1 Manifold Learning

1.1 Find a reasonably high-dimensional (> 100 dimensions) dataset that is interesting to you. It should also have at least 100 instances, but preferably more. Justify your choice.

I decided to use the Free Music Archive (FMA) [1][1] for my analysis. The FMA consists of Creative Commons licensed audio from 106,574 tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres. As part of the FMA a set of 518 pre-computed audio features for each track is available in the features.csv file. These features were computed with the librosa Python library[2]. The feature-set consist of many audio signal processing features like pitch chroma [3], harmonic network [4], Mel Frequency Cepstral Coefficients (MFCCs) [6] and many more. Many of these features were calculated for different sections of a track, therefore leading to a high overall number of features. The code for the feature extraction was made available by the FMA[3] and a full list of the features can be found in [1]. In terms of justification of my choice: I got interested in audio features as part of an AIML421 assignment last year[4]. The data set fulfills the assignment criteria of > 100 features and ≥ 100 instances, so I thought it would make for a good data set to play around with.

1.2 Using your choice of library, apply PCA to the data-set, and present your results. You should show visualisation(s) of the PCs found and also comment on the explained variance.

I selected a subset of 10186 tracks from the original data set by restricting the sample to the genres Pop, Folk, Jazz, Spoken, Country, Classical, Old-Time / Historic, Instrumental. This was done after some initial experiments showed that some genres had far more tracks than others and therefore dominated the picture, as well as the fact that some genres seemed to be almost "inseparable" from each other leading to very little difference between various plots, making for a boring discussion of result :). I thought it would be more interesting to write about plots that show a difference between various methods and therefore "cherry picked" the above genres. In terms of PCA results: as can be seen in Figure 1 in the Annex, there are "diminishing returns" in terms of explained variance as from around 30-40 principal components which equates to 65-70% explained variance. The curve tapers off at that point and the latest as from 300 principal components the amount of additional explained variance is marginal at best. As we can see in Figure 2, the first principal component explains 11.3% of variance, the second 6.7% and the third 4.2%. In terms of class separation the PCA doesn't lead to great results (visually). At least not for the first 3 components (c.f. Figure 2) as there is a very large overlap of the data points from various classes in the principal component space.

1.3 Pick one of the "classic" manifold learning methods and apply it to the same data. Compare the results to that of PCA (e.g. for an embedding with two dimensions). Highlight any differences between the two methods, and hypothesise why they may have occurred.

I chose the Isomap algorithm as a "classic" manifold learning methods. As visible from Figure 3 the class separation is far better in comparison to the results of a PCA in Figure 4. Particularly the genres "Old-Time / Historic" and "Classical" are comparatively well separated from the rest of the genres. The difference in results between the two methods can be explained due to the fact, that PCA

[1] `https://os.unil.cloud.switch.ch/fma/fma_metadata.zip`
[2] `https://github.com/librosa/librosa`
[3] `https://github.com/mdeff/fma/blob/master/features.py`
[4] `https://www.kaggle.com/c/comp309-2021`

is a linear subspace mapping method. It is restricted to linear transformations with the restriction of orthogonality between the principal components. The better class separation of Isomap would indicate that the sub-manifold we are trying to learn is non-linearly embedded in the observation space and Isomap (not being restricted to linear subspace mapping) is better equipped to learn this non-linear manifold embedding.

1.4 Pick one of the "modern" manifold learning methods and apply it to the same data. Compare and contrast the results to the two previous methods, with analysis of any differences seen.

I chose UMAP (c.f. Figure 5) as one of the "modern" manifold learning methods. As was the case already for Iosmap in the previous section, the genres "Old-Time / Historic" and "Classical" are comparatively well separated from the rest of the classes. The genre "Spoken" is partially separated from the rest of the classes. The explanation of difference in results between UMAP and PCA is pretty much identical with the result discussion of Isomap vs. PCA in the previous section. It boils down to the fact that UMAP is able to learn non-linear manifolds, which for this particular data set seems to be a necessary capability. The result comparison between Isomap and UMAP is less obvious. One thing that sticks out is the presence of "sub-clusters" in the UMAP results. Most pronounced in the case of the genre "Classical" but also in the case of the "Old-Time / Historic" genre. The two methods not only differ in how they work but as well how they are parameterized by default in the python packages that implement them. E.g. the sci-kit Isomap implementation used by default 5 neighbours for the local neighbourhood graph while in my experiments this value was set to 10 for UMAP. So there is a chance that one can parameterize UMAP in a way to exhibit results that match more closely those of Isomap (desirable or not). My second hypothesis about why we see more pronounced sub-clusters in the UMAP results is potentially linked to how the two different algorithms work. Isomap uses the euclidean distance to construct a local neighborhood graph and then calculates the geodesic distance between two points using the shortest path between these points on the graph (graph distance). Isomap aims to preserve that geodesic distance in a lower dimension. UMAP also uses some notion of local neighbourhood, but it does so by constructing a fuzzy topological structure informed by nearest neighbours. It aims to preserve this overall topological structure in lower dimensions. So the information about local neighbourhood is feeding directly into the optimisation procedure that performs the manifold learning. The assumption would be that some of the sub-clusters we see in the UMAP results are more distinct in topological terms (the graph representation itself) than they are in terms of a geodesic distance (measured as a distance along the graph).

1.5 Finally, pick one of the two manifold learning methods for further analyse. Your method will have "tunable parameters" — parameters that you can change to get different results. Pick one such parameter, and explore how sensitive the embedding is to changes in this parameter. You should explain the role of this parameter in the manifold learning algorithm, how you tested its effect, and show the results found.

I decided to look into the UMAP with DensMAP [5] algorithm for the purpose of parameter experimentation. The original UMAP algorithm assumes uniformly distributed on some manifold. DensMAP allows to preserve more information about the relative local density of data. The parameter is chose for my experiments is dens_frac. It "controls the fraction of epochs (between 0 and 1) where the density-augmented objective is used in densMAP. The first (1 - dens_frac) fraction of epochs optimize the original UMAP objective before introducing the density correlation term." (c.f. `https://umap-learn.readthedocs.io/en/latest/api.html`). Figures 6 to 10 visualize the results for the parameter values of 0.01, 0.05, 0.1, 0.2 and 0.5. Visually we can see, that with increasing values of dens_frac outliers get pushed more to the outskirts of the plot. The rest of the data points get pulled closer together not only towards their own cluster centers but also globally all clusters get pulled closer together. Non-surprisingly the plots for really small numbers of dens_frac resemble most the original UMAP results, as only a very small fraction of the optimisation epochs take into the

account the DenMAP correlation term. Increasing values of dens_frac are at some point detrimental in terms of class separation.

2 Clustering

2.1 Carry out hierarchical clustering with Euclidean distance and complete linkage

2.1.1 Describe the resulting clustering for 3–6 clusters.

The dendrograms depicting the results from the clustering can be found in Figure 11 to 14 in the Annex. I chose to use agglomerative clustering as the hierarchical clustering approach. In all cases (3-6 clusters) there is one dominant cluster with 6792 data points for 3-4 clusters and 6559 data points for 5-6 clusters. The additional clusters are rather at the fringes of the feature space and cover outliers.

2.1.2 Plot the first 2 principal components against each other with the colour argument set equal to the cluster labels. What can you deduce/observe about the clustering?

The results of the clustering can be seen in Figures 15 to 18. As already described in the previous section, there is one large cluster (visually at the centre of the principal component space). For 3 and 4 clusters the additional cluster just cover isolated data points at the fringes. These data points are not necessarily very close to each other in the principal component space. After 5 and 6 clusters one additional "sizable" cluster is added but with a significant overlap with the dominant cluster in the principal component space. Given the lack of cluster separation along the axes of the first principal components, there doesn't seem to be a connection between hierarchical clustering with euclidean distance and complete linkage and PCA.

2.2 Repeat the cluster analysis using correlation-based distance and complete linkage. NB: you will need to precompute the correlations and pass them into your clustering method. Compare the clusters with those found above.

The results of the clustering can be seen in Figures 19 to 22 In the paradigm of correlation-based clustering, observations with highly correlated features are likely to end up in the same cluster, even if these observations share a large euclidean distance. Particularly in Figure 19 we can see that the cluster separation happens along the first two principal components. Cluster 0 and 2 sit on the positive part of the second principal component, with cluster 0 being located on the negative part of principal component one and cluster 2 on the positive part. Cluster 1 is located in the negative part of the second principal component and spans across the whole range of the first principal component. The picture is less clear cut with higher number of clusters (c.f. Figures 20 to 22) but at least some of the cluster boundaries are still more or less aligned with the dividing lines between the positive and negative parts of the first two principal components. Correlation measures both the strength and direction of the linear relationship between two variables. Correlation is the standardized form of covariance. Given that PCA tries to capture the eigenvectors of the co-variances between variables it is not surprising that the clusters from hierarchical clustering based on a correlation-based distance matrix are somehow aligned with the principal components of a PCA.

2.3 Finally, carry out K-means clustering for 3–6 clusters. Compare the clusters of K-means with that of the above two approaches. Which of the hierarchical clustering results is more similar to that of K-means? Why?

The results of the clustering can be seen in Figures 23 to 26. It is fairly obvious that the correlation-based clustering results are more similar to the K-means results.

According to [2] K-means can be seen as a sparse PCA or a form of the PCA where the continuous principal components take a discrete form (intervals along the continuous principal components). C.f. as well explanations in `https://stats.stackexchange.com/questions/183236/`

what-is-the-relation-between-k-means-clustering-and-pca. This pattern is very well visible in Figures 23 to 26. It is quite similar to what was described in the previous sections. The clusters are initially located at a positive or negative part of one or the other principal component. With increasing cluster numbers the continuous principal components get further "subdivided" into smaller sections or intervals to "accommodate" an increasing number of cluster. E.g. for 6 clusters we can observe that the positive part of the second principal component is divided into three sections along the first principal component (c.f. Figure 26).

3 Regression

Repeat the analysis for the Credit dataset (we have done it in the lecture on Week 7) with pairwise interactions of the features.

3.1 How many predictors are there, i.e. what is p?

There 65 predictors (with the intercept being excluded).

3.2 How did you generate your training and test sets?

The training and test data set has been created with a 50:50 random split with seed 42, as can be seen in the listing below:

```
set.seed(42)
train=sample(1:nrow(X),nrow(X)/2)
test=-train
```

train and test are then used as row selectors for the data frame to refer to the training and test data set.

3.3 Select the tuning parameter for the ridge regression model using cross-validation, and show the process.

Based on http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-r.html I selected lambda using 10-fold cross validation using the training data. The best lambda value that was estimated through this was $\lambda = 1.123324$ The resulting MSE on the test set for this lambda value was MSE=6669.864. In comparison, the resulting MSE on the test set for the plain vanilla unconstrained regression model was MSE=6444.493.

```
# load the libraries we need
library(glmnet)
#read in the data
Credit= read.csv('Downloads/ass2_data_2022_2/part3/Credit.csv', header=T)[,-1]
#define the predictor matrix
X = model.matrix(Balance~.*.,Credit)[,-1]
#define targets
y=Credit$Balance
#set random see for reproducibility
set.seed(42)
#define row indices for training and test 50:50 split
train=sample(1:nrow(X),nrow(X)/2)
test=-train
#grid search with lambda values
grid = 10^seq(3,-1,length= 100)
ridge.mod = glmnet(X[train,],y[train],alpha=0,lambda=grid,thresh = 1e-10)
#source of the following code
#http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-r.html
```

```
# Fit ridge regression model on training data
cv.out = cv.glmnet(X[train,],y[train], alpha = 0,lambda=grid,thresh = 1e-10)
# Select lamda that minimizes training (or rather validation) MSE
bestlam = cv.out$lambda.min
bestlam
#plot results of CV
plot(cv.out)
# Use best lambda to predict test data
ridge.pred = predict(ridge.mod, s = bestlam, newx = X[test,])
# Calculate test MSE
mean((ridge.pred - y[test])^2)
# Calculate test MSE for the plain vanilla linear regression model
ridge.pred.lm = predict(ridge.mod, s = 0, newx = X[test,], exact = T, x=X[train,],y
mean((ridge.pred.lm - y[test])^2)
```

The results from all the cross validation runs are visualized in Figure 27

3.4 Select the tuning parameter for the lasso regression model using cross-validation, and show the process. How many features have been selected by the lasso?

Based on `http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-r.html` I selected lambda using 10-fold cross validation using the training data. The best lambda value that was estimated through this was $\lambda = 1.353048$ The resulting MSE on the test set for this lambda value was MSE=5582.306 and the number of non-zero coefficients was 34.

```
# load the libraries we need
library(glmnet)
#read in the data
Credit= read.csv('Downloads/ass2_data_2022_2/part3/Credit.csv', header=T)[,-1]
#define the predictor matrix
X = model.matrix(Balance~.*.,Credit)[,-1]
#define targets
y=Credit$Balance
#set random see for reproducibility
set.seed(42)
#define row indices for training and test 50:50 split
train=sample(1:nrow(X),nrow(X)/2)
test=-train
#grid search with lambda values
grid = 10^seq(3,-1,length= 100)
lasso.mod = glmnet(X[train,],y[train],alpha=1,lambda=grid,thresh = 1e-10)
#source of the following code
#http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-r.html
# Fit lasso regression model on training data
cv.out = cv.glmnet(X[train,],y[train], alpha = 1,lambda=grid,thresh = 1e-10)
# Select lamda that minimizes training (or rather validation) MSE
bestlam = cv.out$lambda.min
bestlam
#plot results of CV
plot(cv.out)
# Use best lambda to predict test data
lasso.pred = predict(lasso.mod, s = bestlam, newx = X[test,])
# Calculate test MSE
mean((lasso.pred - y[test])^2)
lasso.coef = predict(lasso.mod, type = "coefficients", s = bestlam, newx = X[test,]
```

```
length(lasso.coef[lasso.coef != 0]) # Display only non-zero coefficients
```

The results from all the cross validation runs are visualized in Figure 28.

3.5 Compare and discuss the final form of the model from the linear regression, ridge regression, and lasso regression.

The final form of the models of OLS, ridge regression and lasso can be found in the table below. All models were trained on the full data set. In case of ridge regression and lasso with the lambda values that were determined in the previous sections. OLS and ridge regression have 66 non-zero coefficients (incl. intercept), while lasso has 34 non-zero coefficients. OLS has 27 coefficient above 0.01, ridge 0.28 and lasso 17. As one can see, the L1 penalty from lasso leads to a sparse model with many coefficients being set to zero, whereas the L2 penalty of the ridge regression leads to more small coefficients. Unfortunately it is not straight forward to calculate p-values for the results of ridge regression and lasso regression, so that further comparison of the model coefficients is tricky. One more noteworthy aspect is, that the sign of some coefficients flipped in comparison to the OLS results as part of applying the ridge regression or lasso. The same phenomenon exists as well between lasso and ridge regression results.

Predictor	coefficient from OLS	coefficient from ridge	coefficient from lasso
(Intercept)	-305.432	-297.346	-218.496
Income	-1.382	-1.949	-2.474
Limit	0.113	0.061	0.106
Rating	-0.398	0.577	.
Cards	30.661	17.276	9.884
Age	1.092	1.182	.
Education	4.538	2.667	.
GenderFemale	-53.579	-61.816	.
StudentYes	176.740	120.103	151.258
MarriedYes	65.123	50.144	.
EthnicityAsian	54.956	57.454	.
EthnicityCaucasian	4.440	6.350	.
Income:Limit	-0.001	0.000	.
Income:Rating	-0.005	-0.007	-0.012
Income:Cards	-0.216	-0.229	-0.067
Income:Age	0.028	0.009	-0.007
Income:Education	-0.122	-0.125	-0.049
Income:GenderFemale	-0.687	-0.691	-0.075
Income:StudentYes	-2.003	-2.055	-1.311
Income:MarriedYes	-0.144	-0.145	.
Income:EthnicityAsian	0.636	0.293	.
Income:EthnicityCaucasian	0.840	0.389	.
Limit:Rating	0.000	0.000	0.000
Limit:Cards	0.006	0.007	0.001
Limit:Age	0.000	0.000	.
Limit:Education	0.000	0.002	0.000
Limit:GenderFemale	0.028	0.016	.
Limit:StudentYes	0.140	0.062	0.071
Limit:MarriedYes	-0.019	-0.001	.
Limit:EthnicityAsian	-0.017	0.001	0.001
Limit:EthnicityCaucasian	0.008	0.005	.
Rating:Cards	-0.073	-0.064	.
Rating:Age	-0.008	-0.008	.
Rating:Education	0.012	-0.004	.
Rating:GenderFemale	-0.248	-0.084	.
Rating:StudentYes	-0.877	0.244	.
Rating:MarriedYes	0.250	-0.004	.
Rating:EthnicityAsian	0.166	-0.028	.
Rating:EthnicityCaucasian	-0.245	-0.128	.
Cards:Age	-0.024	0.000	.
Cards:Education	-0.799	-0.383	.
Cards:GenderFemale	9.860	10.311	3.125
Cards:StudentYes	4.572	4.580	3.471
Cards:MarriedYes	-2.530	0.311	.
Cards:EthnicityAsian	-1.895	-5.343	.
Cards:EthnicityCaucasian	4.932	1.430	3.081
Age:Education	-0.043	-0.077	-0.030
Age:GenderFemale	0.533	0.527	.
Age:StudentYes	0.096	-0.028	.
Age:MarriedYes	0.166	-0.030	.
Age:EthnicityAsian	0.098	0.314	0.080
Age:EthnicityCaucasian	-0.331	-0.095	.
Education:GenderFemale	-1.728	-1.489	-0.776
Education:StudentYes	-0.648	2.143	0.144
Education:MarriedYes	-3.204	-1.967	.
Education:EthnicityAsian	-3.346	-3.495	.
Education:EthnicityCaucasian	2.779	2.198	0.646
GenderFemale:StudentYes	-4.068	-12.707	.
GenderFemale:MarriedYes	-1.891	3.498	.
GenderFemale:EthnicityAsian	-4.987	-6.923	.
GenderFemale:EthnicityCaucasian	10.891	7.454	.
StudentYes:MarriedYes	6.875	5.785	3.163
StudentYes:EthnicityAsian	6.354	1.515	.
StudentYes:EthnicityCaucasian	-1.724	4.846	.
MarriedYes:EthnicityAsian	-5.311	0.131	.
MarriedYes:EthnicityCaucasian	-28.839	-29.207	-6.212

3.6 Compare the test errors for the linear model, ridge regression model, and lasso model.

The MSE on the test set for the plain vanilla unconstrained regression model was MSE=6444.493, the test MSE for the ridge regression with the lambda determined by cross validation was MSE=6669.864 and the test MSE for the lasso regression with lambda determined by cross validation was MSE=5582.306. The Lasso regression had the best test MSE, followed by the unconstrained model and then the ridge regression. Intuitively one would have expected the ridge regression to perform better than the unconstrained model, given how many predictor variables are present and the risk that presents in terms of over-fitting on the training data. What seems to have happened instead is that despite the cross validation there was an over-fitting of the lambda and or the coefficients on the training data in case of the ridge regression. E.g. when playing around with lambda, 0.4 would actually lead to a better MSE on the test set (6423.726) than the unconstrained regression. This value is still far worse than the lasso. It seems that for this data set feature selection in the sense that lasso performs it is the best approach. Once a feature has been eliminated (coefficient set to zero) the idiosyncratic noise in the training set (for this feature) can't negatively influence anymore the prediction performance on the test set.

3.7 Plot a comparison of the test predictions for the three approaches.

A comparison of the predictions can be found in Figure 29 in the Annex. One can see that the results of the lasso regression (orange) are on average closer to the diagonal. There are less outlier far off the diagonal with lasso, than there are for the plain vanilla regression (green) or the ridge regression (blue). This is particularly true for all the data point around y=0.

References

[1] Kirell Benzi et al. "FMA: A Dataset For Music Analysis". In: *CoRR* (2016). URL: `http://arxiv.org/abs/1612.01840`.

[2] Chris Ding and Xiaofeng He. "K-means clustering via principal component analysis". In: *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. Banff, Alberta, Canada: ACM, 2004, p. 29.

[3] M. Goto. "A chorus section detection method for musical audio signals and its application to a music listening station". In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.5 (2006), pp. 1783–1794.

[4] Christopher Harte, Mark Sandler, and Martin Gasser. "Detecting Harmonic Change in Musical Audio". In: *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*. AMCMM '06. Santa Barbara, California, USA: Association for Computing Machinery, 2006, pp. 21–26.

[5] Ashwin Narayan, Bonnie Berger, and Hyunghoon Cho. "Density-Preserving Data Visualization Unveils Dynamic Patterns of Single-Cell Transcriptomic Variability". In: *bioRxiv* (2020).

[6] L. R. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Annex

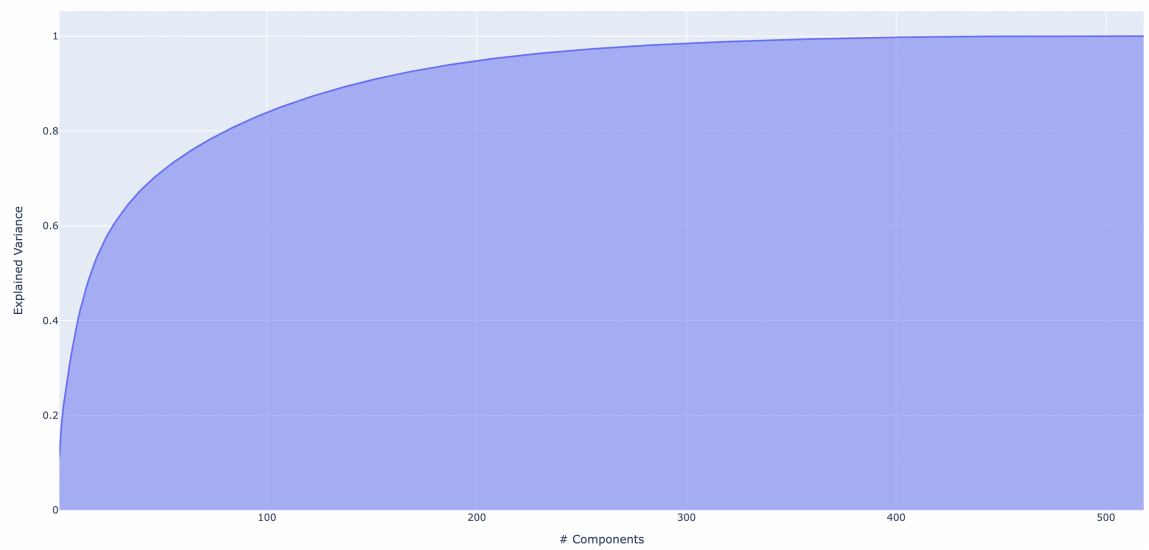

Figure 1: Explained variance as a function of number of principal components

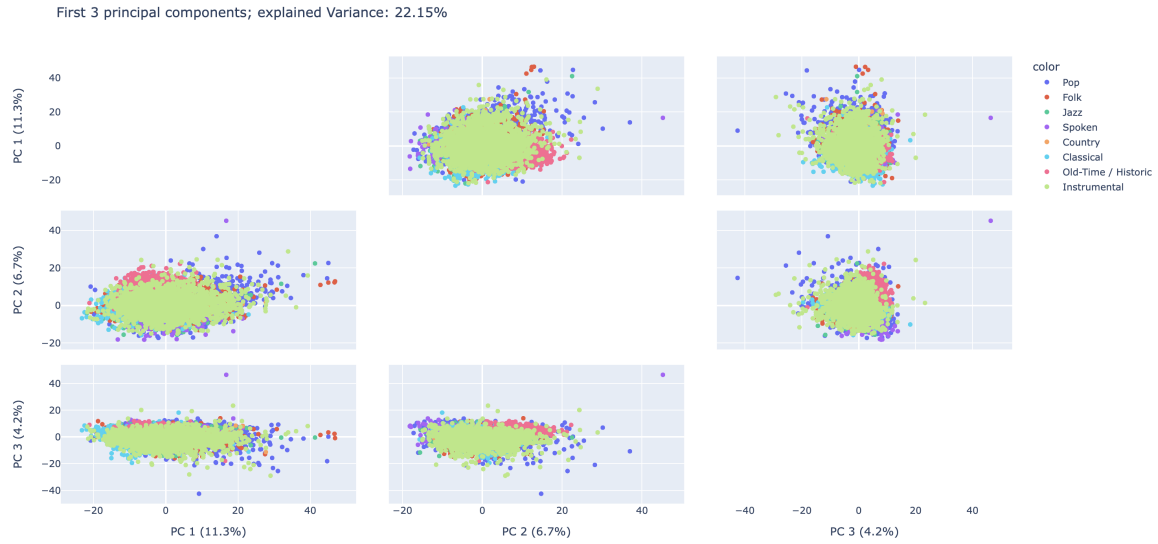

Figure 2: Scatter plot matrix of the fist 3 principal components

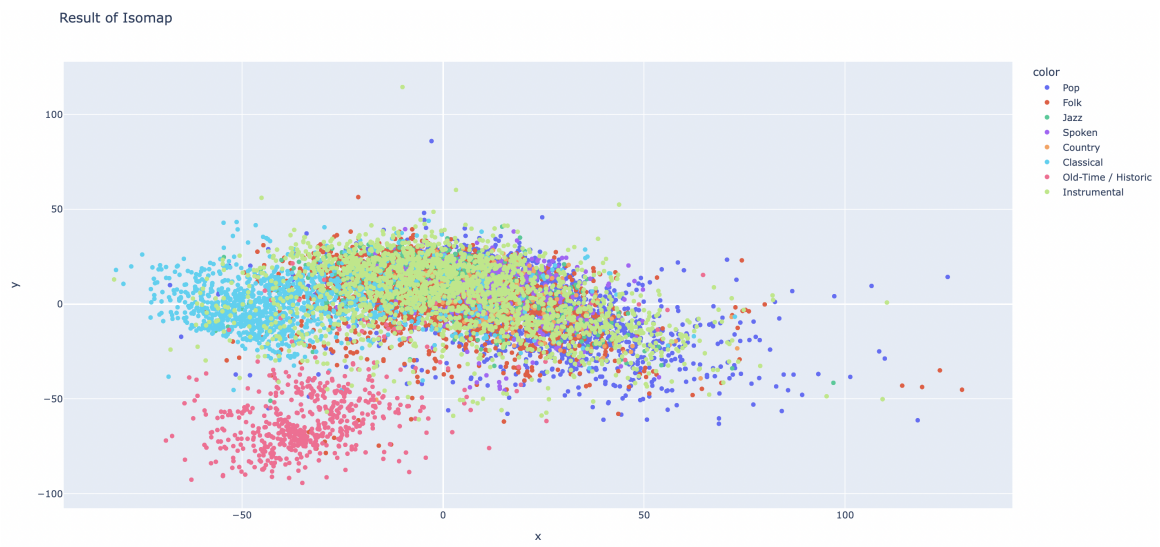

Figure 3: Scatter plot of the Isomap embedding

Figure 4: Scatter plot of the fist 2 principal components of a PCA

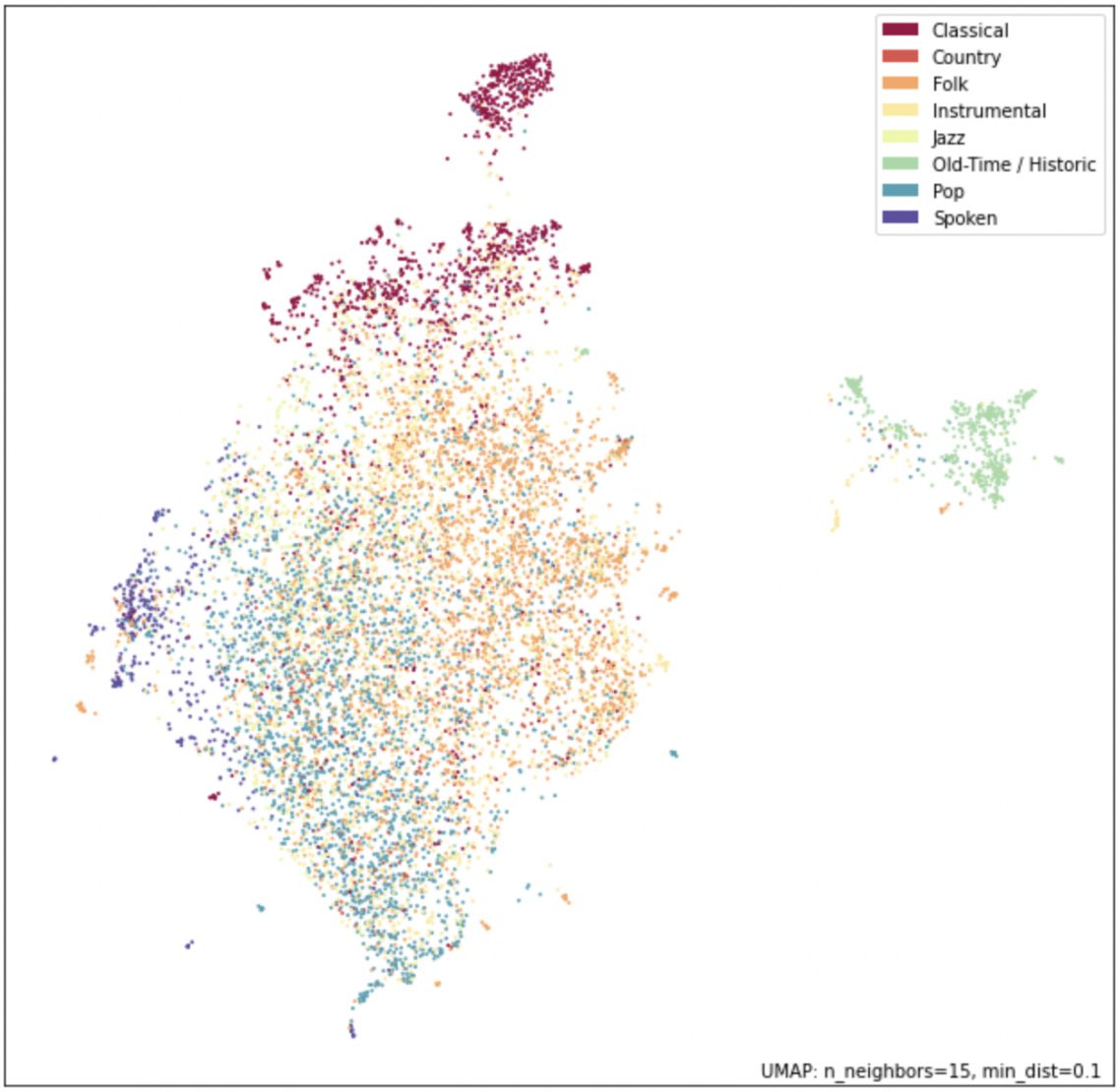

Figure 5: Scatter plot of the UMAP embedding

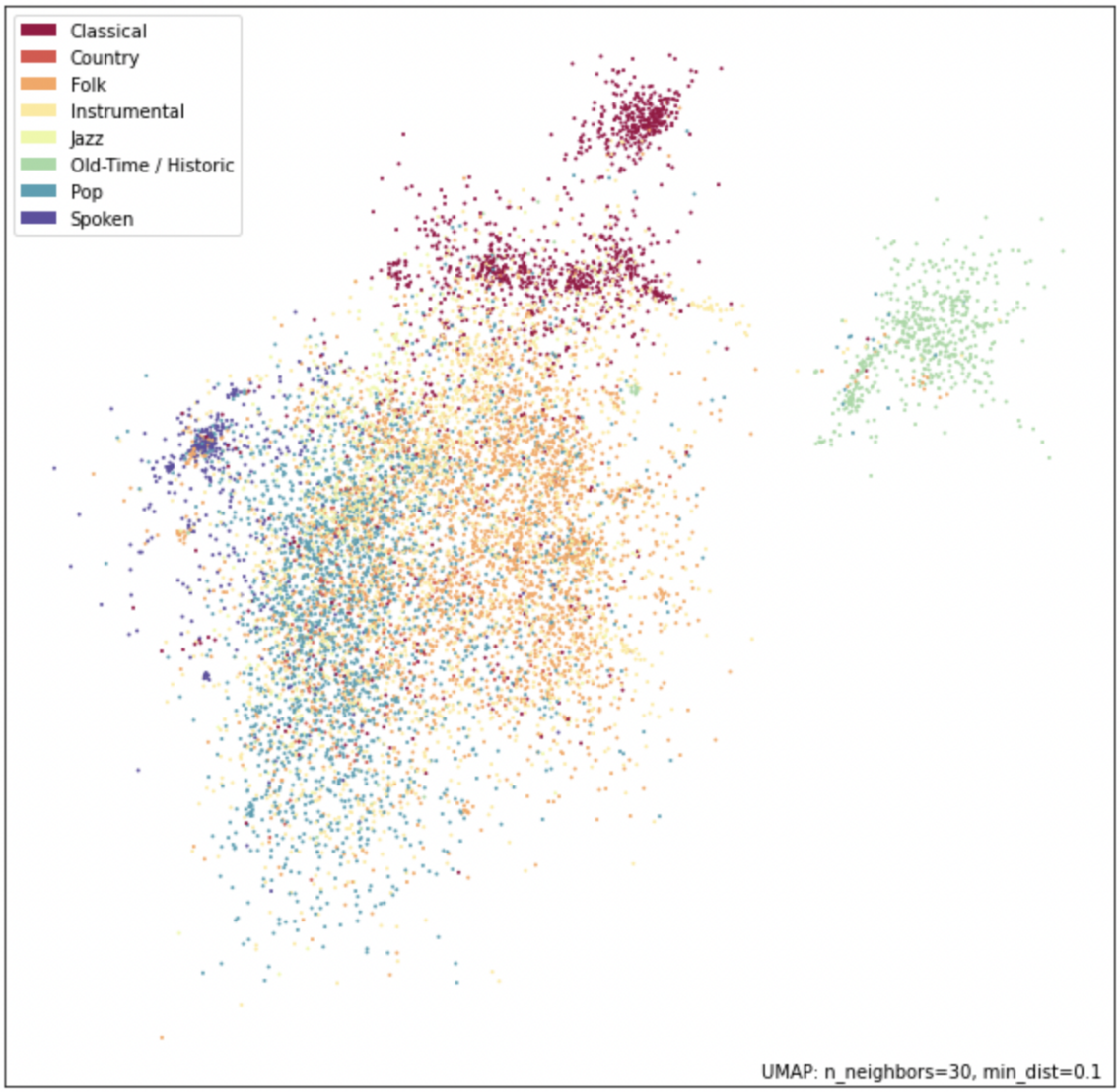

Figure 6: Scatter plot of the UMAP with DensMAP embedding, density fraction = 0.01

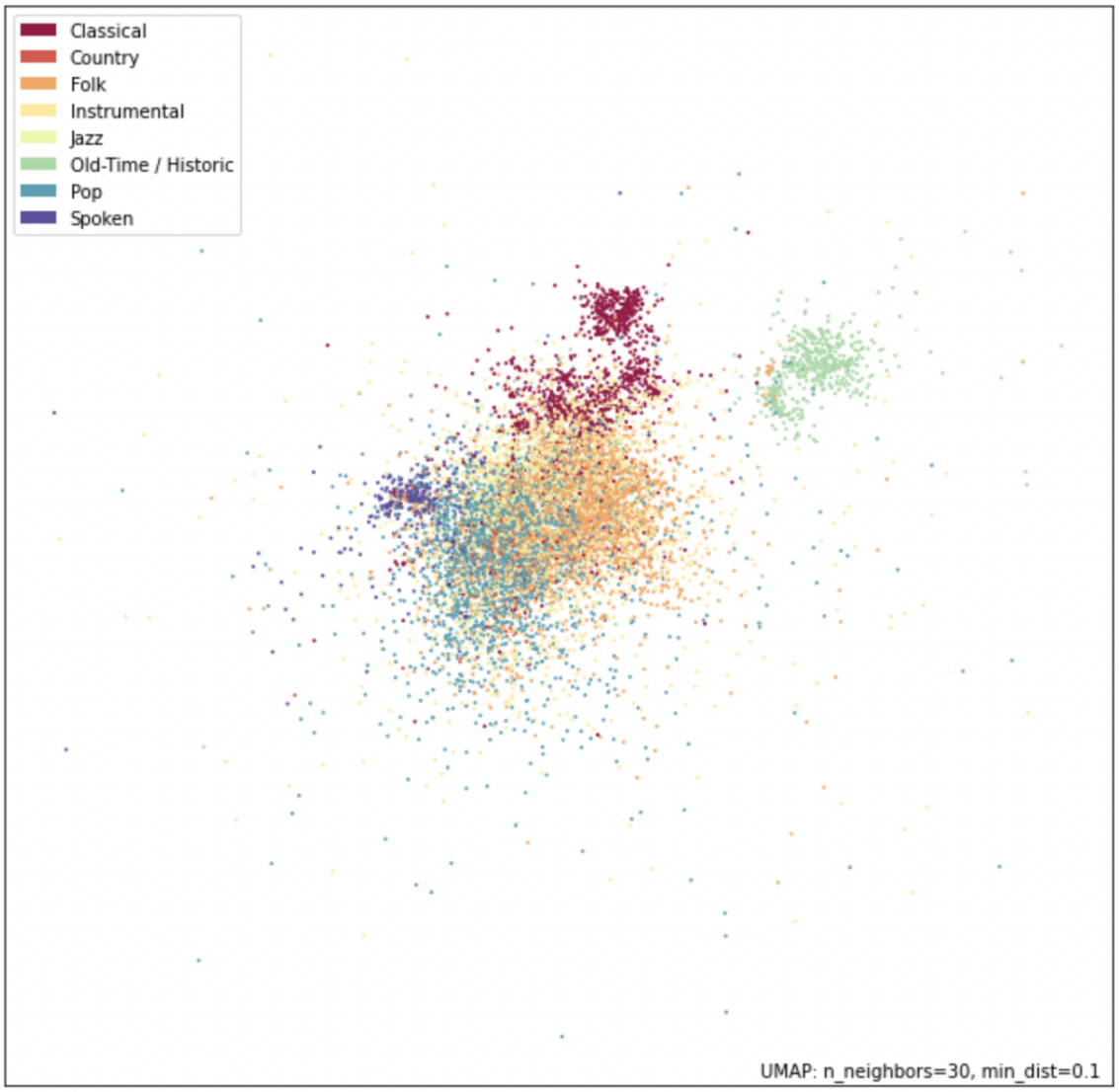

Figure 7: Scatter plot of the UMAP with DensMAP embedding, density fraction = 0.05

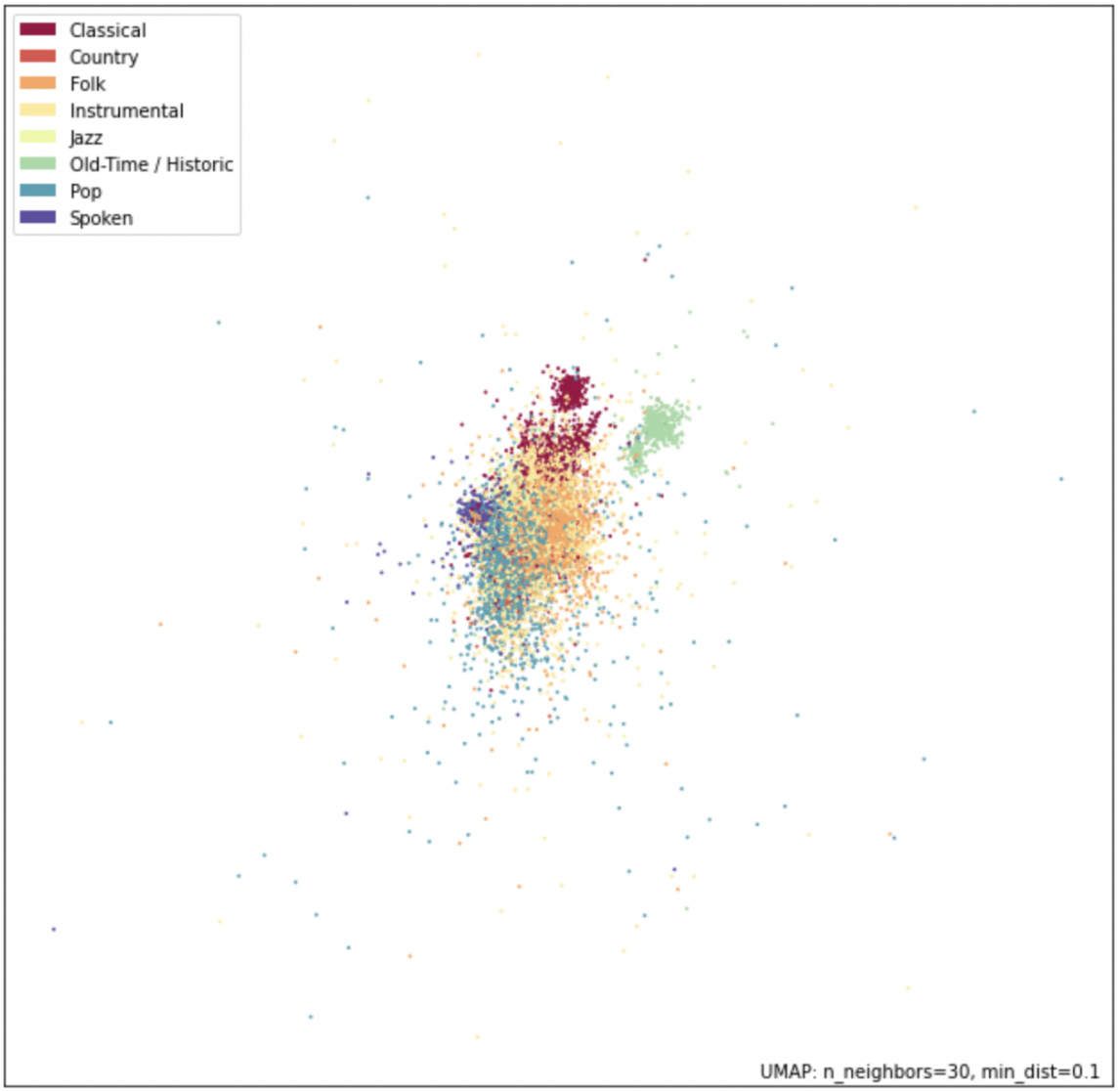

Figure 8: Scatter plot of the UMAP with DensMAP embedding, density fraction = 0.1

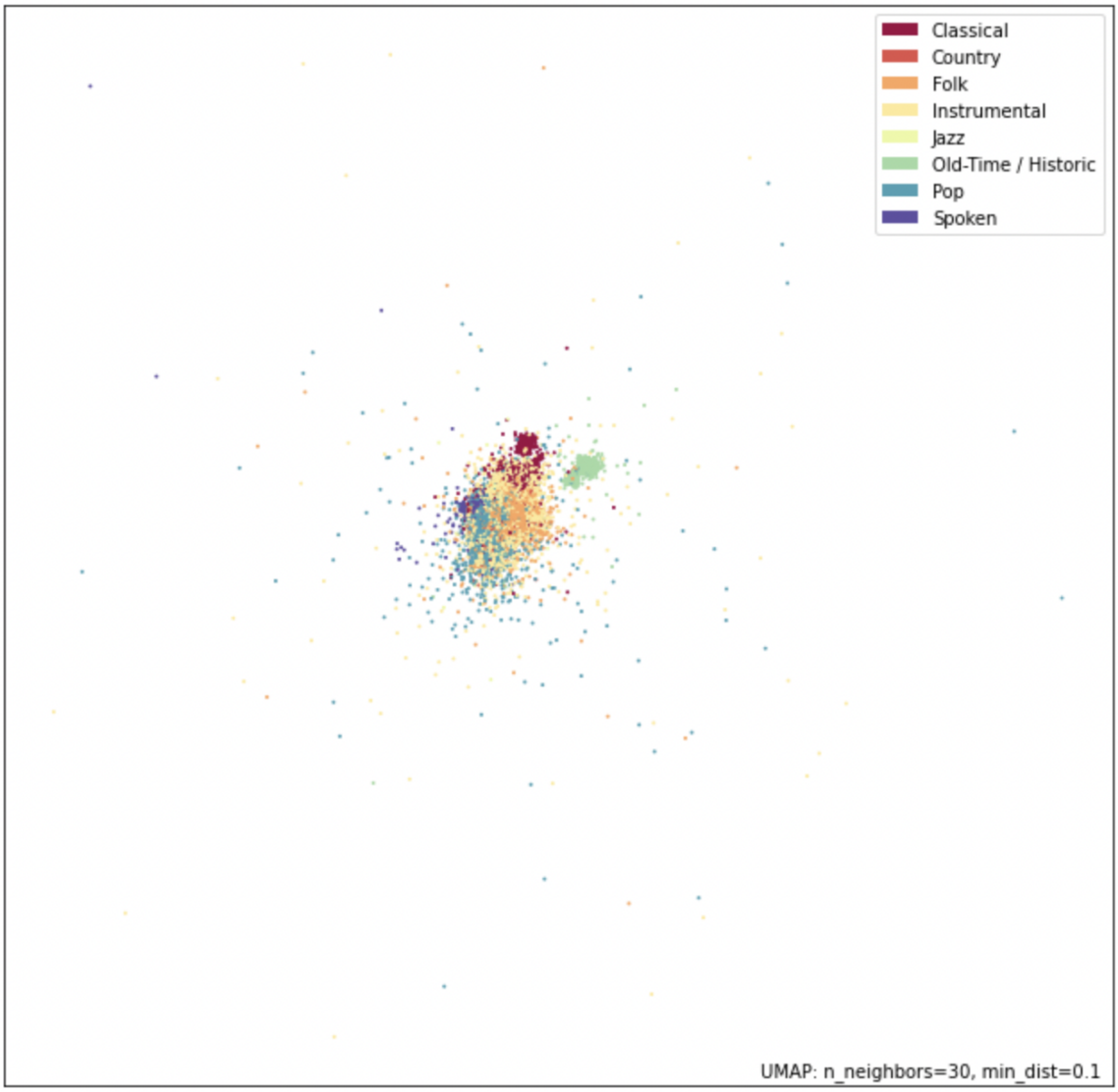

Figure 9: Scatter plot of the UMAP with DensMAP embedding, density fraction = 0.2

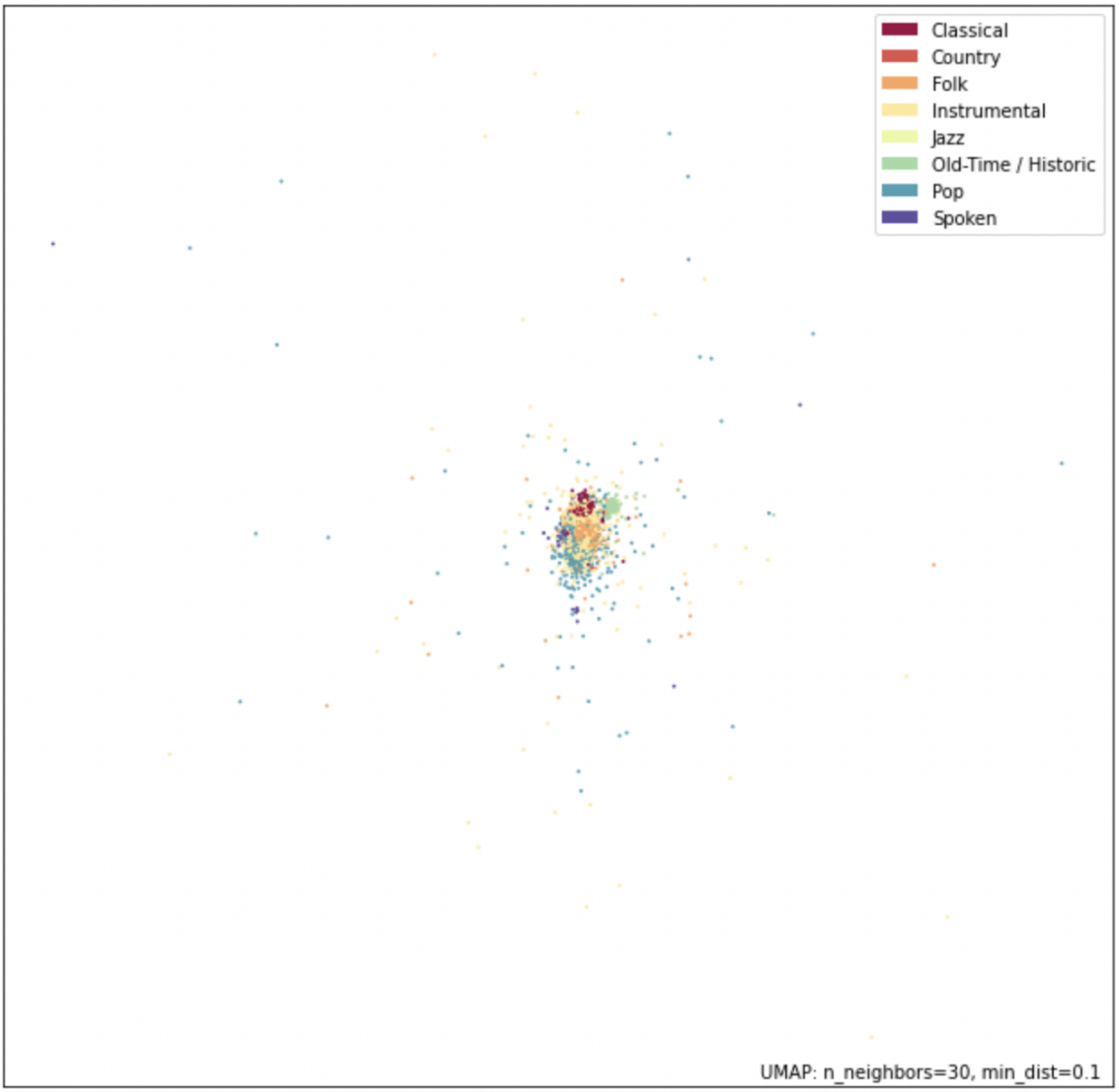

Figure 10: Scatter plot of the UMAP with DensMAP embedding, density fraction = 0.5

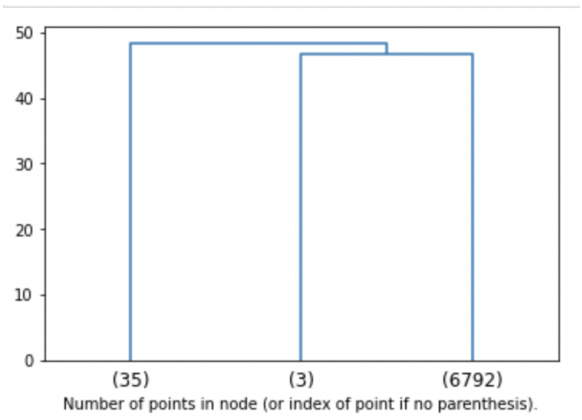

Figure 11: Hierarchical clustering, 3 clusters

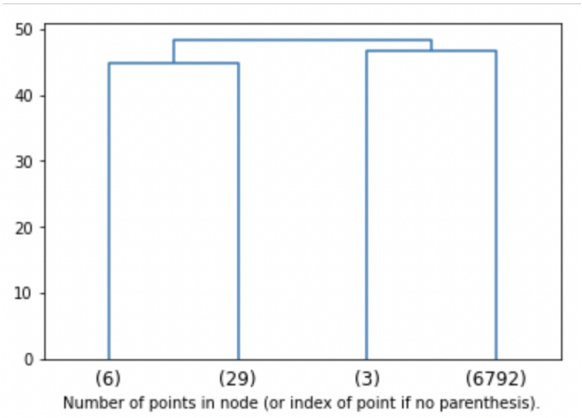

Figure 12: Hierarchical clustering, 4 clusters

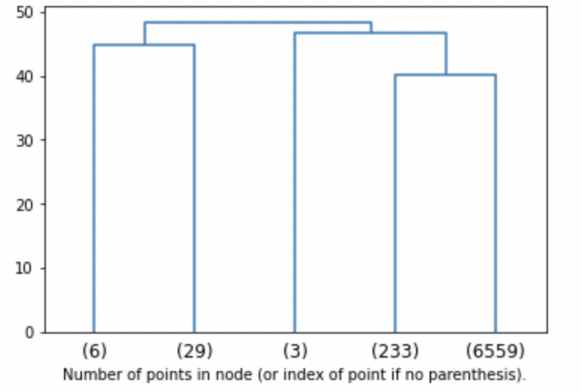

Figure 13: Hierarchical clustering, 5 clusters

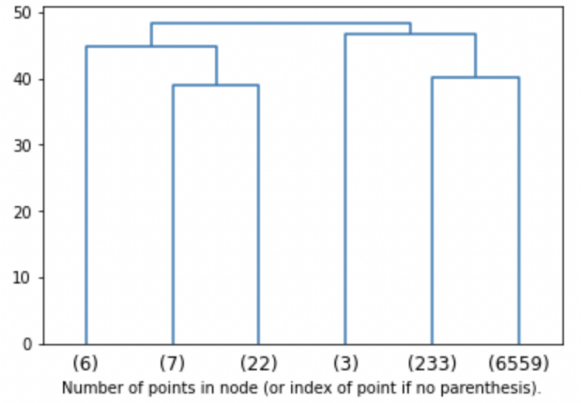

Figure 14: Hierarchical clustering, 6 clusters

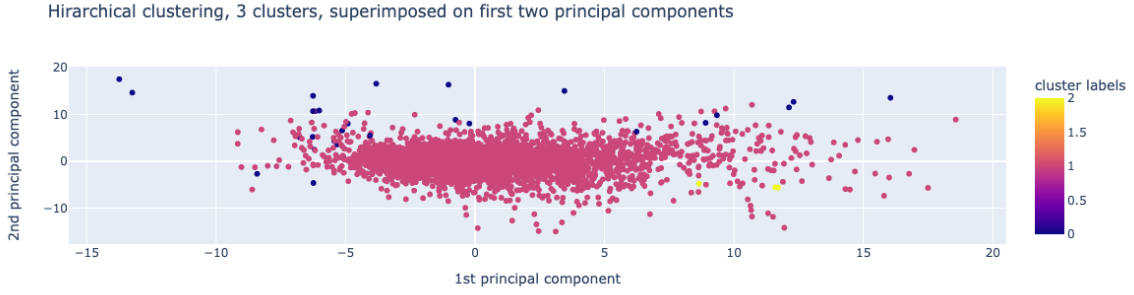

Figure 15: Hierarchical clustering, 3 clusters, superimposed on first 2 principal components of a PCA

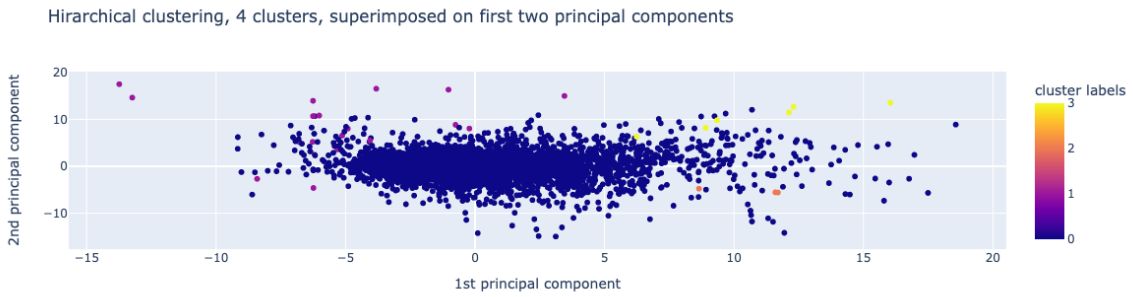

Figure 16: Hierarchical clustering, 4 clusters, superimposed on first 2 principal components of a PCA

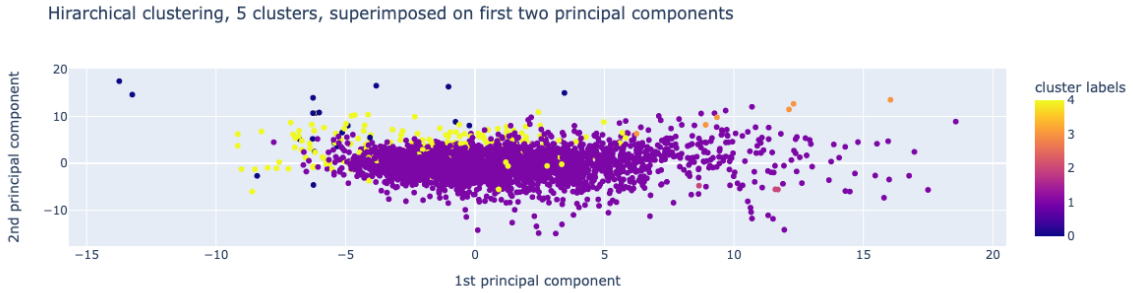

Figure 17: Hierarchical clustering, 5 clusters, superimposed on first 2 principal components of a PCA

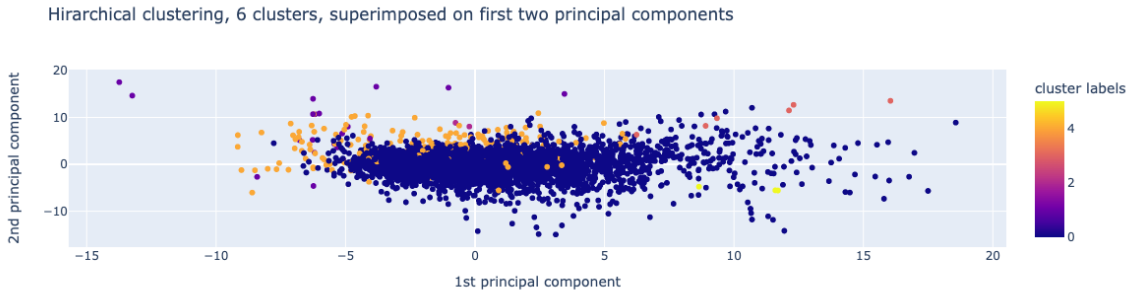

Figure 18: Hierarchical clustering, 6 clusters, superimposed on first 2 principal components of a PCA

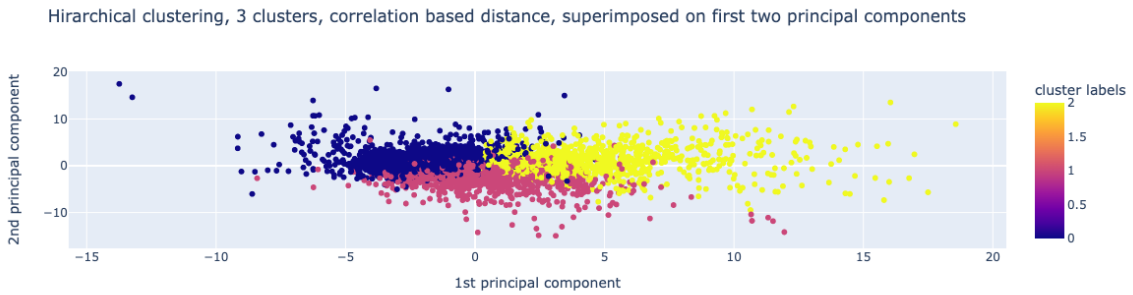

Figure 19: Hierarchical clustering (correlation based), 3 clusters, superimposed on first 2 principal components of a PCA

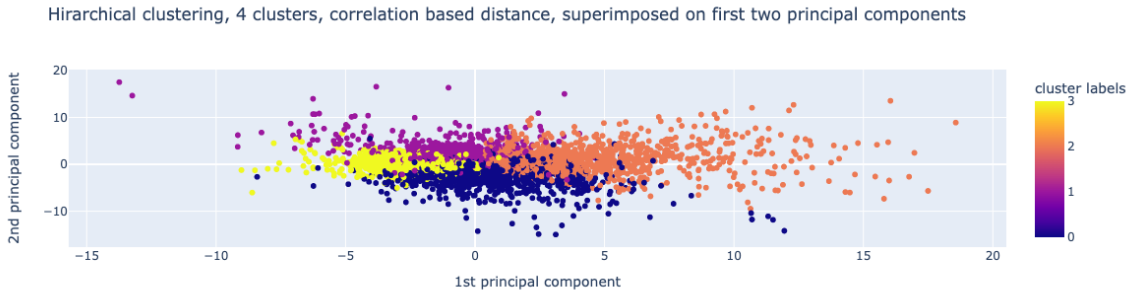

Figure 20: Hierarchical clustering (correlation based), 4 clusters, superimposed on first 2 principal components of a PCA

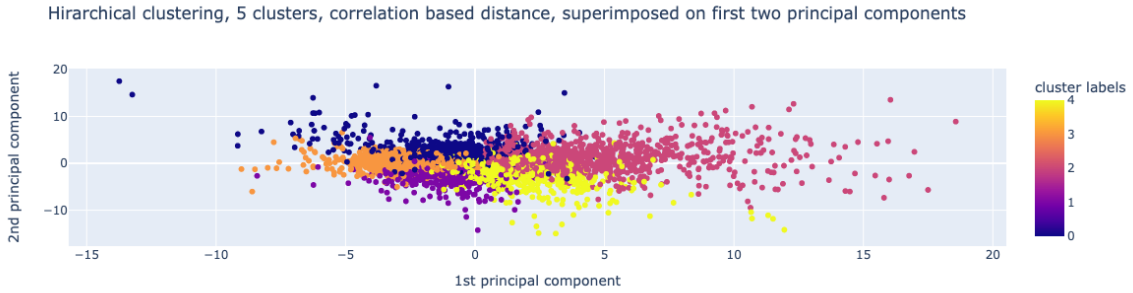

Figure 21: Hierarchical clustering (correlation based), 5 clusters, superimposed on first 2 principal components of a PCA

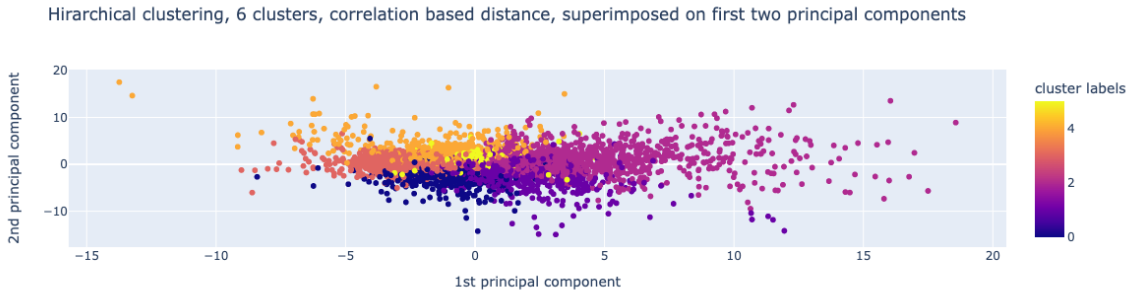

Figure 22: Hierarchical clustering (correlation based), 6 clusters, superimposed on first 2 principal components of a PCA

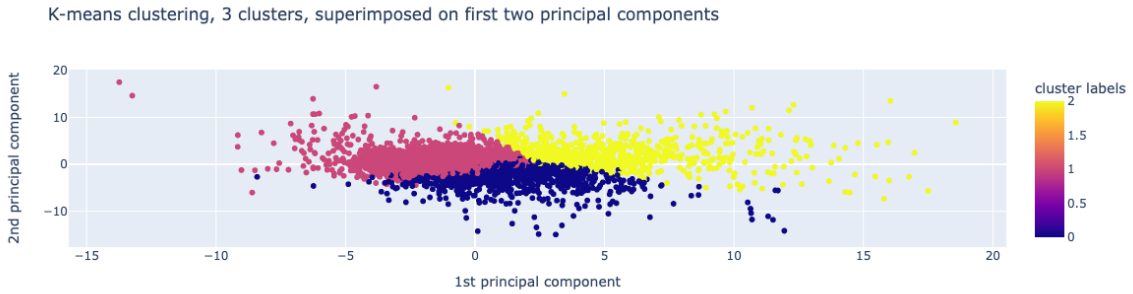

Figure 23: K-means, 3 clusters, superimposed on first 2 principal components of a PCA

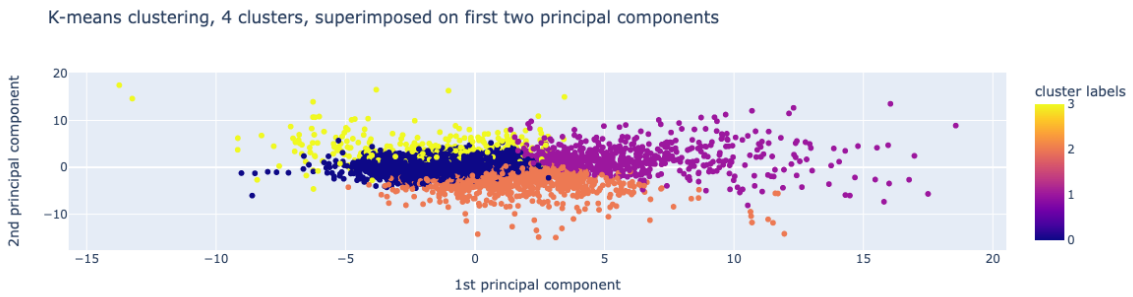

Figure 24: K-means, 4 clusters, superimposed on first 2 principal components of a PCA

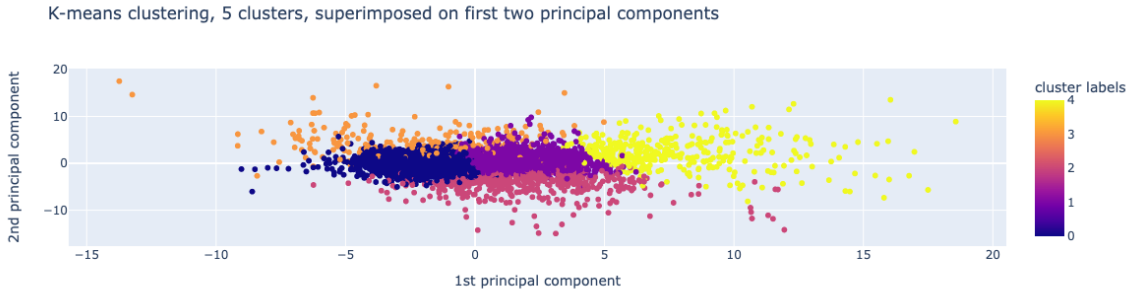

Figure 25: K-means, 5 clusters, superimposed on first 2 principal components of a PCA

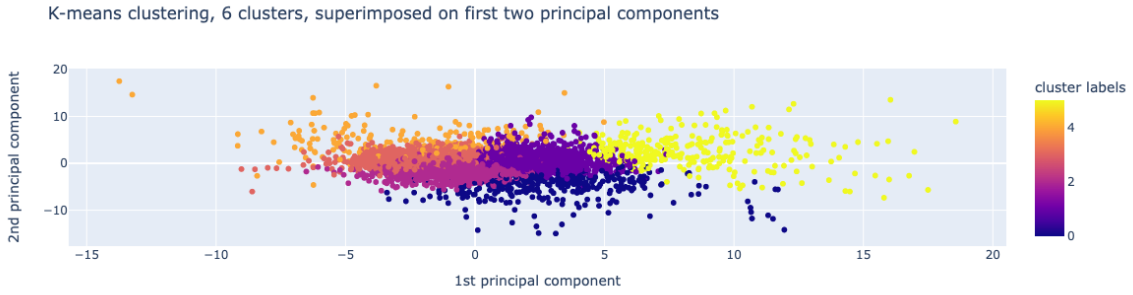

Figure 26: K-means, 6 clusters, superimposed on first 2 principal components of a PCA

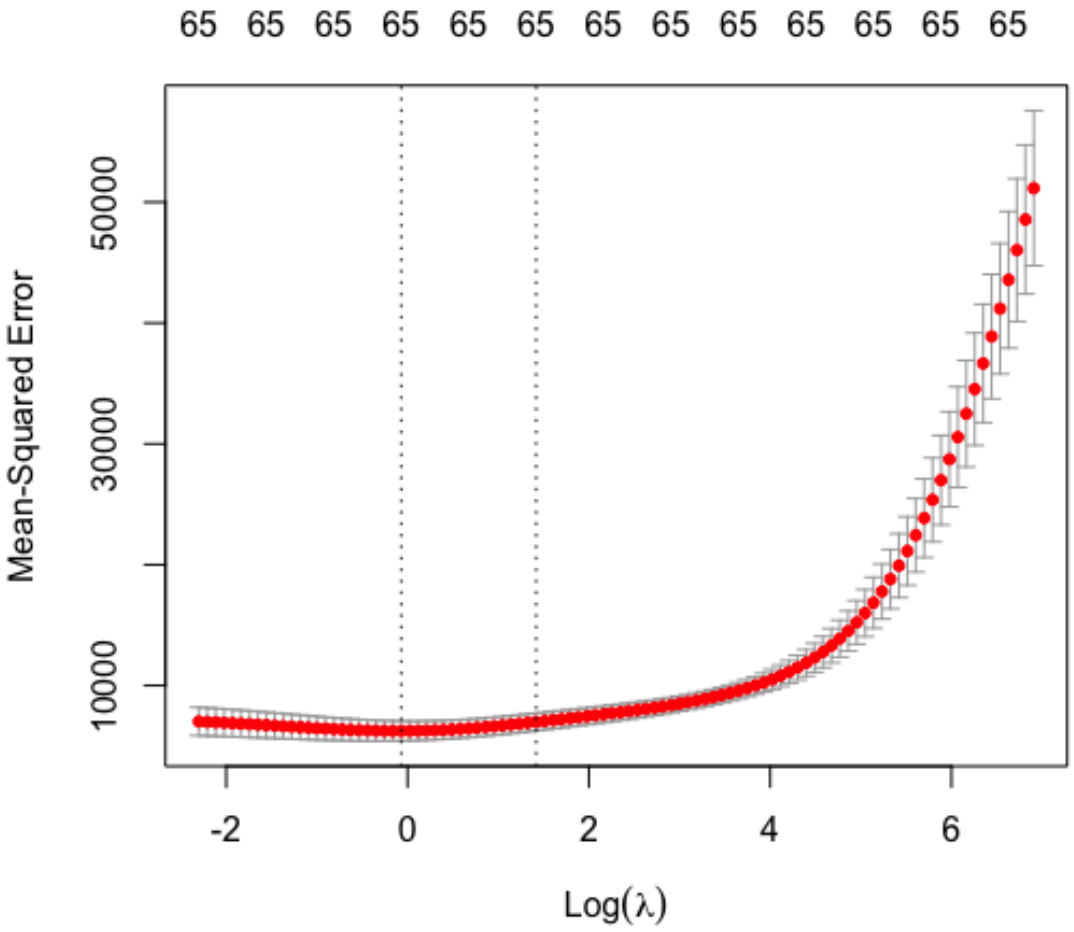

Figure 27: Cross-validation results for lambda in ridge regression

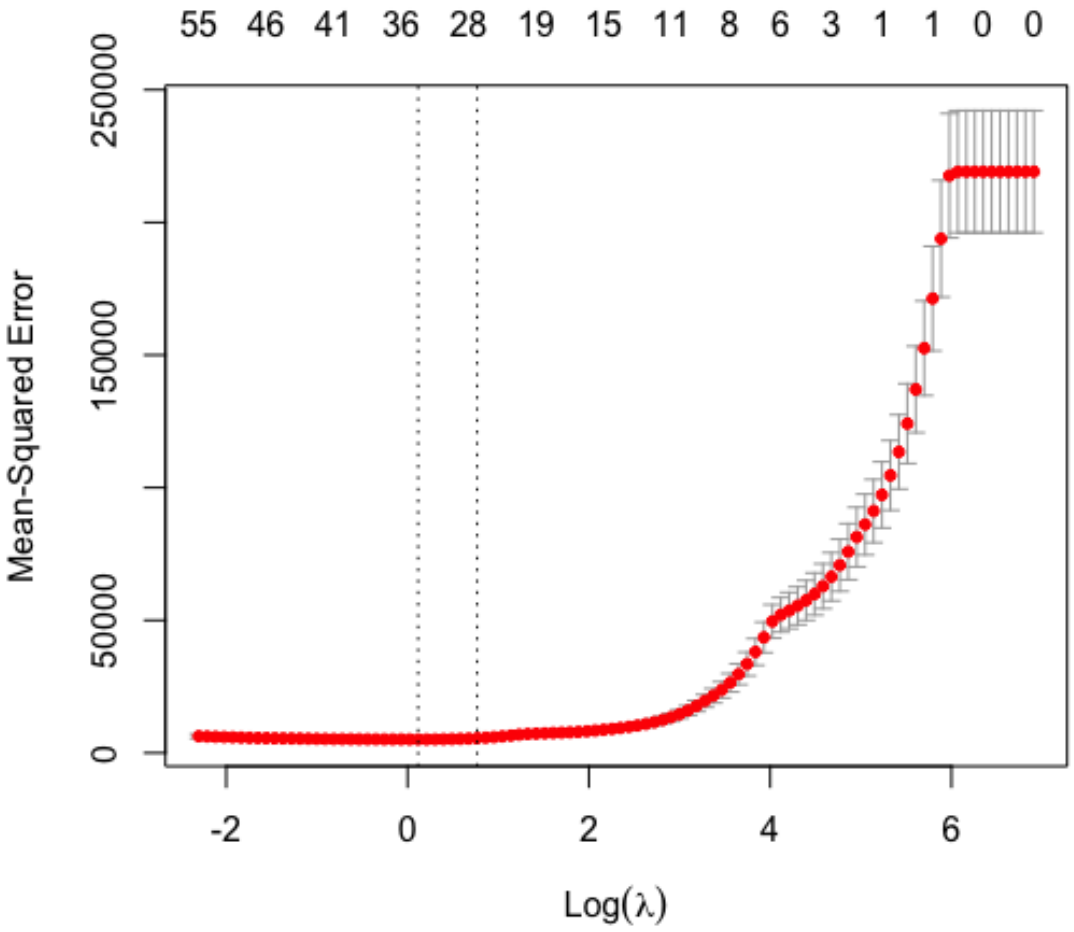

Figure 28: Cross-validation results for lambda in lasso regression

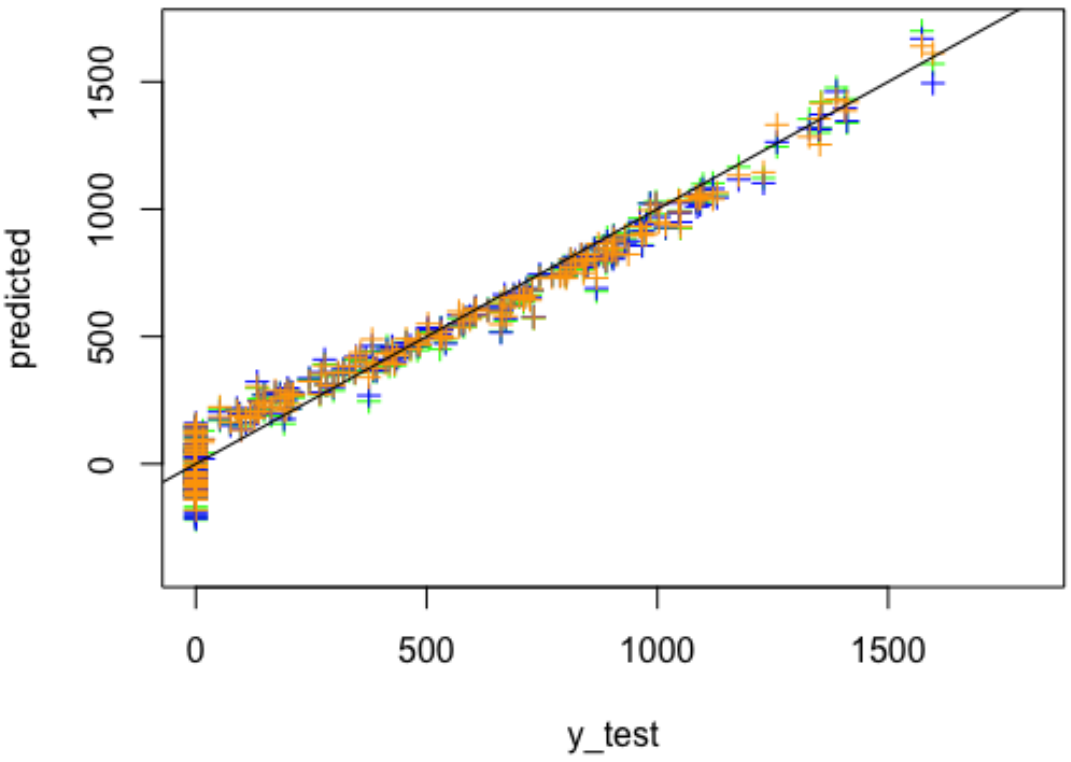

Figure 29: Comparison predictions plain vanilla, ridge and lasso regression