {
  binary_row <- as.numeric(row == max(row))
  names(binary_row) <- colnames(lasso_df)
  return(binary_row)
})

lasso_df_binary <- as.data.frame(t(lasso_df_binary))

lassomax_pred_names <- as.factor(colnames(lasso_df_binary)[max.col(lasso_df_binary, 'first')])
lassomax_test_names <- as.factor(colnames(y_test_dum)[max.col(y_test_dum, 'first')])
lasso_oversamp_matrix = confusionMatrix(lassomax_pred_names, lassomax_test_names)
lasso_oversamp_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction asphalt building car concrete grass pool shadow soil tree
##   asphalt       13        0   0        0     0    0      1    0    0
##   building       0       22   0        6     0    0      0    1    0
##   car            0        1  15        0     0    0      0    0    0
##   concrete       0        0   0       16     1    0      0    1    1
##   grass          0        1   0        0    27    0      0    2    6
##   pool           0        0   0        0     0   14      0    0    0
##   shadow         1        0   0        0     0    1     15    0    0
##   soil           0        1   0        1     0    0      0   10    0
##   tree           0        0   0        0     1    0      0    0   10
##
## Overall Statistics
##
```

```
##                Accuracy : 0.8452
##                  95% CI : (0.7815, 0.8963)
##     No Information Rate : 0.1726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8235
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: asphalt Class: building Class: car Class: concrete
## Sensitivity                 0.92857          0.8800    1.00000         0.69565
## Specificity                 0.99351          0.9510    0.99346         0.97931
## Pos Pred Value              0.92857          0.7586    0.93750         0.84211
## Neg Pred Value              0.99351          0.9784    1.00000         0.95302
## Prevalence                  0.08333          0.1488    0.08929         0.13690
## Detection Rate              0.07738          0.1310    0.08929         0.09524
## Detection Prevalence        0.08333          0.1726    0.09524         0.11310
## Balanced Accuracy           0.96104          0.9155    0.99673         0.83748
##                      Class: grass Class: pool Class: shadow Class: soil
## Sensitivity                0.9310     0.93333       0.93750     0.71429
## Specificity                0.9353     1.00000       0.98684     0.98701
## Pos Pred Value             0.7500     1.00000       0.88235     0.83333
## Neg Pred Value             0.9848     0.99351       0.99338     0.97436
## Prevalence                 0.1726     0.08929       0.09524     0.08333
## Detection Rate             0.1607     0.08333       0.08929     0.05952
## Detection Prevalence       0.2143     0.08333       0.10119     0.07143
## Balanced Accuracy          0.9331     0.96667       0.96217     0.85065
##                      Class: tree
## Sensitivity              0.58824
## Specificity              0.99338
## Pos Pred Value           0.90909
## Neg Pred Value           0.95541
## Prevalence               0.10119
## Detection Rate           0.05952
## Detection Prevalence     0.06548
## Balanced Accuracy        0.79081
```

```
accuracy <- lasso_oversamp_matrix$byClass[, "Balanced Accuracy"]
precision <- lasso_oversamp_matrix$byClass[, "Pos Pred Value"]
recall <- lasso_oversamp_matrix$byClass[, "Sensitivity"]
f1_score <- lasso_oversamp_matrix$byClass[, "F1"]
var(precision)
```

```
## [1] 0.006935073
```

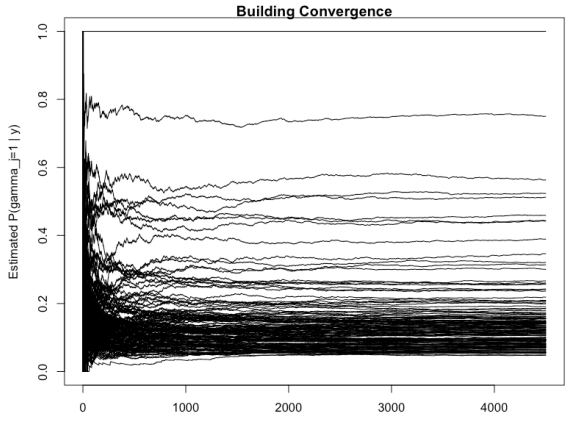
```
var(recall)
```

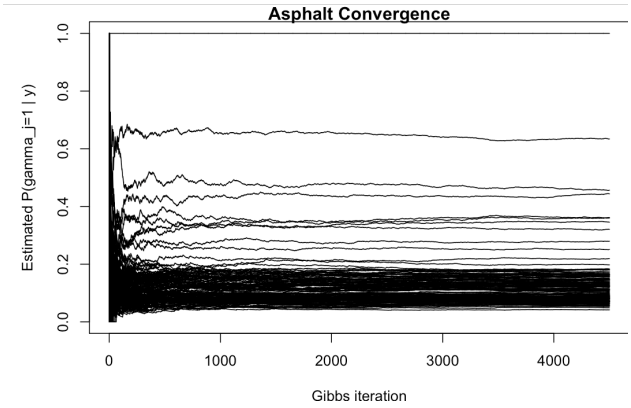
```
## [1] 0.0201641
```

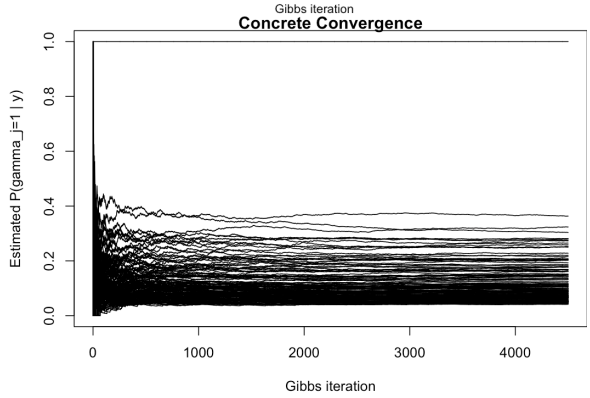
```
var(f1_score)
```

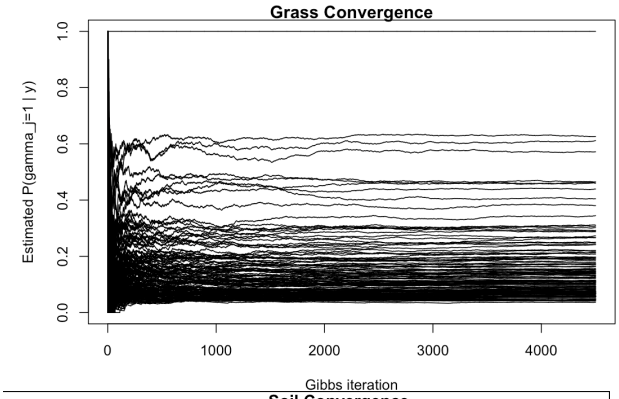
```
## [1] 0.008895928
```

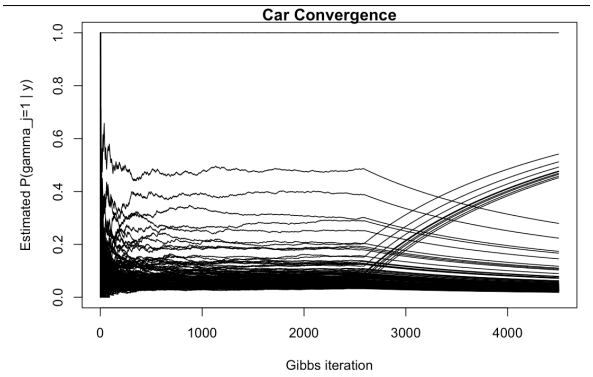
BMA – Convergence Graphs

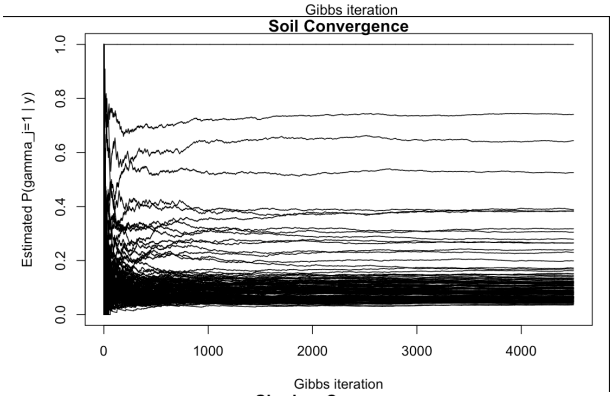


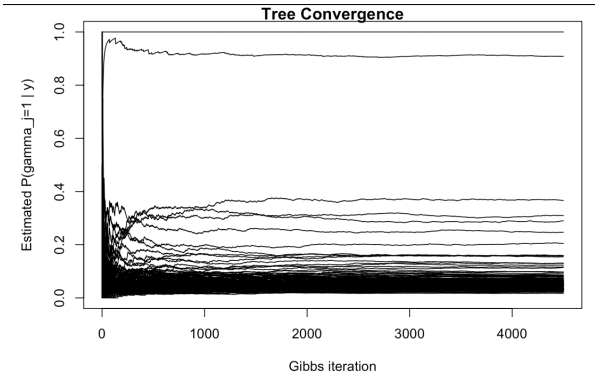


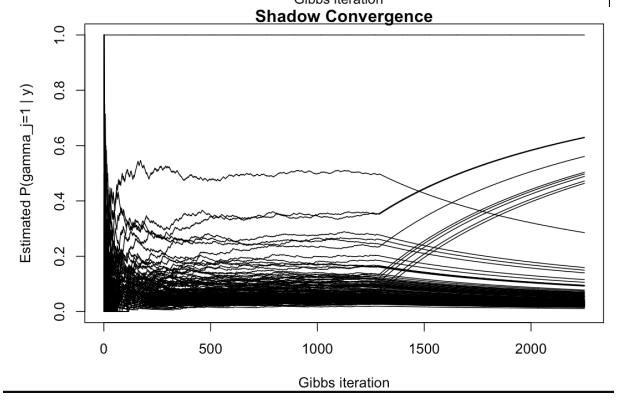


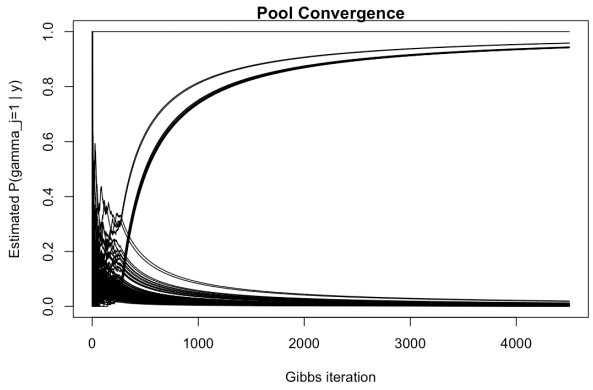











Pool Convergence
Estimated P(gamma_j=1 | y)
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
Gibbs iteration

Code for Unsupervised Learning

Contents


```
knitr::opts_chunk$set(message= FALSE, warning = FALSE)
```

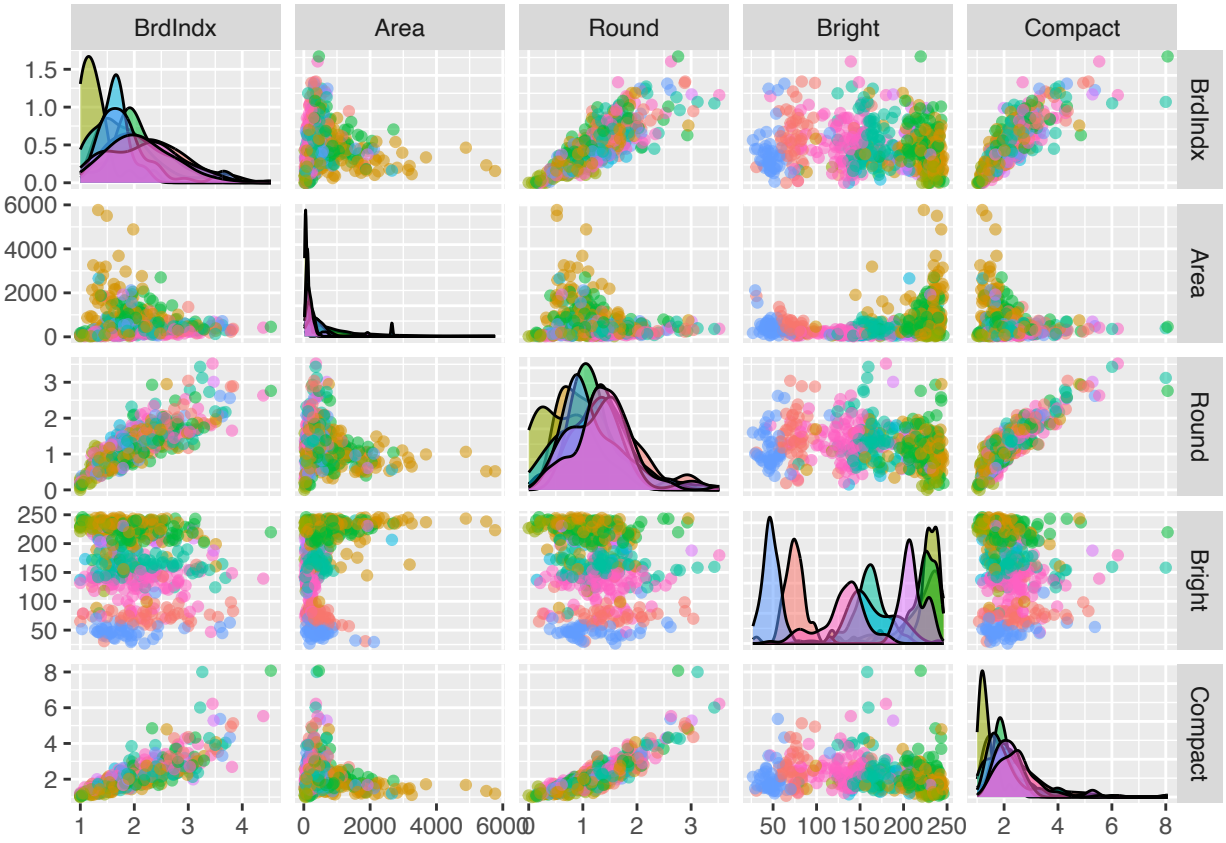
1 Load Data

```
# Read gene data
df_train <- read_csv("data/testing.csv")
df_test <- read_csv("data/training.csv")
```

```
df <- rbind(df_test, df_train)
X <- df[,-1]
y <- df[,1]
X <- scale(X)
```

2 Data Inspection and Visualization

```
library(GGally)

# Assuming 'train' is your dataframe in R and it has the same structure as in Python
# ggpair is used for creating pair plots in R from the GGally package
df_train$class <- as.factor(df_train$class)

ggpairs(df_train,
        columns = c('BrdIndx', 'Area', 'Round', 'Bright', 'Compact'),
        upper = list(continuous = "points"),
        ggplot2::aes(colour = class, alpha = 0.5)) + theme(legend.position = "bottom")
```

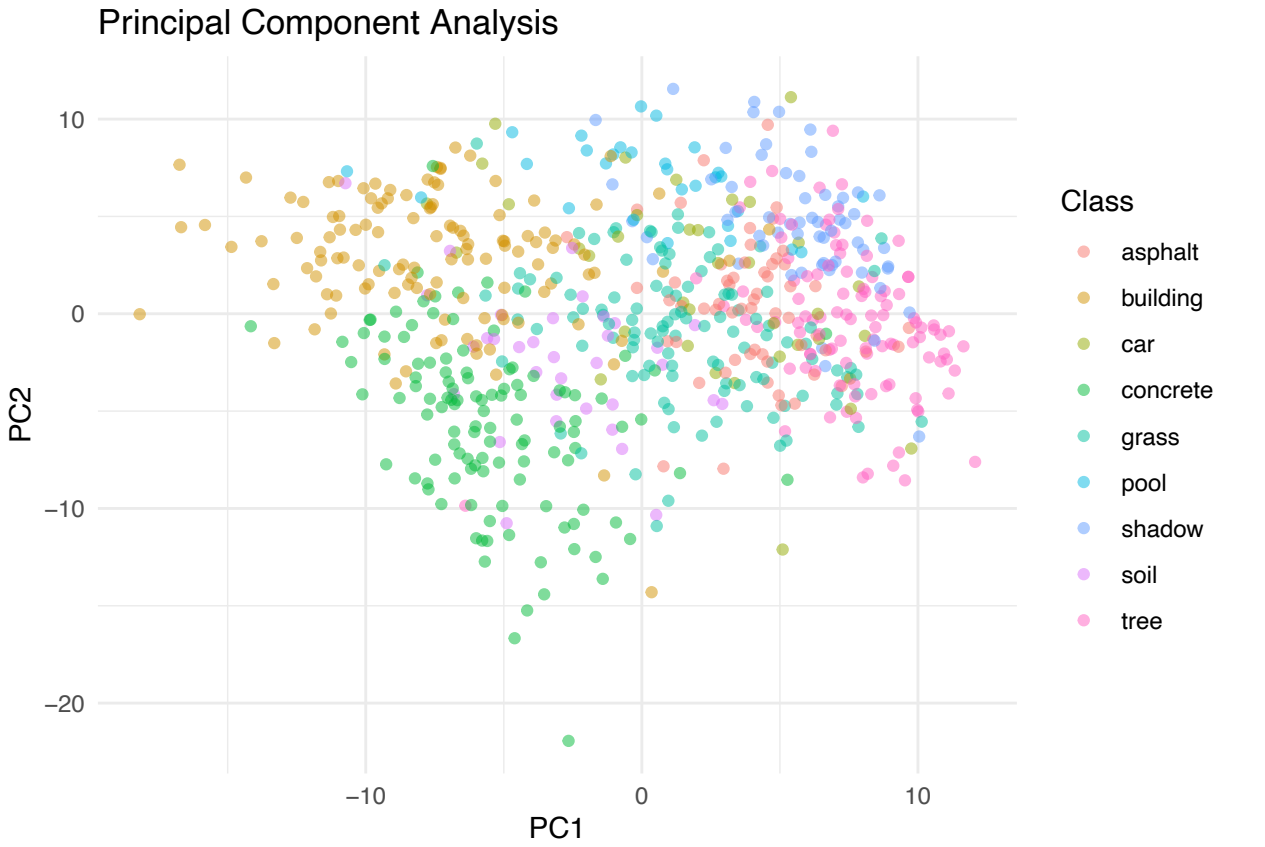


3 Principal Component Analysis

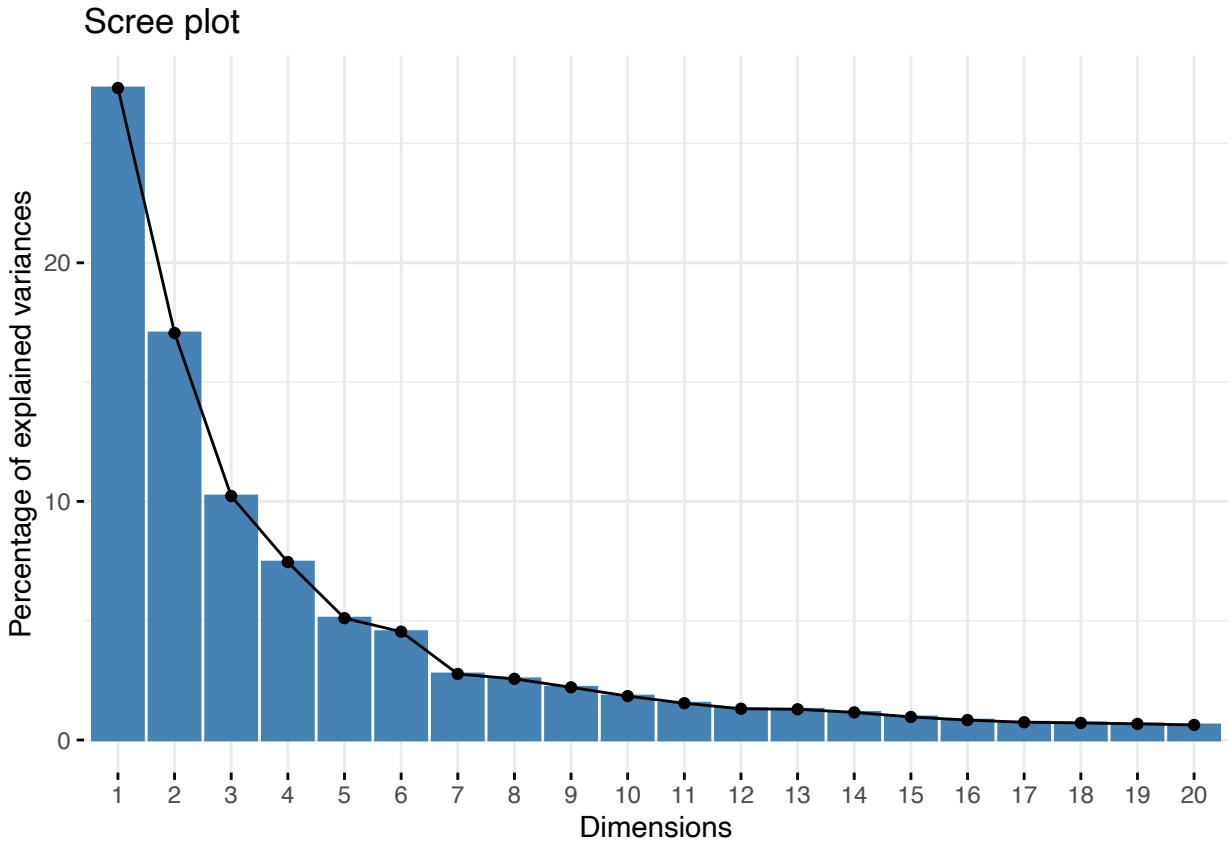
3.1 PCA

```
# perform PCA
pca <- prcomp(X)
X_pca = pca$x
```

```r
# plot the 2D rep using first 2 components
df_pca <- as_tibble(list("PC1" = pca$x[, 1],
                         "PC2" = pca$x[, 2],
                         "Class" = df$class))

p_pca <- ggplot(df_pca, aes(x = PC1, y = PC2, colour = Class)) +
  ggtitle("Principal Component Analysis") +
  geom_point(alpha = 0.5) +
  theme_minimal()
p_pca
```




```r
fviz_eig(pca,ncp=20)
```



The Elbow seems to be between 6 and 7 Principal Components, so I will preserve only the first 6 principal components.

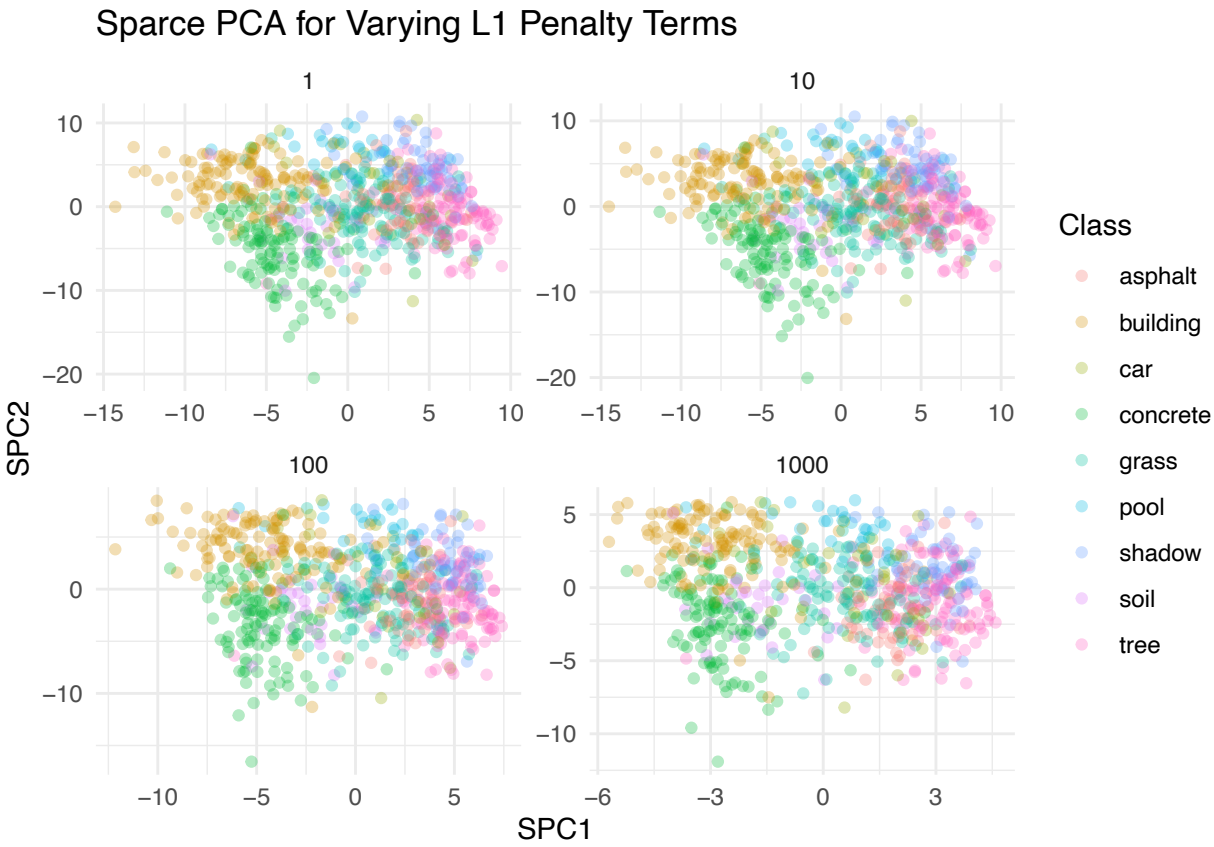
```
X_pca <- X_pca[,1:9]
```

3.2 Sparse PCA

3.2.1 Multiple Penalties

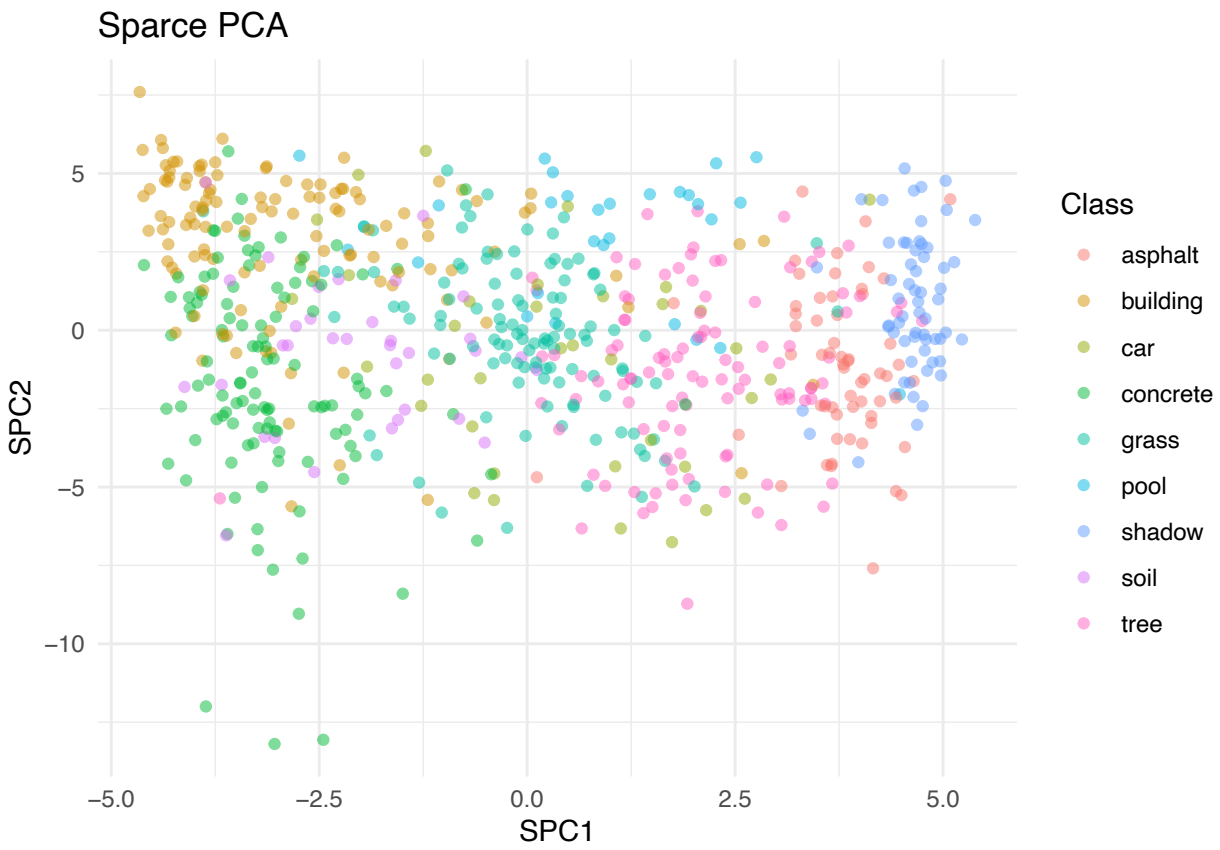
```
# perform sPCA on toy_df for a range of L1 penalty terms
penalties <- exp(seq(log(1), log(1000), length.out = 4))
df_ls <- lapply(penalties, function(penalty) {
  spca_sim_p <- elasticnet::spca(X, K = 2, para = rep(penalty, 2),
                    type = "predictor", sparse = "penalty")$loadings
  spca_sim_p <- as.matrix(X) %*% as.matrix(spca_sim_p)
  spca_out <- list("SPC1" = spca_sim_p[, 1],
                   "SPC2" = spca_sim_p[, 2],
                   "penalty" = round(rep(penalty, nrow(X))),
                   "Class"  = df$class) %>%
    as_tibble()
  return(spca_out)
})
df_spca_penaly <- dplyr::bind_rows(df_ls)

# plot the results of sPCA
p_spca_penalty <- ggplot(df_spca_penaly, aes(x = SPC1, y = SPC2, colour = Class)) +
  geom_point(alpha = 0.3) +
```

```
  ggtitle("Sparce PCA for Varying L1 Penalty Terms") +
  facet_wrap(~ penalty, nrow = 2, scales = "free") +
  theme_minimal()
p_spca_penalty
```



3.2.2 One Penalty

```
# perform sparse PCA, apparently 9 principal components is the maximum
sparse_pca <- elasticnet::spca(X, K = 9, type = "predictor", para = rep(100, 9), sparse="penalty")

X_spca <- as.matrix(X) %*% as.matrix(sparse_pca$loadings)
df_spca <- list("SPC1" = X_spca[, 1],
                "SPC2" = X_spca[, 2],
            #    "penalty" = round(rep(penalty, nrow(toy_df))),
                "Class"  = df$class) %>% as_tibble()

p_spca <- ggplot(df_spca, aes(x = SPC1, y = SPC2, colour = Class)) +
  ggtitle("Sparce PCA") +
  geom_point(alpha = 0.5) +
  theme_minimal()
p_spca
```



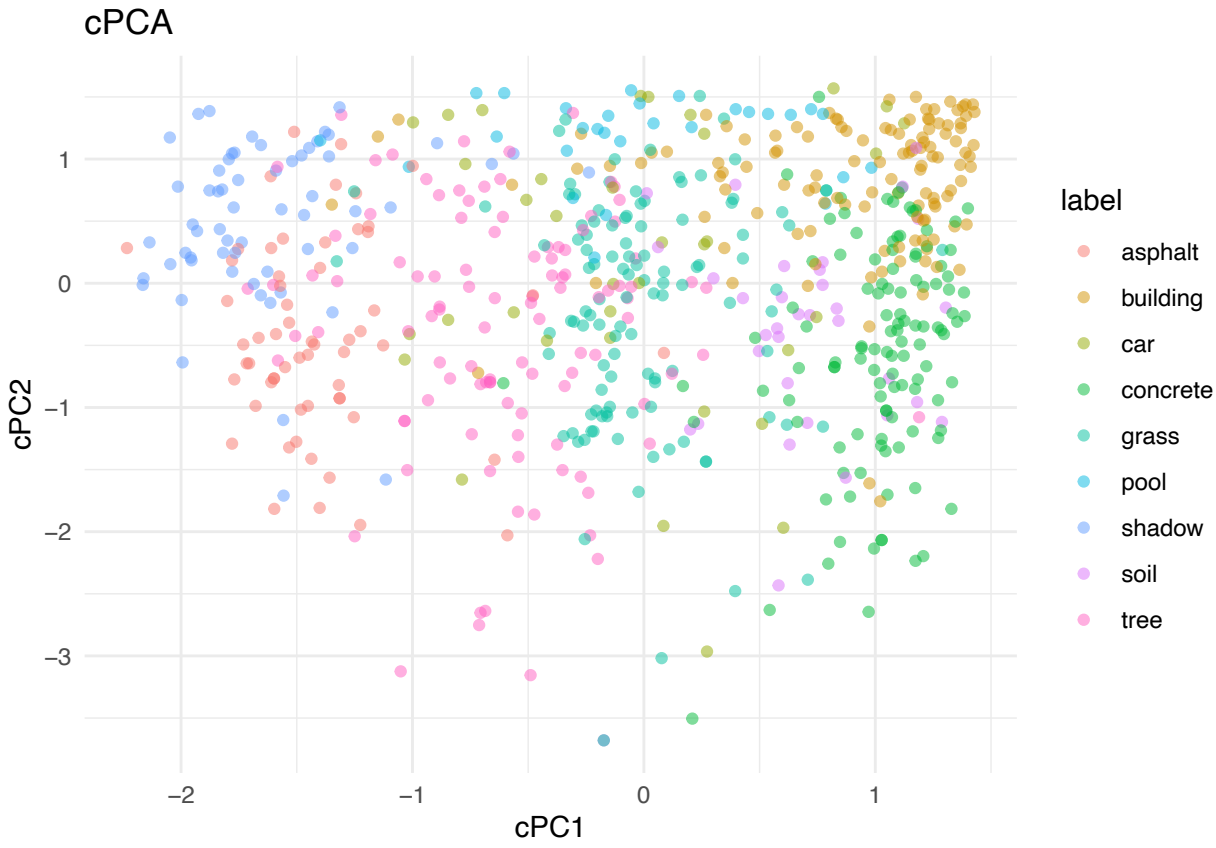
3.2.3 Contrastive PCA

Create Simulated Matrix

```
rows <- nrow(X)

# Function to generate a simulated column
simulate_column <- function(column) {
    column_mean <- mean(column, na.rm = TRUE)
    column_sd <- sd(column, na.rm = TRUE)
    rnorm(rows, mean = column_mean, sd = column_sd)
}

# Apply the function to each column of the original matrix and store the result
simulated_X <- apply(X, 2, simulate_column)
```

```
library(scPCA)
cpca_sim <- scPCA(target = X,
                  background = simulated_X,
                  penalties = 0.5,
                  n_centers = 9)

# create a dataframe to be plotted
cpca_df <- cpca_sim$x %>%
  as_tibble() %>%
  mutate(label = df$class)
colnames(cpca_df) <- c("cPC1", "cPC2", "label")
```

```
# plot the results
p_cpca <- ggplot(cpca_df, aes(x = cPC1, y = cPC2, colour = label)) +
  geom_point(alpha = 0.5) +
  ggtitle("cPCA") +
  theme_minimal()
p_cpca
```



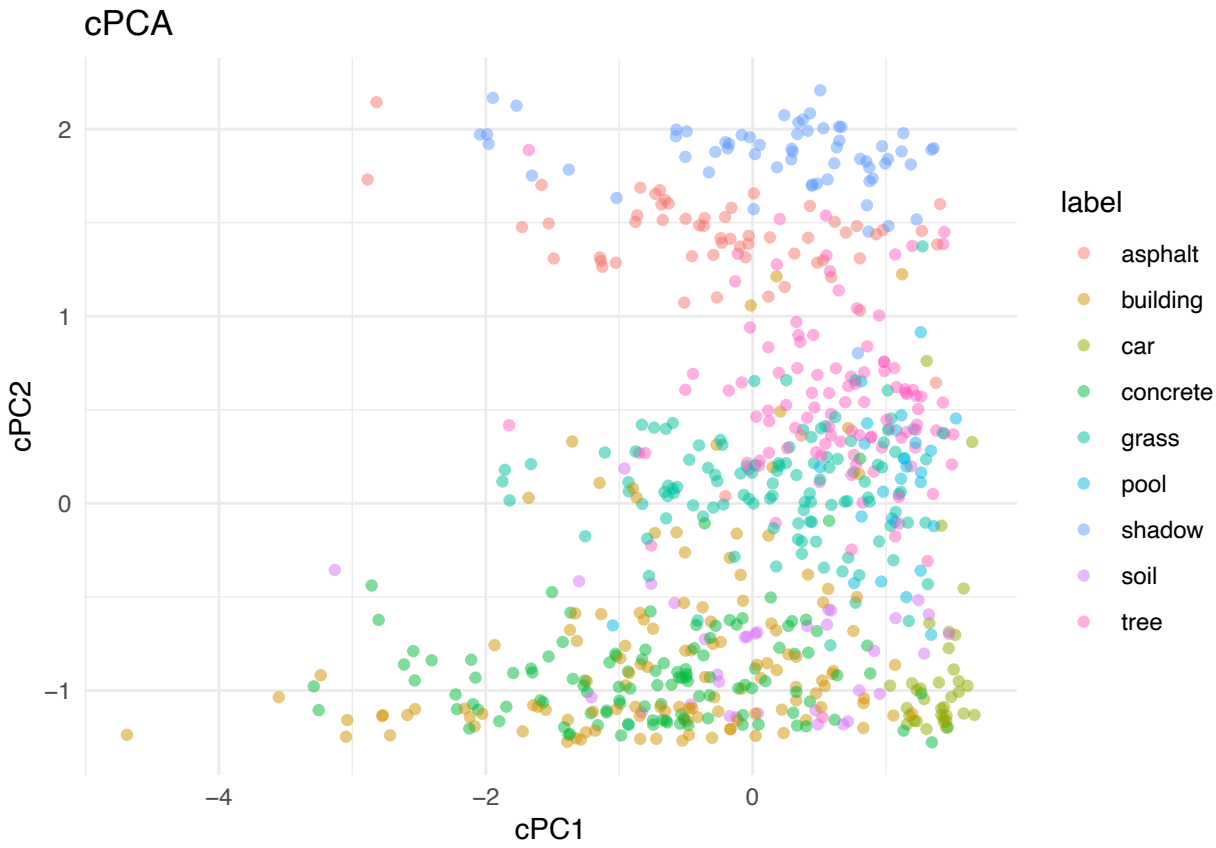
3.2.4 Contrastive PCA on 20 - 60 with Background 100 to 140

Create Simulated Matrix

```
X_20_60 = df %>%
  select(!ends_with(vars = colnames(X), c("80", "100", "120", "140"))) %>%
  select(- c("class", "GLCM3_140")) %>%
  as.matrix() %>%
  unname() %>%
  scale()

X_100_140 = df %>%
  select(ends_with(vars = colnames(X), c("100", "120", "140"))) %>%
  select(-GLCM3_80) %>%
  as.matrix() %>%
  unname() %>%
  scale()
```

```
library(scPCA)
cpca_sim <- scPCA(target = X_20_60,
                  background = X_100_140,
```

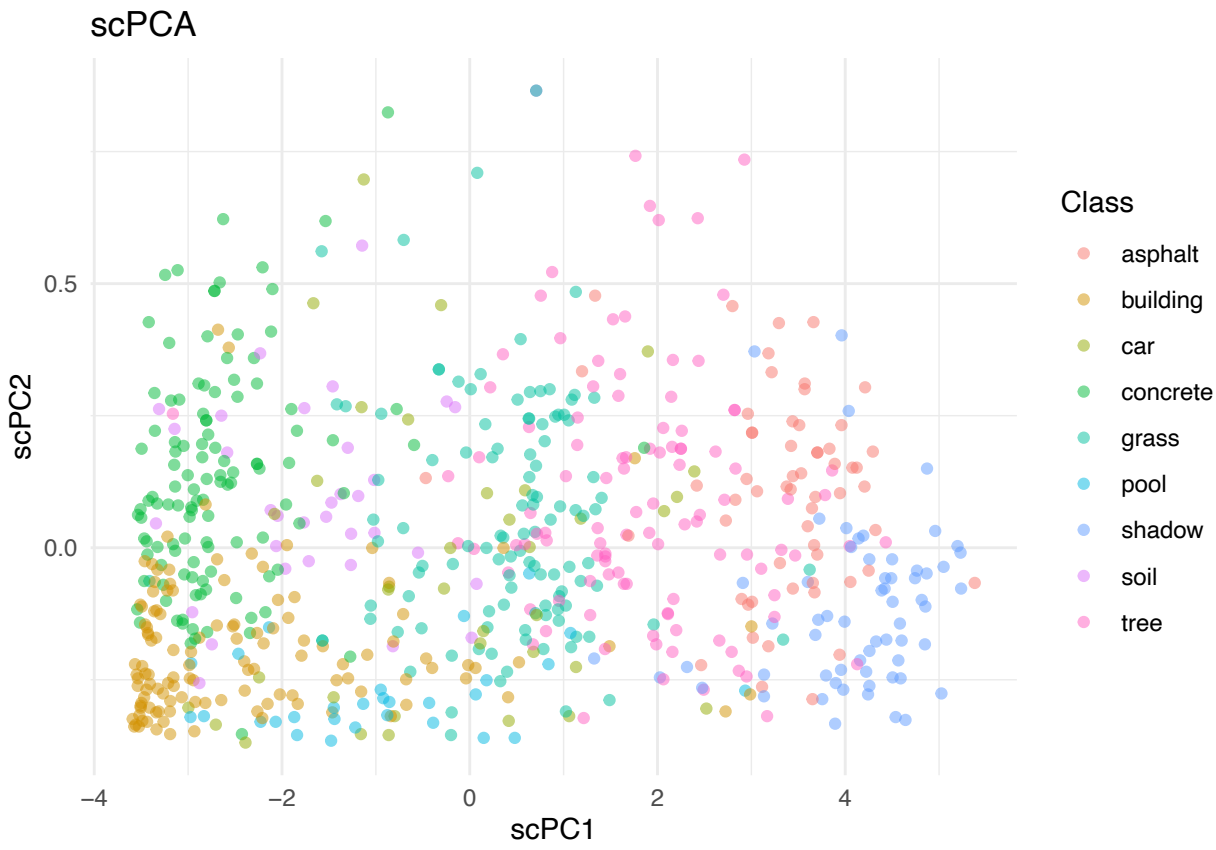
```
                  penalties = 1.0,
                  n_centers = 9)

# create a dataframe to be plotted
cpca_df <- cpca_sim$x %>%
  as_tibble() %>%
  mutate(label = df$class)
colnames(cpca_df) <- c("cPC1", "cPC2", "label")

# plot the results
p_cpca <- ggplot(cpca_df, aes(x = cPC1, y = cPC2, colour = label)) +
  geom_point(alpha = 0.5) +
  ggtitle("cPCA") +
  theme_minimal()
p_cpca
```



3.2.5 Sparse Contrastive PCA (scPCA)

```
# run scPCA for using 40 logarithmically seperated contrastive parameter values
# and possible 20 L1 penalty terms
scpca_sim <- scPCA(target = X,
                   background = simulated_X,
                   n_centers = 9,
                   penalties = exp(seq(log(0.01), log(0.5), length.out = 10)),
                   alg = "var_proj")
```

```r
# create a dataframe to be plotted
scpca_df <- scpca_sim$x %>%
  as_tibble() %>%
  mutate(Class = df$class)
colnames(scpca_df) <- c("scPC1", "scPC2", "Class")

# plot the results
p_scpca <- ggplot(scpca_df, aes(x = scPC1, y = scPC2, colour = Class)) +
  geom_point(alpha = 0.5) +
  ggtitle("scPCA") +
  theme_minimal()
p_scpca
```



3.2.6 Hyperparameter Tuning

```r
# cpca_cv_sim <- scPCA(target = X,
#                      background = simulated_X,
#                      penalties = 0,
#                      n_centers = 9,
#                      cv = 3)
#
# # create a dataframe to be plotted
# cpca_cv_df <- cpca_cv_sim$x %>%
#   as_tibble() %>%
#   dplyr::mutate(label = df$class)
```

```
# colnames(cpca_cv_df) <- c("cPC1", "cPC2", "label")
#
# # plot the results
# p_cpca_cv <- ggplot(cpca_cv_df, aes(x = cPC1, y = cPC2, colour = label)) +
#   geom_point(alpha = 0.5) +
#   ggtitle("cPCA of Simulated Data") +
#   theme_minimal()
# p_cpca_cv
```

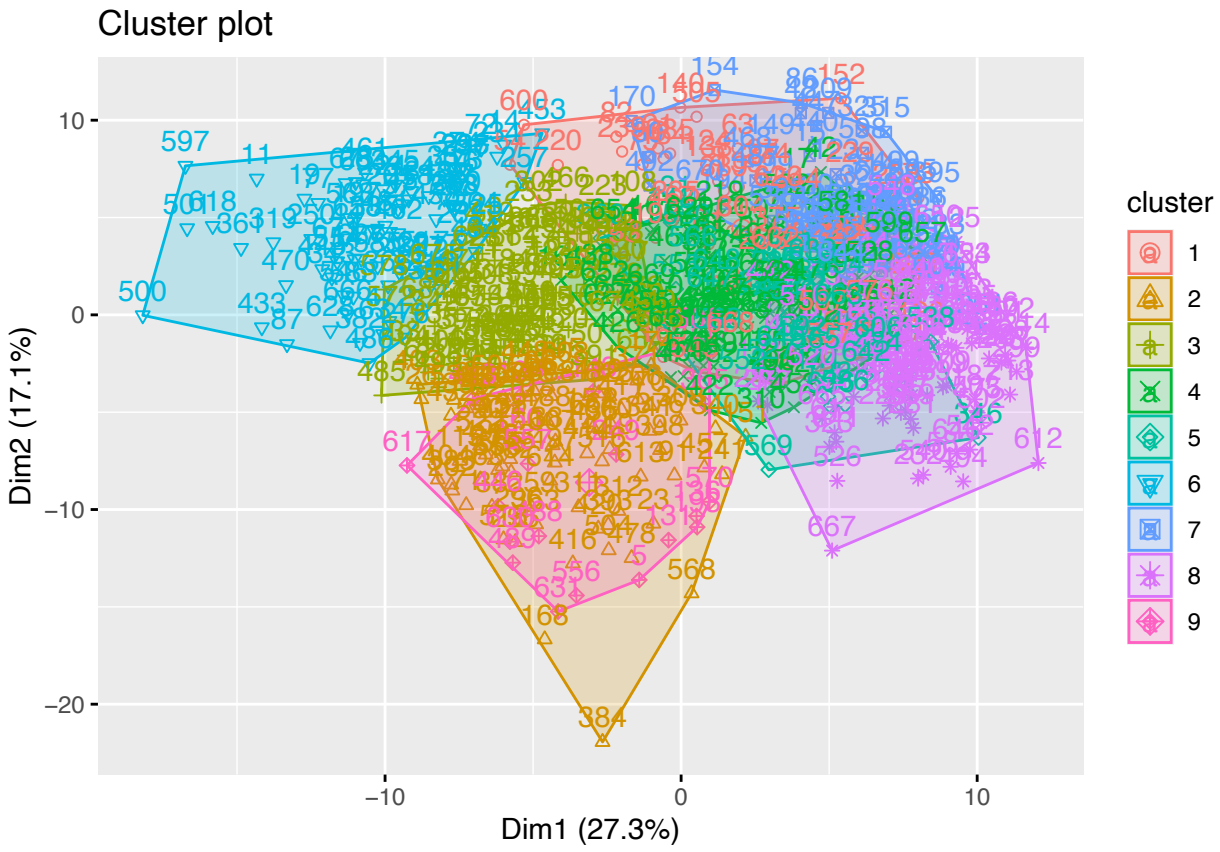
```
# scpca_cv_sim <- scPCA(target = X,
#                       background = simulated_X,
#                       n_centers = 9,
#                       n_eigen = 2,
#                       cv = 10,
#                       penalties = exp(seq(log(0.01), log(0.5), length.out = 10)),
#                       alg = "var_proj")
# X_scpca <- scpca_cv_sim$x
#
# # create a dataframe to be plotted
# scpca_cv_df <- scpca_cv_sim$x[,1:2] %>%
#   as_tibble() %>%
#   mutate(label = df$class)
# colnames(scpca_cv_df) <- c("scPC1", "scPC2", "label")
#
# # plot the results
# p_scpca_cv <- ggplot(scpca_cv_df, aes(x = -scPC1, y = -scPC2, colour = label)) +
#   geom_point(alpha = 0.5) +
#   ggtitle("scPCA of Simulated Data") +
#   theme_minimal()
# p_scpca_cv
```

4 Clustering the High Dimensional Data

4.1 K-Means

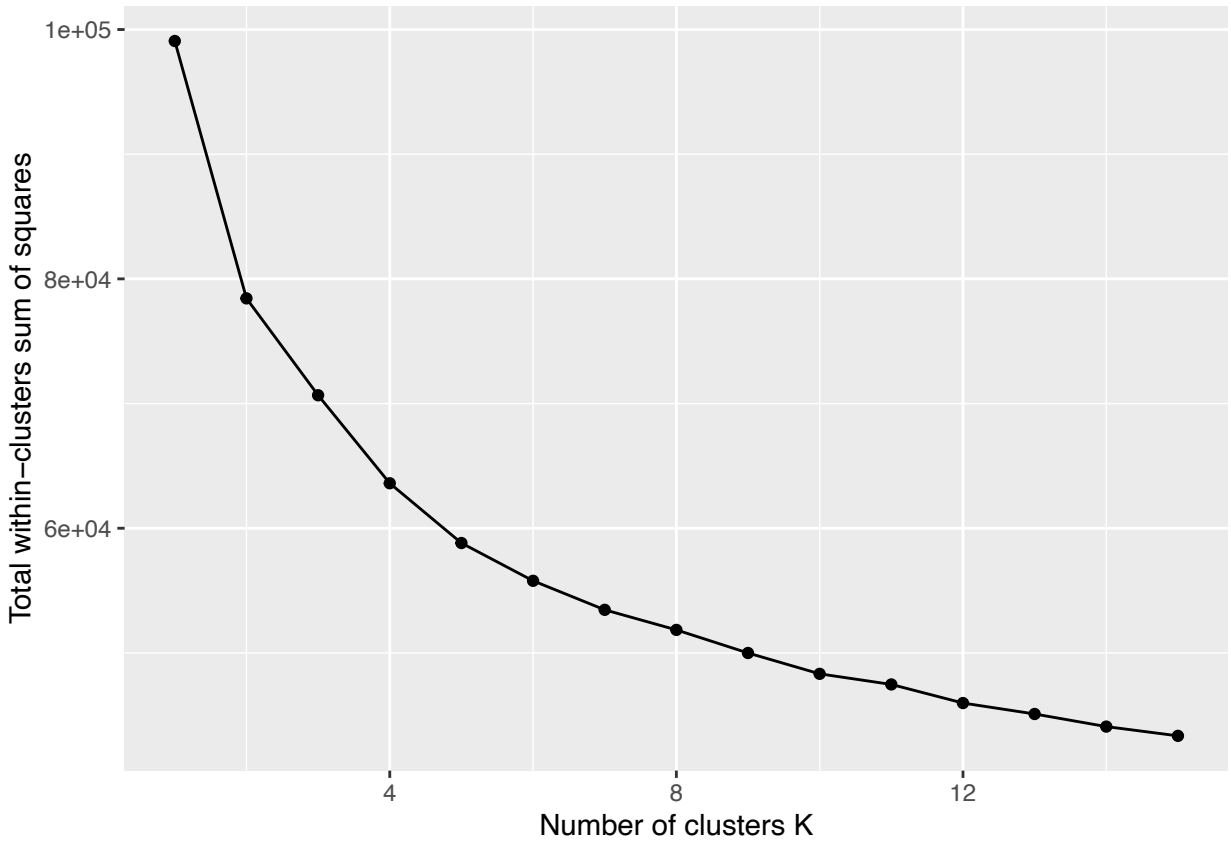
4.1.1 Plot

```
k9 <- kmeans(X, centers = 9, nstart = 10, iter.max = 20)
kmeans.groups <- k9$cluster
fviz_cluster(k9, data = X)
```



4.1.2 K-Means Elbow Plot

```
set.seed(947386)
wss <- function(k) {
  kmeans(X, k, nstart = 10)$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

wss_df <- data.frame(k=k.values, wss= wss_values)
ggplot(wss_df, aes(x=k,y=wss)) +
  geom_point() +
  geom_line() +
  xlab("Number of clusters K") +
  ylab("Total within-clusters sum of squares")
```



No apparent optimal number of clusters

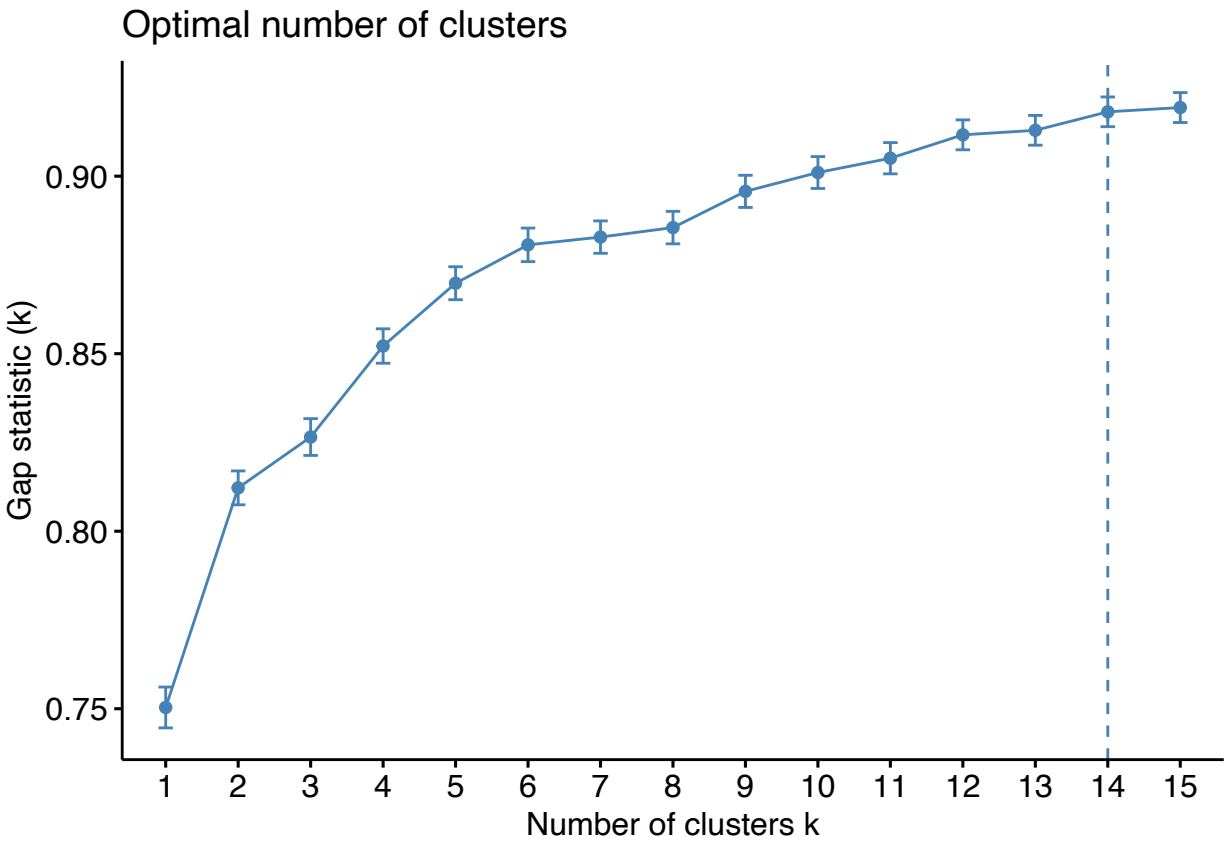
4.1.3 Gap Statistic

```
# compute gap statistic
gap_stat <- clusGap(X, FUN = kmeans, nstart = 20, iter.max = 50,
                    K.max = 15, B = 20)


# Print the result using Tibs2001SEmax
print(gap_stat, method = "Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = X, FUNcluster = kmeans, K.max = 15, B = 20, nstart = 20, iter.max = 50)
## B=20 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 6
##           logW   E.logW       gap      SE.sim
##  [1,] 7.930675 8.681027 0.7503523 0.005750640
##  [2,] 7.808286 8.620488 0.8122017 0.004766716
##  [3,] 7.756505 8.583024 0.8265196 0.005179671
##  [4,] 7.701201 8.553361 0.8521599 0.004848769
##  [5,] 7.662618 8.532464 0.8698462 0.004637051
##  [6,] 7.633844 8.514500 0.8806562 0.004727553
##  [7,] 7.618167 8.501012 0.8828446 0.004562964
##  [8,] 7.603111 8.488629 0.8855182 0.004575426
##  [9,] 7.582754 8.478476 0.8957217 0.004529458
## [10,] 7.568007 8.469031 0.9010242 0.004491297
```

```
## [11,] 7.555170 8.460228 0.9050583 0.004400803
## [12,] 7.540749 8.452392 0.9116430 0.004214375
## [13,] 7.531909 8.444825 0.9129160 0.004187604
## [14,] 7.519940 8.438065 0.9181249 0.004193786
## [15,] 7.511713 8.431041 0.9193273 0.004214598
```

```r
# plot the gap
fviz_gap_stat(gap_stat)
```

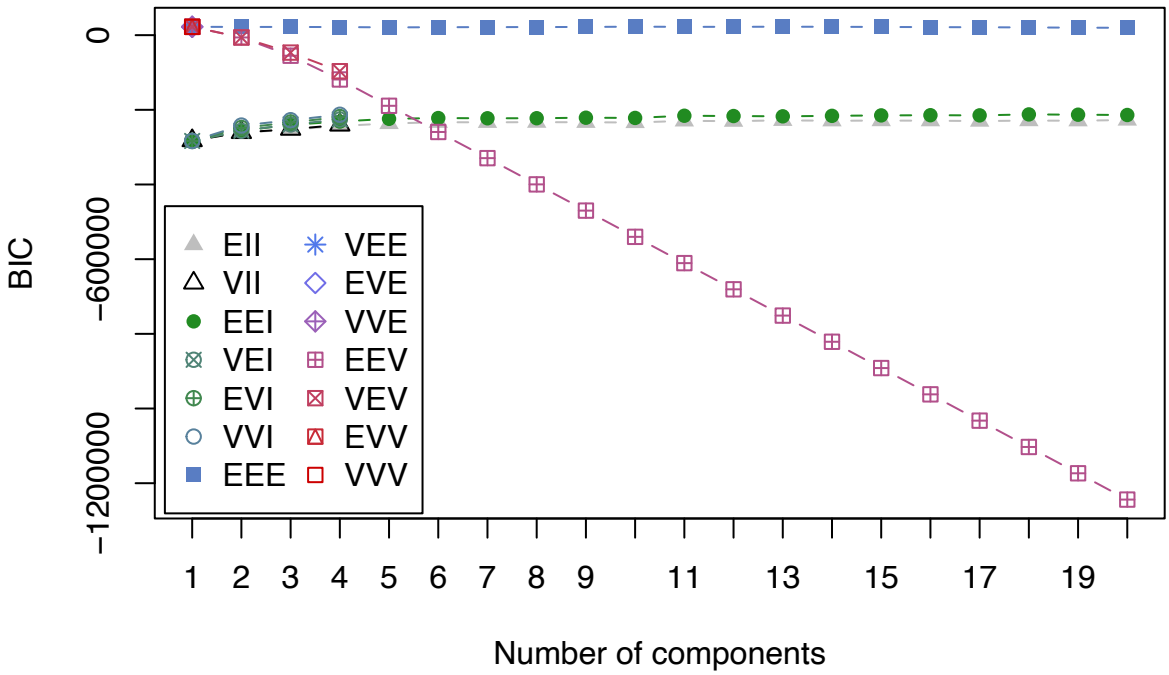


4.1.4 Gaussian Mixture Models

```r
mod <- Mclust(X, G = 1:20)
mod10 <- Mclust(X, G = 10)
gmm.groups <- mod10$classification
summary(mod$BIC)
```

```
## Best BIC values:
##             EEE,10     EEE,13     EEE,14
## BIC      22654.39 22582.4129 22360.0453
## BIC diff     0.00   -71.9748  -294.3424
```

```r
plot(mod, what = "BIC", ylim = range(mod$BIC[,-(1:2)], na.rm = TRUE),
     legendArgs = list(x = "bottomleft"))
```



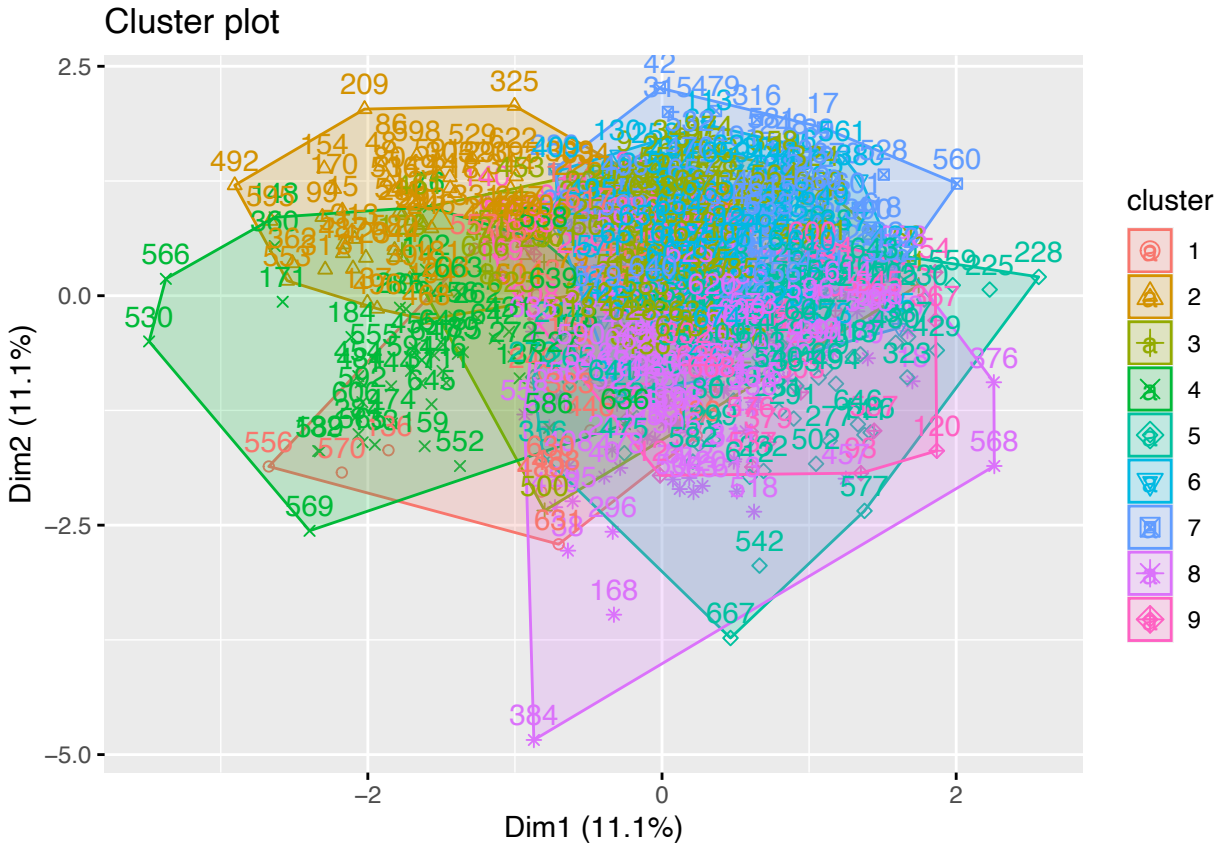
This plot shows no clear distinction in the BIC value.

5 Clustering for PCA

5.1 K-Means

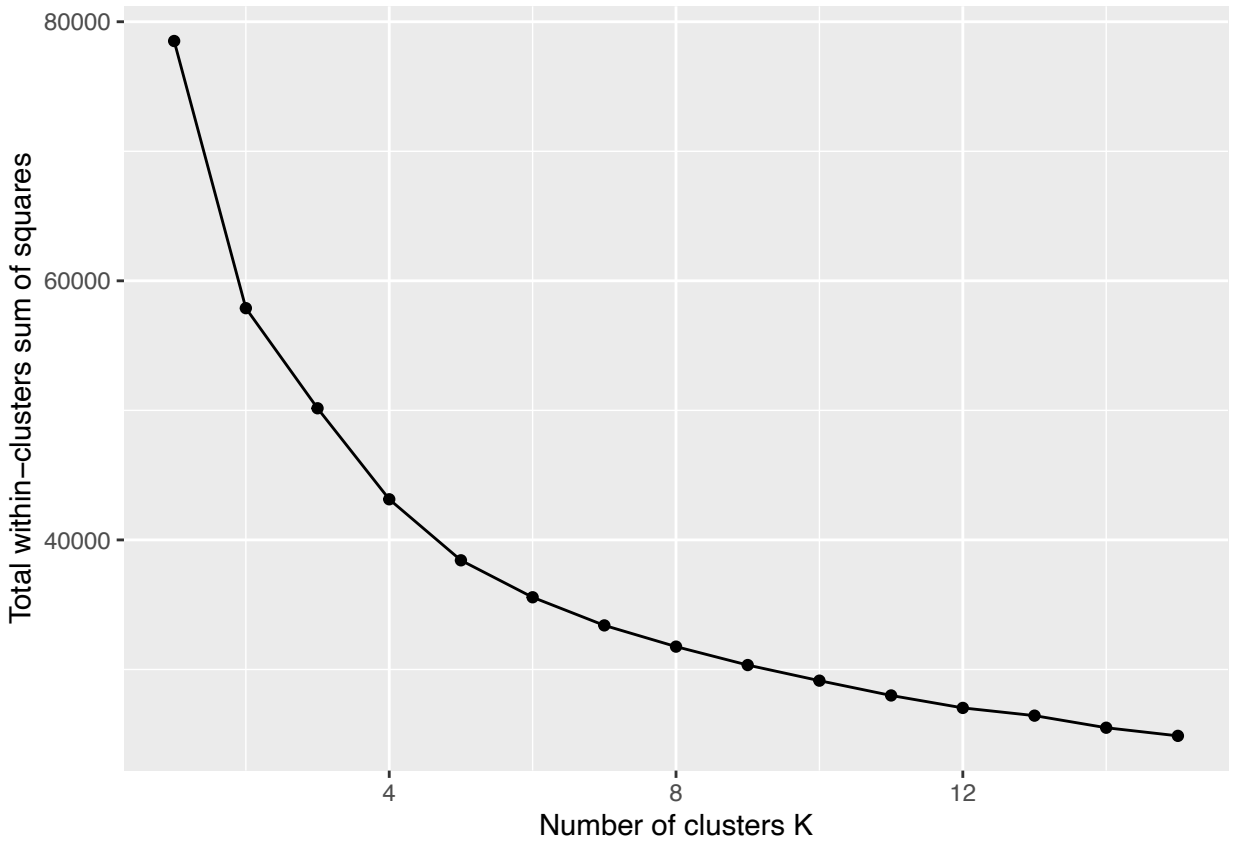
5.1.1 Plot

```
k9_pca <- kmeans(X_pca, centers = 9, nstart = 10, iter.max = 20)
kmeans_pca.groups <- k9_pca$cluster
fviz_cluster(k9_pca, data = X_pca)
```



5.1.2 K-Means Elbow Plot

```
set.seed(947386)
wss <- function(k) {
  kmeans(X_pca, k, nstart = 10)$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

wss_df <- data.frame(k=k.values, wss= wss_values)
ggplot(wss_df, aes(x=k,y=wss)) +
  geom_point() +
  geom_line() +
  xlab("Number of clusters K") +
  ylab("Total within-clusters sum of squares")
```



No apparent optimal number of clusters

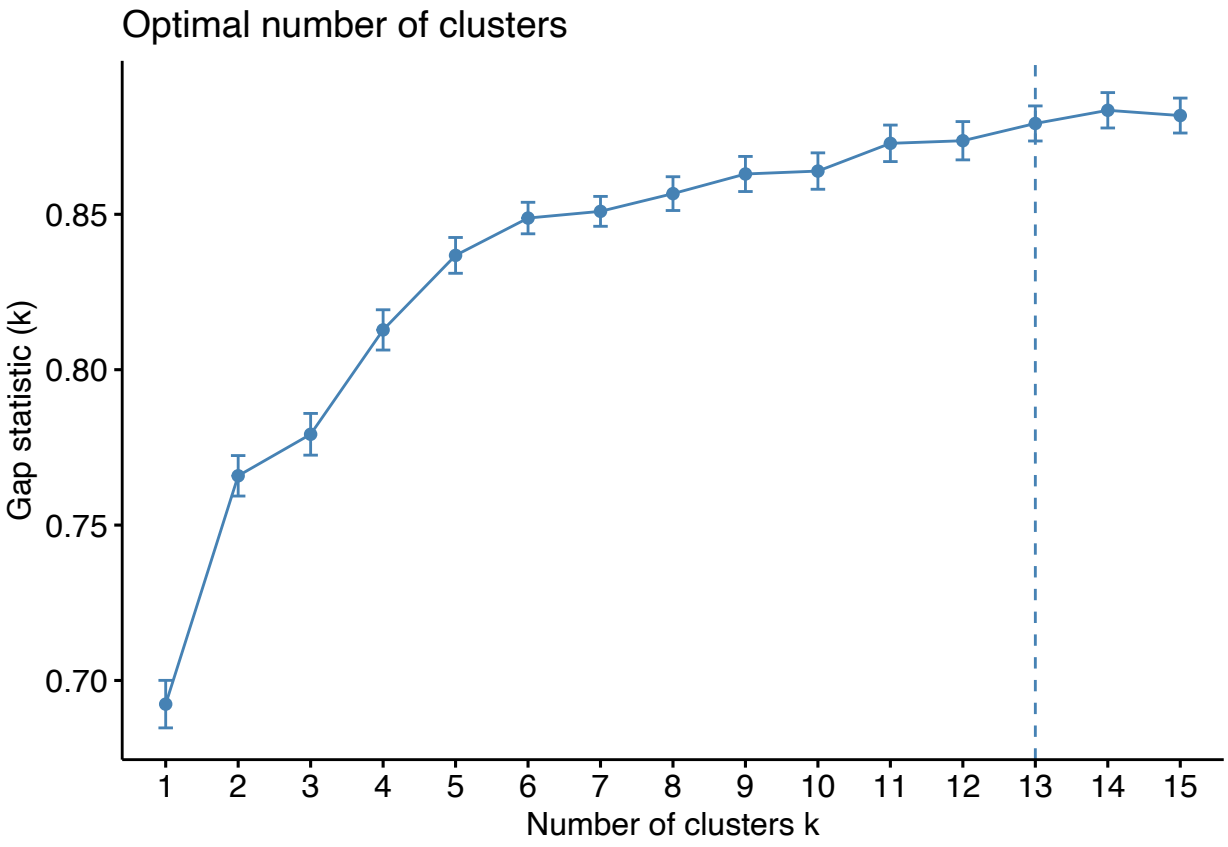
5.1.3 Gap Statistic PCA

```
# compute gap statistic
gap_stat_pca <- clusGap(X_pca, FUN = kmeans, nstart = 20, iter.max = 50,
                    K.max = 15, B = 20)


# Print the result using Tibs2001SEmax
print(gap_stat_pca, method = "Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = X_pca, FUNcluster = kmeans, K.max = 15, B = 20, nstart = 20, iter.max = 50)
## B=20 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 6
##           logW   E.logW       gap      SE.sim
##  [1,] 7.800868 8.493255 0.6923871 0.007646102
##  [2,] 7.638705 8.404560 0.7658546 0.006513820
##  [3,] 7.566978 8.346194 0.7792159 0.006714848
##  [4,] 7.487681 8.300485 0.8128032 0.006477785
##  [5,] 7.429946 8.266728 0.8367821 0.005762157
##  [6,] 7.387766 8.236561 0.8487950 0.005063482
##  [7,] 7.362757 8.213708 0.8509513 0.004809847
##  [8,] 7.336384 8.193037 0.8566528 0.005434127
##  [9,] 7.312897 8.175871 0.8629745 0.005647048
## [10,] 7.295688 8.159600 0.8639118 0.005855624
```

```
## [11,] 7.272404 8.145241 0.8728367 0.005871940
## [12,] 7.257973 8.131651 0.8736787 0.006161346
## [13,] 7.240391 8.119622 0.8792307 0.005639057
## [14,] 7.224698 8.108168 0.8834704 0.005694583
## [15,] 7.215260 8.097033 0.8817731 0.005619892
```

```
# plot the gap
fviz_gap_stat(gap_stat_pca)
```

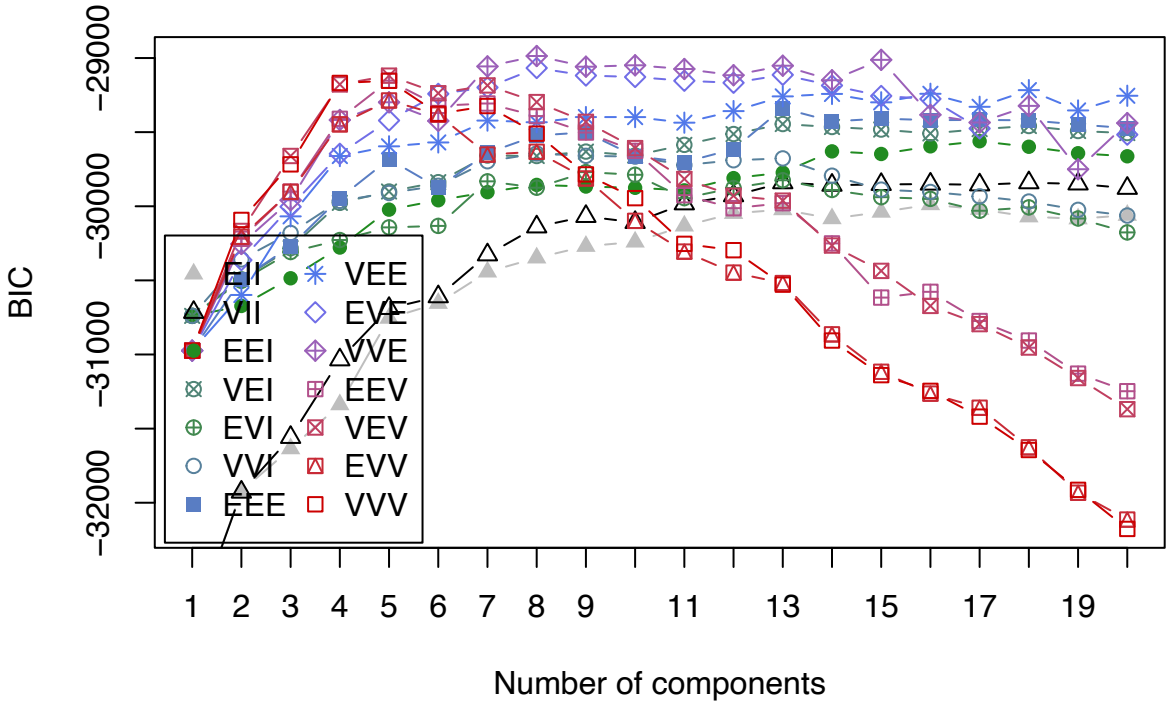


5.1.4 Gaussian Mixture Models with PCA

```
mod_pca <- Mclust(X_pca, G = 1:20)
mod_pca8 <- Mclust(X_pca, G = 8)
gmm.pca.groups <- mod_pca8$classification
summary(mod_pca$BIC)
```

```
## Best BIC values:
##              VVE,8       VVE,15       VVE,10
## BIC      -28986.36 -29012.23145 -29048.96417
## BIC diff      0.00    -25.86974    -62.60245
```

```
plot(mod_pca, what = "BIC", ylim = range(mod_pca$BIC[,-(1:2)], na.rm = TRUE),
     legendArgs = list(x = "bottomleft"))
```

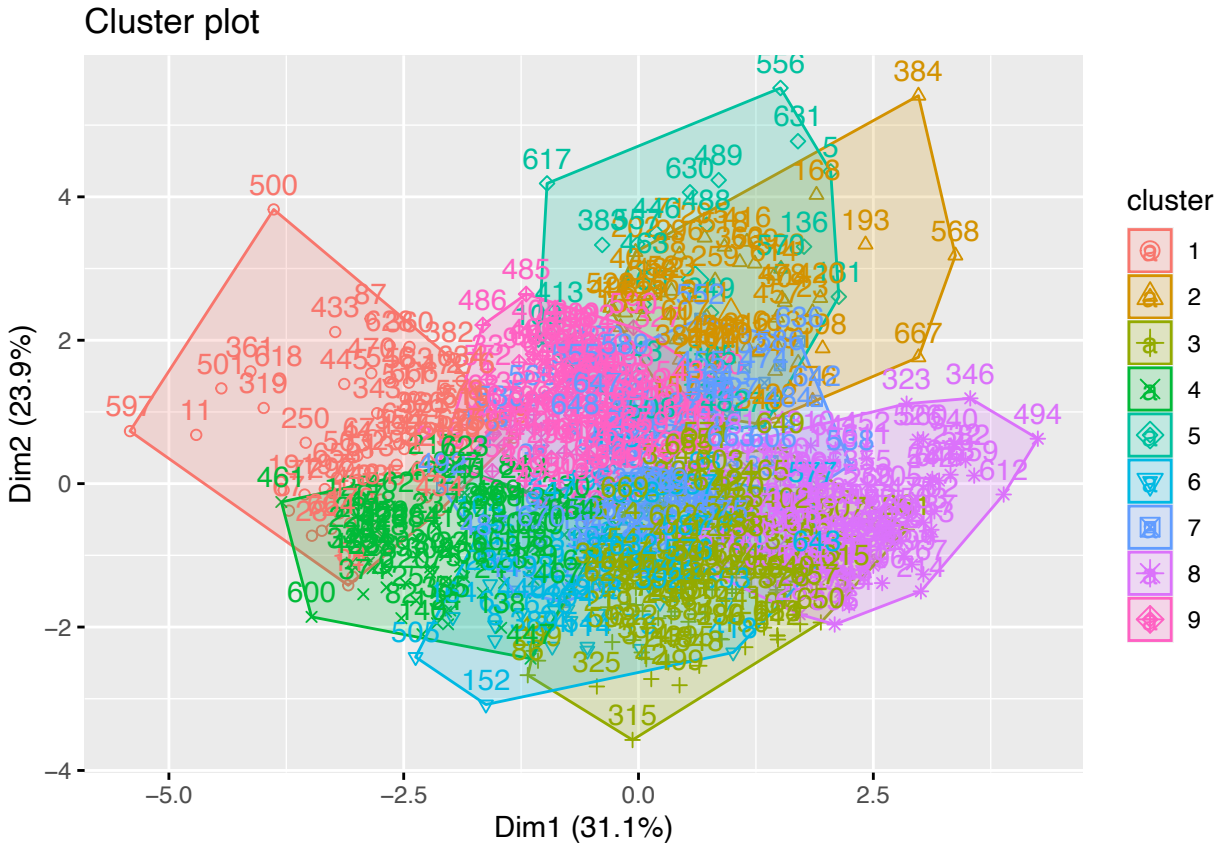


6 Clustering for Sparce PCA

6.1 K-Means

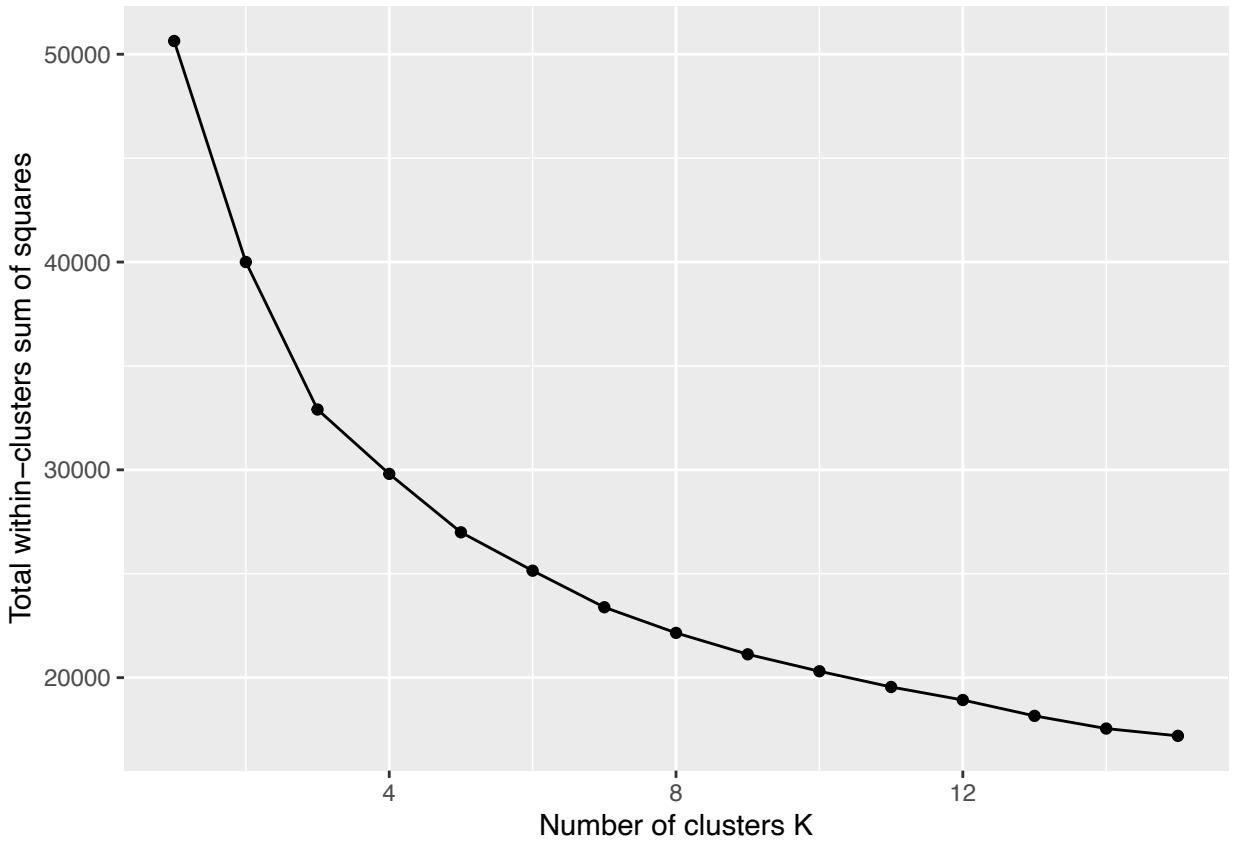
6.1.1 Plot

```
k9_spca <- kmeans(X_spca, centers = 9, nstart = 10, iter.max = 20)
kmeans_spca.groups <- k9_spca$cluster
fviz_cluster(k9_spca, data = X_spca)
```



6.1.2 K-Means Elbow Plot

```
set.seed(947386)
wss <- function(k) {
  kmeans(X_spca, k, nstart = 10)$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

wss_df <- data.frame(k=k.values, wss= wss_values)
ggplot(wss_df, aes(x=k,y=wss)) +
  geom_point() +
  geom_line() +
  xlab("Number of clusters K") +
  ylab("Total within-clusters sum of squares")
```



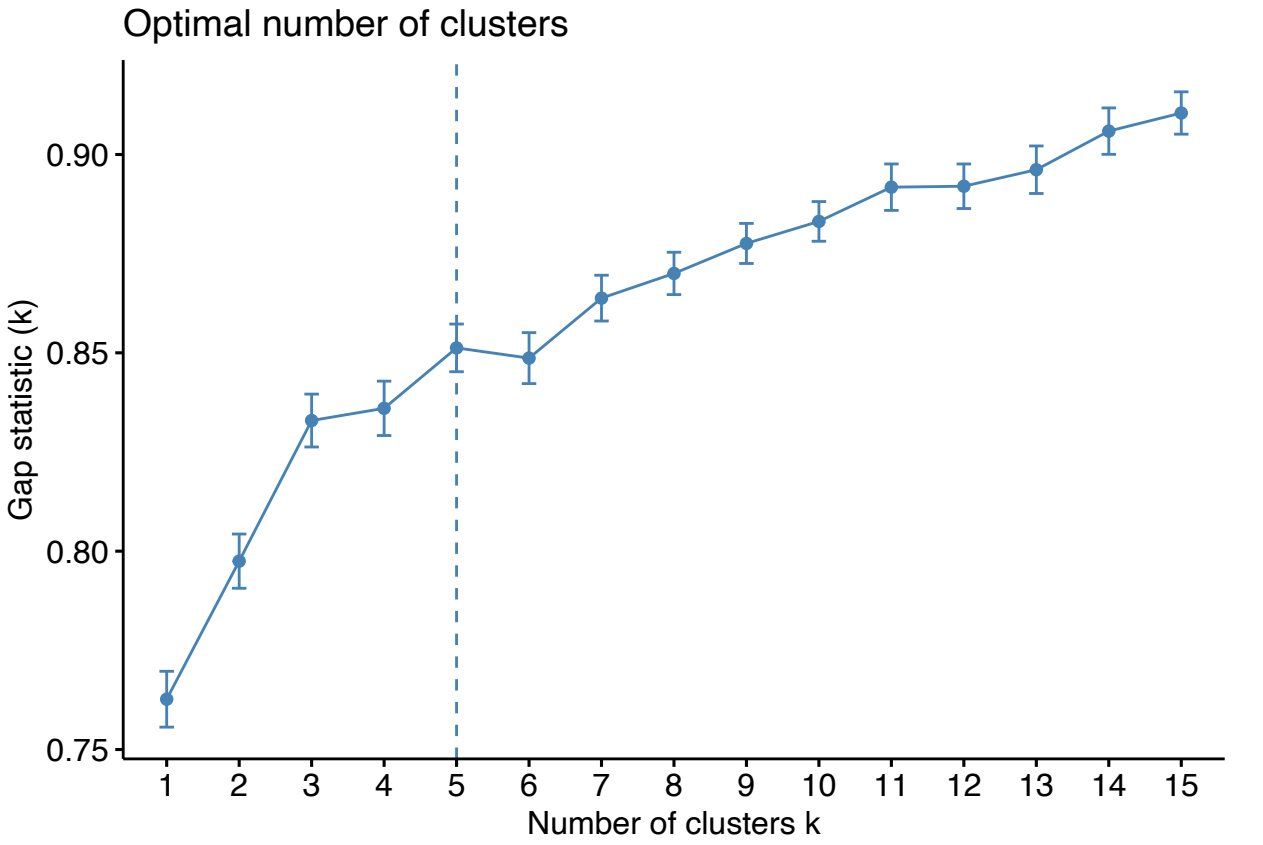
6.1.3 Gap Statistic sparce PCA

```
# compute gap statistic
gap_stat_spca <- clusGap(X_spca, FUN = kmeans, nstart = 20, iter.max = 50,
                    K.max = 15, B = 20)


# Print the result using Tibs2001SEmax
print(gap_stat_spca, method = "Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = X_spca, FUNcluster = kmeans, K.max = 15, B = 20, nstart = 20, iter.max = 50)
## B=20 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 3
##           logW   E.logW       gap      SE.sim
##  [1,] 7.576770 8.339453 0.7626836 0.007039821
##  [2,] 7.452590 8.250075 0.7974845 0.006838898
##  [3,] 7.354671 8.187604 0.8329331 0.006666225
##  [4,] 7.300386 8.136396 0.8360100 0.006856617
##  [5,] 7.248841 8.100088 0.8512473 0.006009016
##  [6,] 7.218620 8.067288 0.8486678 0.006423018
##  [7,] 7.179308 8.043099 0.8637916 0.005765663
##  [8,] 7.153082 8.023099 0.8700166 0.005336007
##  [9,] 7.129330 8.006908 0.8775786 0.005047722
## [10,] 7.109212 7.992345 0.8831330 0.004999762
## [11,] 7.086966 7.978728 0.8917628 0.005851108
## [12,] 7.073549 7.965546 0.8919970 0.005615151
```

```
## [13,] 7.057533 7.953694 0.8961612 0.005988149
## [14,] 7.036804 7.942697 0.9058931 0.005851582
## [15,] 7.021293 7.931755 0.9104622 0.005336129
```

```
# plot the gap
fviz_gap_stat(gap_stat_spca)
```

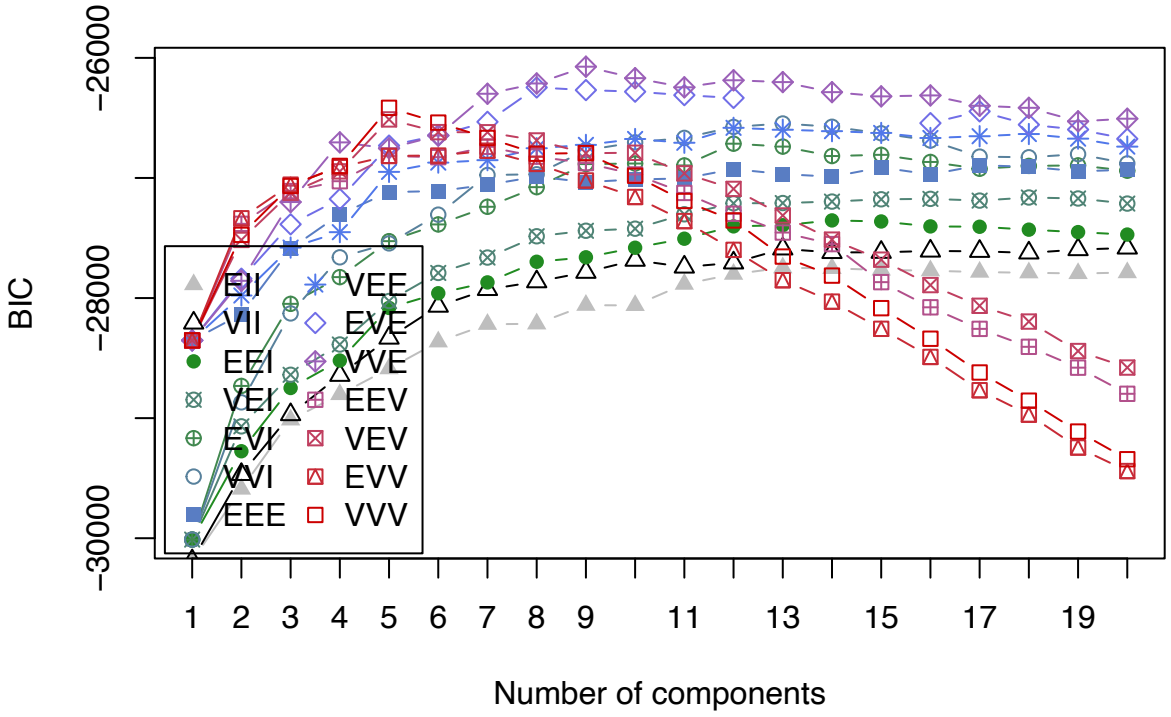


6.1.4 Gaussian Mixture Models with Sparse PCA

```
mod_spca <- Mclust(X_spca, G = 1:20)
mod_spca9 <- Mclust(X_spca, G = 9)
gmm.spca.groups <- mod_spca9$classification
summary(mod_spca$BIC)
```

```
## Best BIC values:
##              VVE,9       VVE,10      VVE,12
## BIC      -26073.38 -26168.74239 -26186.5405
## BIC diff      0.00    -95.35931   -113.1574
```

```
plot(mod_spca, what = "BIC", ylim = range(mod_spca$BIC[,-(1:2)], na.rm = TRUE),
     legendArgs = list(x = "bottomleft"))
```



7 Comparing Cluster Assigments to the truth

```
truth.labels <- as.matrix(df[,1])
truth <- as.numeric(as.factor(as.matrix(df[,1])))
table(truth.labels)
```

```
## truth.labels
##  asphalt building      car concrete    grass     pool   shadow     soil
##       59      122       36      116      112       29       61       34
##     tree
##      106
```

```
compare.kmeans.truth <- cluster.stats(
  clustering= kmeans.groups,
  alt.clustering = truth,
  compareonly = TRUE)

compare.kmeans.pca.truth <- cluster.stats(
  clustering= kmeans_pca.groups,
  alt.clustering = truth,
  compareonly = TRUE)

compare.kmeans.spca.truth <- cluster.stats(
  clustering= kmeans_spca.groups,
  alt.clustering = truth,
  compareonly = TRUE)

compare.gmm.truth <- cluster.stats(
  clustering= gmm.groups,
  alt.clustering = truth,
```

```
  compareonly = TRUE)

compare.gmm.pca.truth <- cluster.stats(
  clustering= gmm.pca.groups,
  alt.clustering = truth,
  compareonly = TRUE)

compare.gmm.spca.truth <- cluster.stats(
  clustering= gmm.spca.groups,
  alt.clustering = truth,
  compareonly = TRUE)

compare.res <- rbind(
  unlist(compare.kmeans.truth),
  unlist(compare.kmeans.pca.truth),
  unlist(compare.kmeans.spca.truth),
  unlist(compare.gmm.truth),
  unlist(compare.gmm.pca.truth),
  unlist(compare.gmm.spca.truth))
compare.res <- data.frame(cbind(c('K-means','K-means PCA','K-means SPCA', 'Gaussian Mixtures', 'Gaussia
colnames(compare.res)[1] <- 'clustering method'
compare.res %>% arrange(desc(corrected.rand))
```

```
##        clustering method corrected.rand    vi
## 1 Gaussian Mixtures SPCA           0.47 1.743
## 2  Gaussian Mixtures PCA          0.447 1.729
## 3            K-means PCA          0.422 1.875
## 4                K-means          0.395 1.971
## 5           K-means SPCA          0.284 2.305
## 6      Gaussian Mixtures          0.186  1.91
```

```
table(gmm.spca.groups,truth.labels)
```

```
##                truth.labels
## gmm.spca.groups asphalt building car concrete grass pool shadow soil tree
##               1       0        7  19        1     0   27      3    0    0
##               2       0       22   0       87     0    0      0    3    1
##               3       0       66   0       10     1    0      0    3    0
##               4       2        5   0        6     5    1      6    1    0
##               5       0        0   0        1    44    1      0    1   99
##               6       0       11   0        3    60    0      0   22    0
##               7      19        4   0        0     0    0     50    0    6
##               8      35        0   0        0     0    0      1    0    0
##               9       3        7  17        8     2    0      1    4    0
```