COMP0050

Machine Learning with Applications in Finance

Coursework

Question 1

Bank Default Prediction

Question 2

Equity Return Clustering

Date:

March 2023

QUESTION 1: Bank Default Prediction

1 Introduction

The objective of this task is to build a predictive model that can accurately determine whether a bank is likely to default or not (classification). To accomplish this, we will evaluate the performance of various methods, such as logistic regression and classification trees/forests, to determine which approach is most effective at predicting defaults.

Additionally, we will consider the issue of unbalanced data and explain our rationale for focusing on a specific subset of features or banks in our analysis. Through this process, we aim to develop a reliable model that can provide valuable insights into the likelihood of default for banks.

2 Methodology

To achieve this task, we will compare 3 different supervised learning models: **Logistic Regression, Decision Trees and Random Forests**. Each model will be tested with unbalanced data and balanced data (oversampling and undersampling) and classification reports and confusion matrices will be compared to find the best performing model.

2.1 Confusion Matrices

A confusion matrix is a table used to evaluate the performance of a classification model by comparing the predicted classes against the actual classes. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) of a model's predictions. This will allow us to easily visualise the performance of each model.

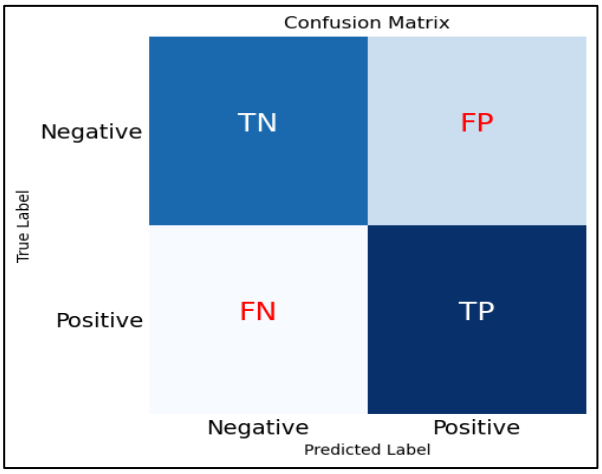


Figure 1 - Confusion Matrix Diagram

2.2 Classification Report

Classification reports are a type of performance evaluation report that is commonly used in classification problems. They provide a summary of the model's performance by class, including metrics shown below:

1. **Precision**: the ratio of true positive predictions to the total number of positive predictions (TP / (TP + FP))
2. **Recall**: the ratio of true positive predictions to the total number of actual positives (TP / (TP + FN))
3. **F1-score**: the harmonic mean of precision and recall (2 * (precision * recall) / (precision + recall))
4. **Support**: the number of occurrences of each class in the test set
5. **Accuracy**: the overall proportion of correctly classified samples (TP+TN)/(TP+TN+FP+FN)

2.3 Logistic Regression

Logistic regression is a statistical method used to analyse the relationship between a dependent variable (binary) and one or more independent variables (continuous or categorical). The logistic regression model uses the logistic function, also known as the sigmoid function, to model the probability that an input belonging to a certain class. The governing equation for logistic regression is:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n}}$$

where p is the predicted probability of the binary outcome, β_0 is the intercept term, β_1, β_2, ..., β_n are the coefficients for the independent variables x_1, x_2, ..., x_n.

2.4 Decision Trees

A decision tree is a simple but powerful tool for predictive modelling. It is a tree-like structure that allows you to make decisions or predictions based on a set of conditions. Decision trees divide the dataset into subsets based on the value of the input variables and then apply a decision rule to each subset. Decision trees use a series of if-then statements that describe the decision rules used to classify the data.

2.5 Random Forests

A random forest is an ensemble learning method that combines multiple decision trees to make a prediction. A random forest builds multiple decision trees on different sub-samples of the dataset and aggregates their predictions to get a more accurate and stable prediction. Each decision tree in the forest is trained using a subset of the available features, selected at random, to split the data at each node. The trees in the forest are constructed independently of each other, using different subsets of the training data, and then combined to improve the overall accuracy of the model.

3 Data Processing

This section covers how the data is cleaned to prepare for the analysis. Python will be used for this purpose and Jupyter Notebook will enable us to visualise the data and results in an intuitive way.

3.1 Data Cleaning

The data used for this project is a .csv file, consisting of 7783 rows and 16 columns containing information about assets held by 7783 US commercial banks in their balance sheet in the 4th quarter of 2007. 15 columns refer to features of the dataset, while the final column refers to the binary output variable that we aim to predict, namely '1' which refers to default, or '0' which refers to non-default.

The data is loaded into the notebook as a *Pandas* dataframe, and the column names are added. The dataframe is then checked for null values, and the data is divided into two separate dataframes: *data_features* containing the first 15 columns of the original dataframe, and *data_output* containing the binary output variable.

3.2 Data Correlation

Correlation plots are created to explore the correlation of different data features with each other, and particular with the output which we seek to predict. These plots enable us to see the most important features which will be most important for our prediction models, and what features may be obsolete and could be removed to simplify our models. **Appendix A** shows the correlation of the data features with each other, while **Appendix B** shows the correlation of the data features with the output variable. This will be further explored in section 5 when optimising our final model.

3.3 Data Balancing

The class balance of the original dataset is calculated. The data contains 306 rows of the default class and 7477 rows of the non-default class. To ensure our models are trained on a representative sample of all possible outcomes, the dataset will be balanced. This can be achieved by removing samples from the majority class (undersampling) or adding new samples to the minority class (oversampling) until they have the same number of samples. This can help to prevent bias and overfitting as the model will be optimised to minimise the overall error, rather than prioritising minimising the error for the majority class.

To balance the dataset by undersampling, the *fit_resample* method is used from the *RandomUnderSampler* class in the *imbalanced-learn* library. After undersampling, the dataset contains 306 rows of the default class and 306 rows of the non-default class.

To balance the dataset by oversampling, the *fit_resample* method is used from the *SMOTE* (Synthetic Minority Over-sampling Technique) class in the *imbalanced-learn* library. After oversampling, the dataset contains 7477 rows of the default class, and 7477 rows of the non-default class.

All three datasets (unbalanced, oversampled and undersampled) will be used to compare and test the performance of the 3 supervised learning models. To create the test/train split for the data, the *train_test_split()* function from the *scikit-learn* library is used with a 70/30 split.

4 Model Implementation

All three classification methods are first fitted on the unbalanced training data and a function is defined to fit the test data to the model, predict the test data, and output a confusion matrix.

For each classification method, the process of fitting the model is repeated on the balanced datasets generated by both undersampling and oversampling methods. However, the balanced dataset models will be tested on both the balanced test sets and the unbalanced test set to assess their

performance on the original data.

The result from each model is shown in the appendices, detailing the classification report results and the confusion matrix outputs. The best performing model is highlighted in green, and the model parameters are shown in the header of the table. Default parameters will be used for the initial comparison.

Logistic regression is performed on the dataset using the *LogisticRegression* class from the *scikit-learn* library. The model is trained using the *fit* method. The results are shown in **Appendix C**.

Decision tree classifiers are trained on the dataset using the *DecisionTreeClassifier* class from the *scikit-learn* library. The models are fitted using the fit method. The results are shown in **Appendix D**.

Random forest models are implemented similarly to decision trees, but by using the *RandomForestClassifier* class from the *scikit-learn* library. The models are trained using the *fit* method. The results are shown in **Appendix E**.

4.1 Model Comparison

Based on the performance tables, it is evident that the random forest models exhibit superior performance in both balanced and unbalanced datasets, followed by decision trees and then logistic regression. Balancing the dataset has a positive impact on the performance of all models. However, the oversampling technique proves to be more effective than undersampling in improving performance across all models. Possible reasons for this will be discussed in section 6.

It is important to note that in classification problems where the minority class it much smaller than the majority class, accuracy can be a misleading statistic, as a model could simply classify all data points as negatives. Our main performance metric is the f1-score, particularly of the minority class, which considers both false negatives and false positives. This means that for this problem are weighing false positives and false negatives with equal importance (i.e. classifying default when the bank did not default or classifying non-default when the bank defaulted). Often, comparing the outcomes from confusion matrices offers a more intuitive way to evaluate the performance of classification models.

5 Model Selection

We can now select the oversampled random forest model as the best performing model. This model was able to predict 92/93 defaults from the original test data, with only 34 false positives and 1 false negative from the original test dataset of 2335 entries.

This is in comparison to 72 false positives and 2 false negatives for the decision tree model and 313 false positives and 31 false negatives for the logistic regression model. As such this model will be taken forward for final optimisation, exploring various model improvements.

5.1 Model Optimisation

Standardisation, recursive feature elimination, and parameter optimisation are applied to the model to explore performance improvements.

5.1.1 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a feature selection technique that identifies the most important features in a dataset, improving model performance and reducing overfitting. By iteratively removing features and constructing models with the remaining features, RFE ranks each feature's importance according to the decrease in model performance upon its removal. This technique is implemented using the *RFE* class from the *scikit-learn* library and reducing the number of features from the original 15 and comparing the subsequent change in performance.

5.1.2 Standardisation

Standardisation is a pre-processing technique that transforms the features of a dataset to have zero mean and unit variance. This ensures that all features are on the same scale and aims to improve the convergence rate and performance of algorithms. This technique is implemented on the oversampled data using the *StandardScaler* class from the *scikit-learn* library.

5.1.3 Parameter Optimisation

To improve model performance, we can optimise the input parameters of the Random Forest model. While our initial model used default parameters, we will now explore adjusting the following parameters:

1. *n_estimators*: Number of decision trees in the forest.
2. *max_depth*: Maximum depth of each decision tree.
3. *min_samples_split*: Minimum number of samples required to split an internal node of a decision tree.
4. *min_samples_leaf*: Minimum number of samples required to be at a leaf node of a decision tree.
5. *max_features*: Maximum number of features that can be considered for splitting at each node.

Using the *GridSearchCV* class from the *scikit-learn* library we can search over these parameters to find the optimal values and repeat the model fitting process as before.

5.2 Final Model

In **Appendix F**, you can find the results of the model optimization techniques that were implemented to improve the model's performance. The final version of the model is indicated in yellow.

RFE has a negative effect on the model performance. Despite **Appendix B** showing weak correlation between some of the dataset features and the output variable, the performance is shown to decay exponentially when the number of features is reduced. The results of a reduction to 10 features has been shown in **Appendix F**, showing a slight decrease in all performance metrics. This may be worth the trade-off if the dataset is large and computation times are long, however for this dataset we will not proceed with RFE on the final model.

Feature scaling has a slight negative effect on the model performance. This is partly expected, as the random forest algorithm is robust to different scales and distributions of features, however when testing the model on the original test set (scaled), the performance is very poor. This may have happened because the scaling process can sometimes distort the original distribution of the data, which can lead to overfitting on the oversampled data and worse performance on the original test data. As such, this technique will not be applied to the final model.

The optimal hyperparameters from the GridSearch optimisation are shown below and are applied to the model to output the final confusion matrix:

max_depth: 50, *max_features*: auto, *min_samples_leaf*: 1, *min_samples_split*: 2, *n_estimators*: 200

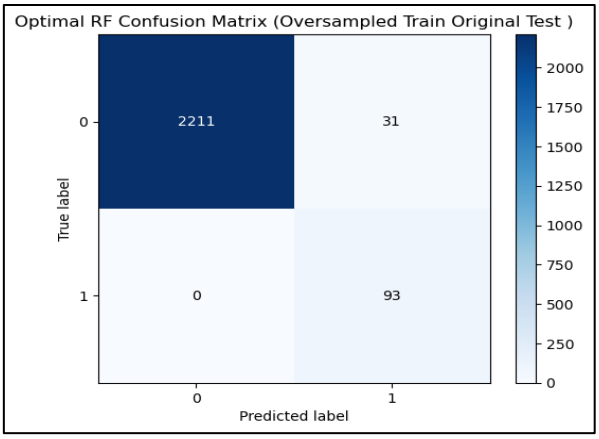


Figure 2 - Final Model Confusion Matrix

The application of optimal hyperparameters to the model results in a slight improvement. The model accurately detected all 93/93 cases of default on the original test data, with 0 false negatives and 31 false positives. This indicates an f1-score of 0.86 on the default class for the original test data. When tested on the oversampled test data, the model predicts 2254/2281 cases of default, resulting in an average f1-score of 0.97, with 27 false negatives and 97 false positives out of 4487 total samples. The improved representation of the minority class and the increase in dataset size is found to enhance the model's ability to learn patterns in the minority class, leading to better performance in predicting default. This marks the final model.

6 Discussion

We have selected the oversampled random forest model as our final model after comparing the three chosen supervised learning methods. In this section, we will explore the possible reasons for its superior performance and discuss other relevant findings from literature.

It is expected that oversampling is the better choice for this problem, due to the significant class imbalance common in default/fraud datasets. Undersampling can result in a loss of information by reducing the size of the available data, leading to bias towards the majority class. On the other hand, oversampling can help to balance the dataset by increasing the size of the minority

class, providing more representative samples, and improving model performance.

There are a number of possible reasons why tree/forest algorithms have performed better than logistic regression. One disadvantage of logistic regression is that it assumes a linear relationship between the predictors and the outcome, which may not be true in our dataset. According to a study by Kim et al. (2015), logistic regression can perform well in predicting bank default when the data is preprocessed and selected carefully, but it may not be as effective as other models when dealing with highly nonlinear relationships between the predictors and the outcome [1]. Decision trees can be prone to overfitting as they have a high variance and can capture noise in the data The paper by Breiman (2001) discusses the use of random forests to address the problem of overfitting in decision trees [3]. Breiman argues that decision trees can be prone to overfitting due to their high variance, which can lead to models that perform well on the training data but poorly on new, unseen data. He claims that random forests can help to reduce the variance of decision trees by creating an ensemble of trees that are trained on different subsets of the data and features. The aggregation of multiple decision trees helps to reduce the impact of individual trees that may have overfit to the training data.

Our research results are consistent with prior studies in the field of bank default classification, where various research works have evaluated the effectiveness of these algorithms comparatively. For example, a study by Liu and Gao in 2015 compared the performance of logistic regression, decision trees, and random forests in predicting bank loan default risk [4]. They found that the random forest model had the best performance and concluded that it may be the most suitable algorithm for bank loan default prediction due to its high predictive power and ability to handle large datasets. Another study by Jha et al. (2021) compared the same algorithms for predicting loan default in Indian banks [5]. They found that random forest again outperformed the other models in terms of accuracy, while logistic regression had the lowest performance.

QUESTION 2: Equity Return Clustering

1 Introduction

The objective of this task is to perform a clustering analysis on daily equity return data, with the goal of finding interpretations of the clusters and whether certain industries tend to cluster together.

We will first analyse two common clustering methods using Python to find the best performing model. We will then explore any interesting findings.

To conclude we will explore the relationship between the time period and the clustering results. Specifically, we will explore if there exits patterns and differences between bull and bear market years.

2 Methodology

In this study, we will compare two clustering methods, namely **K-Means Clustering and Hierarchical Clustering**, to determine their effectiveness in clustering daily equity return data. Both methods will cluster the data based on their (standardised) mean return and mean standard deviation. We will then investigate the optimal number of clusters and evaluate the performance of each algorithm to determine which one performs best.

We will then use the best performing algorithm to compare bull and bear market years, to identify patterns in the industries which tend to cluster together given different economic landscapes. Importing NYSE composite index data using the Yahoo Finance API will allow us to calculate the annual return of the index for each year and identify bull and bear market years for comparison.

2.1 Elbow Method

The elbow method is a heuristic approach used to determine the optimal number of clusters in clustering algorithms. It involves plotting the within-cluster sum of squared distances (or other evaluation metrics) against the number of clusters. The optimal number of clusters is typically found at the "elbow" point of the curve, where adding more clusters results in a marginal improvement in the evaluation metric. This point represents a balance between minimizing the within-cluster

variance while avoiding overfitting by using too many clusters.

2.2 Introduction to Clustering Methods

Clustering methods are a set of unsupervised machine learning techniques that are used to group similar objects or data points together based on their features or characteristics. The aim of clustering is to divide a dataset into groups or clusters such that the data points within each cluster are more similar to each other than they are to data points in other clusters.

2.2.1 K-Means Clustering

K-Means Clustering partitions data points into a predefined number of clusters based on their similarity. The algorithm operates by iteratively optimizing the position of the cluster centres, known as centroids, to minimize the sum of the squared distances between each data point and its assigned centroid. Specifically, the algorithm involves the following steps: (1) randomly initializing K centroids, (2) assigning each data point to the nearest centroid, (3) recalculating the position of the centroids based on the mean of the data points assigned to them, and (4) repeating steps (2) and (3) until convergence, i.e., when the centroids no longer move.

2.2.2 Hierarchical Clustering

Hierarchical Clustering groups data points into clusters based on their similarity, creating a dendrogram to visualize the clustering hierarchy. It does not require a pre-specified number of clusters and has two main types: agglomerative and divisive. Agglomerative clustering merges similar clusters, starting with each data point as its own cluster until all belong to a single cluster. Divisive clustering starts with all data points in one cluster and recursively splits them into smaller, more homogeneous clusters. The similarity between data points is measured using a distance metric, while the linkage criterion determines how distances between clusters are computed.

3 Data Cleaning

The dataset is imported into the Jupyter Notebook as a .csv file. The data is stored in a *Pandas* dataframe, and the first 24100 rows are selected, which contain the market cap weighted equity return values for different industries. This format of data is chosen over equally weighted returns as it will provide a more accurate representation of the industry return when comparing different years of index returns.

The appropriate column names are added to the dataframe and the rows that contain NaN values are removed. This still leaves us with 12196 rows of complete data, roughly equivalent to 48 years of trading days. Finally, the row indexes are converted to the datetime format which will allow us to easily select different time periods.

3.1 Data Standardisation

Data standardisation in clustering methods refers to the process of transforming the features or variables of a dataset to a common scale or distribution. This is done to ensure that all features have equal importance when applying clustering algorithms, which typically rely on distance-based similarity measures, and can improve algorithm performance and comparison.

First, we pick a subset of the data (year 2000) to allow us to compare the clustering methods. We then calculate the mean return and standard deviation for that time period and save the data to a new dataframe. The data can now be standarised using the *StandardScaler* class from the *scikit-learn* library. The *fit_transform* method allows us to remove the mean and scale the data to unit variance.

4 Clustering Implementation

This section covers the implementation of the two previously introduced clustering algorithms for comparison: K-Means Clustering and Hierarchical Clustering.

4.1 Number of Clusters

To determine the optimal number of clusters, we will use the elbow method on both models as defined on section 2.1. The code calculates the within-cluster sum of squares (WCSS) for different numbers of clusters (k) ranging from 1 to 10. The WCSS values are stored in a list and then plotted against the number of clusters to visualize the elbow curve. In addition, we can use the *KneeLocator* module from the *kneed* library to automatically identify the knee point in the curve, which represents the optimal number of clusters.

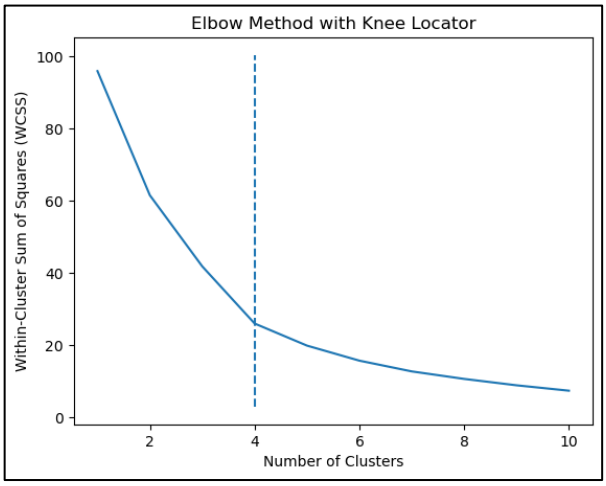


Figure 3 - Elbow method to determine optimal number of clusters for the K-Means algorithm

4.2 Model Fitting

The K-Means Clustering method is implemented using the *KMeans* class from the *scikit-learn* library. The *fit* method is applied to the standardised data. The plot is shown in **Appendix G**.

The Hierarchical Clustering method is implemented using the *AgglomerativeClustering* class from the *scikit-learn* library. The fit method is also applied to the standardised data and the plot is shown in **Appendix H**.

4.3 Performance Comparison

To measure the performance of both methods we can use the silhouette score and the Calinski-Harabasz score. The silhouette score measures the similarity of an object within its own cluster compared to other clusters, with values ranging from -1 to 1. A higher silhouette score indicates well-separated clusters. This is implemented using the *silhouette_score* function from the *scikit-learn* library. The Calinski-Harabasz score is a ratio of between-cluster dispersion to within-cluster dispersion, with higher values also indicating better clustering performance and well-defined clusters. This is implemented using the *calinski_harabasz_score* function from the *scikit-learn* library.

Both methods returned identical scores for the two performance metrics, with silhouette score of 0.39 and a Calinski Harabasz score of ~37. This is because both methods identically clustered the stock industries when the same number of clusters was applied.

5 Clustering Market Conditions

This section will explore whether certain years of display correlation when clustering analysis is performed. We will focus on identifying any potential relationship between market index performance and industry clustering characteristics by comparing years during bull and bear markets.

To compare clustering plots, we can use the Adjusted Rand Index (ARI); a measure used to evaluate the similarity between two clustering results, while considering the impact of random chance. It ranges from -1 to 1, where a higher score indicates more similar clusters. This statistic will be calculated using the *adjusted_rand_score* function from the *scikit-learn* library and passing the cluster labels from two different years.

5.1 Market Performance Data

First, we need to acquire the market returns. To achieve this, we will use the NYSE Composite Index as our benchmark and import the relevant stock data from the Yahoo Finance API.

This data is imported into a *Pandas* dataframe, and the monthly and annual returns are calculated from the daily close prices. The tail of the dataframe is printed to view the return of the index and identify bull and bear market years that can be used for our comparative analysis.

5.2 Bull Market Comparison

To investigate if specific industries tend to cluster together during bull markets, we will analyse the years 1995 and 1997, during which the NYSE Composite Index returned 31.3% and 30.3%, respectively. We will follow the same data standardisation process outlined in section 3.1 and calculate the mean return and standard deviation for each year. After applying a K-Means clustering model to the data, the clustering plots for both years are displayed in **Appendix I**. Accompanying tables show the industries present in each cluster, highlighting those that appear in the same cluster across both years.

The Adjusted Rand Index for this comparison is 0.247, suggesting there is some degree of similarity between the two plots. Notably, both the **Gold** (precious metals) and **Coal** industries showed low returns yet high volatility.

Precious metal stocks often underperform in bull markets as investors shift their focus to riskier, high-growth assets. Coal stocks' performance is more typically related to other factors like energy demand, regulatory changes, and global economic conditions, which may lead to lower returns in bull markets compared to other industries that are more directly linked to economic growth.

5.3 Bear Market Comparison

For a comparison of bear market years, we will examine the years 2001 and 2002, when the index experienced returns of -10.2% and -19.8%, respectively. We will calculate each year's mean return and standard deviation and use these values to generate clustering plots. The resulting plots for both years can be found in **Appendix J**, along with tables detailing the industries present in each cluster. Notably, the tables highlight industries that consistently cluster together across both years.

The Adjusted Rand Index for this comparison is 0.241, also suggesting there is some positive correlation between the two plots. Industries which displayed high volatility in both bear market years included **Chips** (electronic equipment) and **Comps** (computers), while industries which had high returns included **Guns** (defence) and **Gold**.

Technology stocks are often more volatile during bear markets due to higher valuations, growth sensitivity, and investor risk aversion. In contrast, precious metal stocks, particularly gold-related stocks, tend to perform better during bear markets as they serve as safe-haven assets, inversely correlate with equities, and act as a hedge against inflation.

5.4 Bull vs Bear Market Comparison

Lastly, we will compare bull market years with bear market years to determine if any correlation exists. We will first compare 1997 (bull) vs 2001 (bear) and then 1995 (bull) vs 2002 (bear). Since both pairs of years have already been used in sections 5.1 and 5.2, no additional data processing is necessary. The first clustering comparison can be found in **Appendix K**, and the second in **Appendix L**, each displaying the industries in each cluster and those common to the same cluster across both plots.

The Adjusted Rand Index values for both analyses are 0.08 and 0.02, respectively. These low values indicate that the correlation between the clustering plots is minimal.

6 Discussion

From the cluster plots we can see that when using clustering model with 3 clusters, the industries tend to be clustered into the following categories:

1. High volatility and low return
2. Low volatility and low return
3. Average volatility and high return

Similar industries often tend to cluster together, for example technology industries (**Comps**, **Chips**), commodity industries (**Gold**, **Coal**) and consumer staple industries (**Food**, **Hshld**), however their placement into one of the categories above appears to show correlation with the wider market conditions.

Both the bull market comparison and the bear market comparison produced a reasonable positive ARI, showing similarity between the compared clustering plots. This suggests that there exist common patterns between which industries tend to cluster together during particular market conditions. The low ARI values when comparing bull market years against bear market years is an attempt to strengthen this hypothesis.

A key limitation of this analysis is the small sample size, which may lead to findings influenced by random chance or specific events during those years. To further validate the hypothesis, future work could expand the sample size and apply statistical tests, such as a permutation test, to assess the significance of clustering results by comparing observed ARI scores to a null distribution.

Our findings align with existing research. Füss, Miebs, and Trübenbach (2013) [5] investigated the role of economic, financial, and political risk factors in the clustering of industries in stock returns, using data from 48 industries across 23 developed countries between 1973 and 2008. They found that industries with similar risk exposures tend to cluster together in both bull and bear markets, but the composition of these clusters might change depending on market conditions.

References

[1] - Kim, J. H., Kim, J. Y., & Kim, J. H. (2015). Comparison of logistic regression and decision tree for prediction of bank loan default risk. Expert Systems with Applications, 42(6), 3123-3130. doi:10.1016/j.eswa.2014.11.065

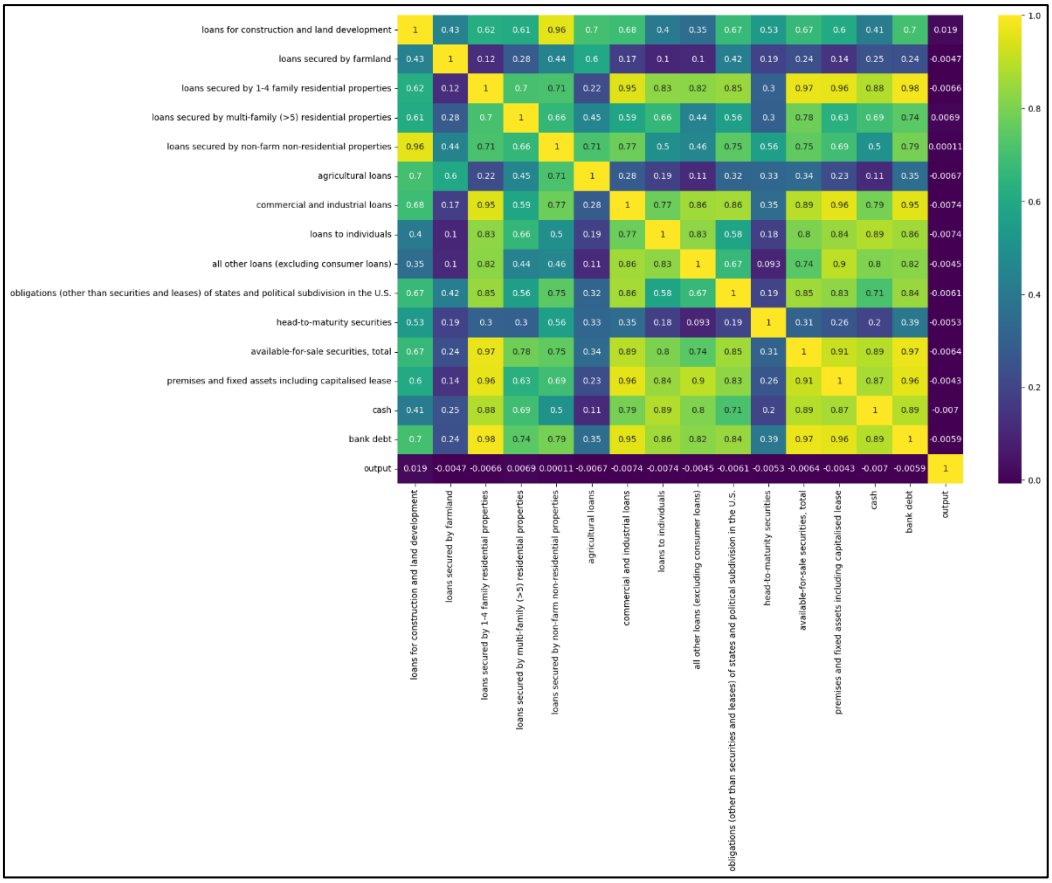
[2] - Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. doi.org/10.1023/A:1010933404324

[3] - Liu, T., & Gao, Y. (2018). A Comparison of Logistic Regression, Decision Trees, and Random Forests in the Prediction of Bank Loan Default Risk. Journal of Business Research, 88, 99-107. doi:10.1016/j.jbusres.2018.01.044

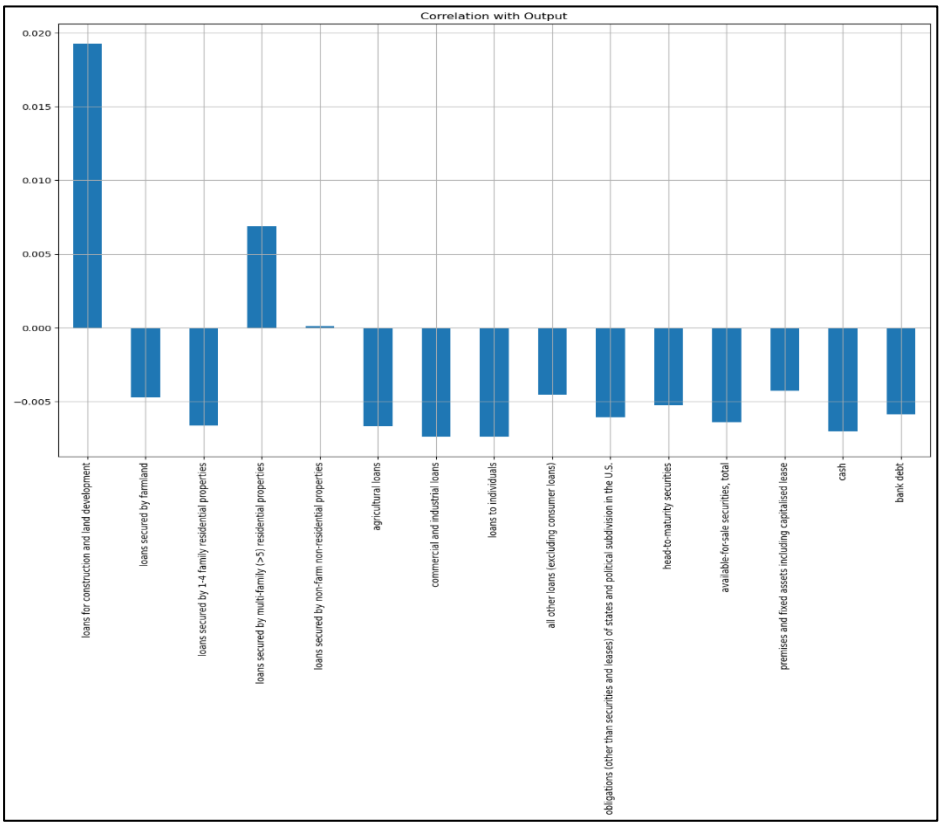
[4] - Jha, R. K., Bajpai, V., & Bhattacharya, A. (2021). Predicting loan default in Indian banks using logistic regression, decision tree, and random forest models. Journal of Risk and Financial Management, 14(2), 50.

[5] - Füss, R., Miebs, F., & Trübenbach, F. (2013). The clustering of industries in stock returns: The role of economic, financial, and political risk factors. Journal of Banking & Finance, 37(10), 4016-4034.

Appendix A: Correlation Plot with Data Features



Appendix B: Correlation Plot with Output



Appendix C: Logistic Regression Model Performance

Model: class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)									
		Precision	**Recall**	**F1-score**	**Support**	**TN**	**FN**	**FP**	**TP**
Unbalanced	**0**	0.97	0.98	0.97	2242	2194	68	48	25
	1	0.34	0.27	0.30	93				
	Accuracy	-	-	0.95	2335				
	Macro. Avg.	0.66	0.62	0.64	2335				
	Weighted Avg.	0.94	0.95	0.95	2335				
Undersampled (undersampled test data)	**0**	0.80	0.80	0.8	104	83	21	21	59
	1	0.74	0.74	0.74	80				
	Accuracy	-	-	0.77	184				
	Macro. Avg.	0.77	0.77	0.77	184				
	Weighted Avg.	0.77	0.77	0.77	184				
Undersampled (original test data)	**0**	0.99	0.83	0.9	2242	1870	26	372	67
	1	0.15	0.72	0.25	93				
	Accuracy	-	-	0.83	2335				
	Macro. Avg.	0.57	0.78	0.58	2335				
	Weighted Avg.	0.95	0.83	0.88	2335				
Oversampled (oversampled test data)	**0**	0.75	0.87	0.8	2206	1914	644	292	1637
	1	0.85	0.72	0.78	2281				
	Accuracy	-	-	0.79	4487				
	Macro. Avg.	0.8	0.79	0.79	4487				
	Weighted Avg.	0.8	0.79	0.79	4487				
Oversampled (original test data)	**0**	0.98	0.87	0.92	2242	1950	31	292	62
	1	0.18	0.67	0.28	93				
	Accuracy	-	-	0.86	2335				
	Macro. Avg.	0.58	0.77	0.60	2335				
	Weighted Avg.	0.95	0.86	0.90	2335				

Appendix D: Decision Tree Model Performance

Model: class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0, class_weight=None, ccp_alpha=0)									
		Precision	**Recall**	**F1-score**	**Support**	**TN**	**FN**	**FP**	**TP**
Unbalanced	**0**	0.96	0.96	0.96	2242	2142	79	100	14
	1	0.12	0.15	0.14	93				
	Accuracy	-	-	0.92	2335				
	Macro. Avg.	0.54	0.55	0.55	2335				
	Weighted Avg.	0.93	0.92	0.93	2335				
Undersampled (undersampled test data)	**0**	0.78	0.65	0.71	104	68	19	36	61
	1	0.63	0.76	0.69	80				
	Accuracy	-	-	0.7	184				
	Macro. Avg.	0.71	0.71	0.7	184				
	Weighted Avg.	0.72	0.70	0.7	184				
Undersampled (original test data)	**0**	1.00	0.69	0.81	2242	1538	6	704	87
	1	0.11	0.94	0.20	93				
	Accuracy	-	-	0.7	2335				
	Macro. Avg.	0.55	0.81	0.50	2335				
	Weighted Avg.	0.96	0.7	0.79	2335				
Oversampled (oversampled test data)	**0**	0.94	0.90	0.92	2206	1989	133	217	2148
	1	0.91	0.94	0.92	2281				
	Accuracy	-	-	0.92	4487				
	Macro. Avg.	0.92	0.92	0.92	4487				
	Weighted Avg.	0.92	0.92	0.92	4487				
Oversampled (original test data)	**0**	1.00	0.97	0.99	2242	2180	4	62	89
	1	0.59	0.96	0.73	93				
	Accuracy	-	-	0.97	2335				
	Macro. Avg.	0.79	0.96	0.86	2335				
	Weighted Avg.	0.98	0.97	0.97	2335				

Appendix E: Random Forest Model Performance

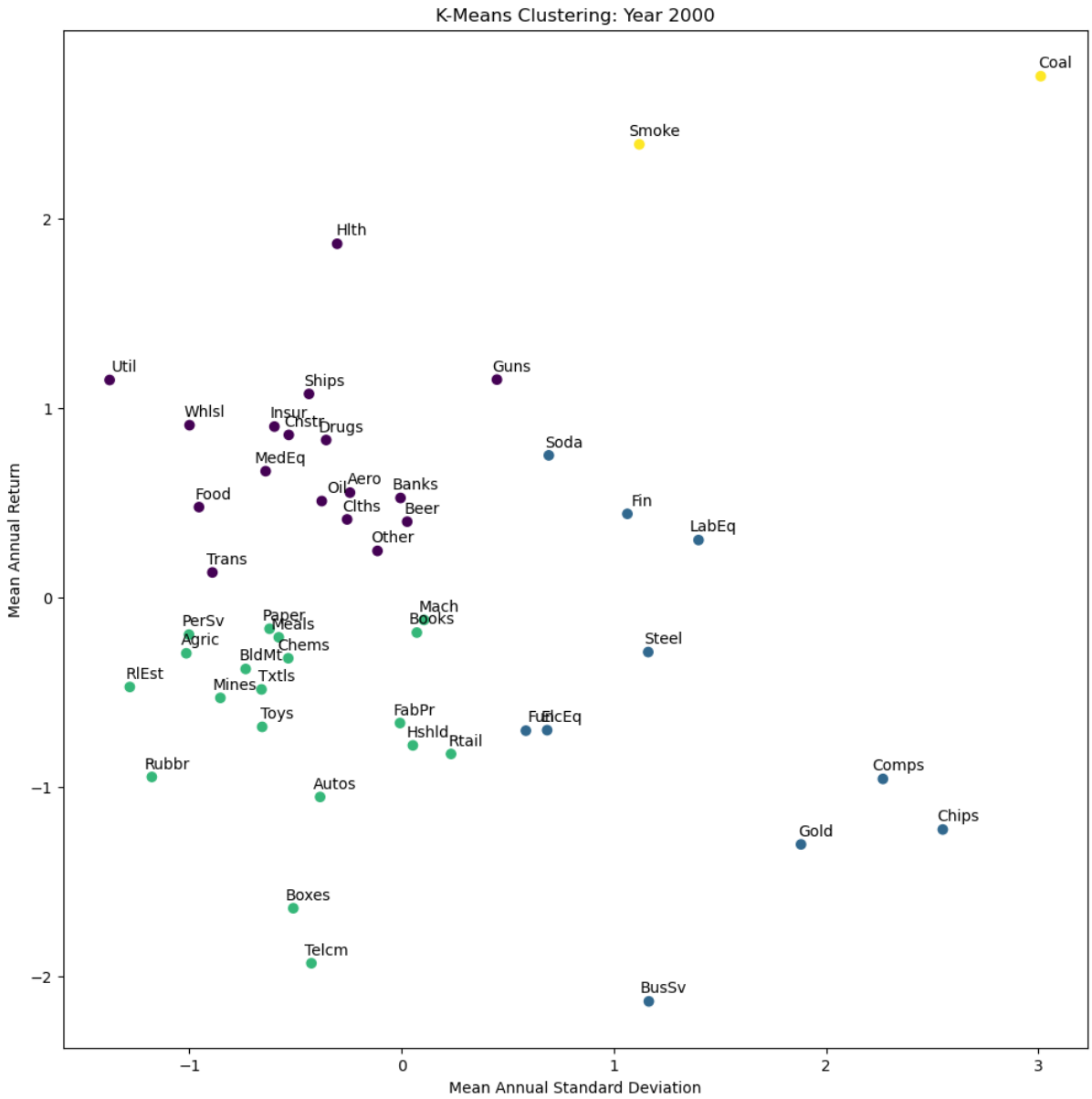
Model:class sklearn.ensemble.RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0, max_samples=None)									
		Precision	**Recall**	**F1-score**	**Support**	**TN**	**FN**	**FP**	**TP**
Unbalanced	**0**	0.96	1.00	0.98	2242	2239	90	3	3
	1	0.50	0.03	0.06	93				
	Accuracy	-	-	0.96	2335				
	Macro. Avg.	0.73	0.52	0.52	2335				
	Weighted Avg.	0.94	0.96	0.94	2335				
Undersampled (undersampled test data)	**0**	0.87	0.73	0.80	104	76	11	28	69
	1	0.71	0.86	0.78	80				
	Accuracy	-	-	0.79	184				
	Macro. Avg.	0.79	0.80	0.79	184				
	Weighted Avg.	0.80	0.79	0.79	184				
Undersampled (original test data)	**0**	1.00	0.74	0.85	2242	1655	5	587	88
	1	0.13	0.95	0.23	93				
	Accuracy	-	-	0.75	2335				
	Macro. Avg.	0.56	0.84	0.54	2335				
	Weighted Avg.	0.96	0.75	0.82	2335				
Oversampled (oversampled test data)	**0**	0.99	0.95	0.97	2206	2106	27	100	2254
	1	0.96	0.99	0.97	2281				
	Accuracy	-	-	0.97	4487				
	Macro. Avg.	0.97	0.97	0.97	4487				
	Weighted Avg.	0.97	0.97	0.97	4487				
Oversampled (original test data)	**0**	1.00	0.99	0.99	2242	2209	0	33	93
	1	0.74	1.00	0.85	93				
	Accuracy	-	-	0.99	2335				
	Macro. Avg.	0.87	0.99	0.92	2335				
	Weighted Avg.	0/99	0.99	0.99	2335				

Appendix F: Oversampled Random Forest Model Improvement Performance

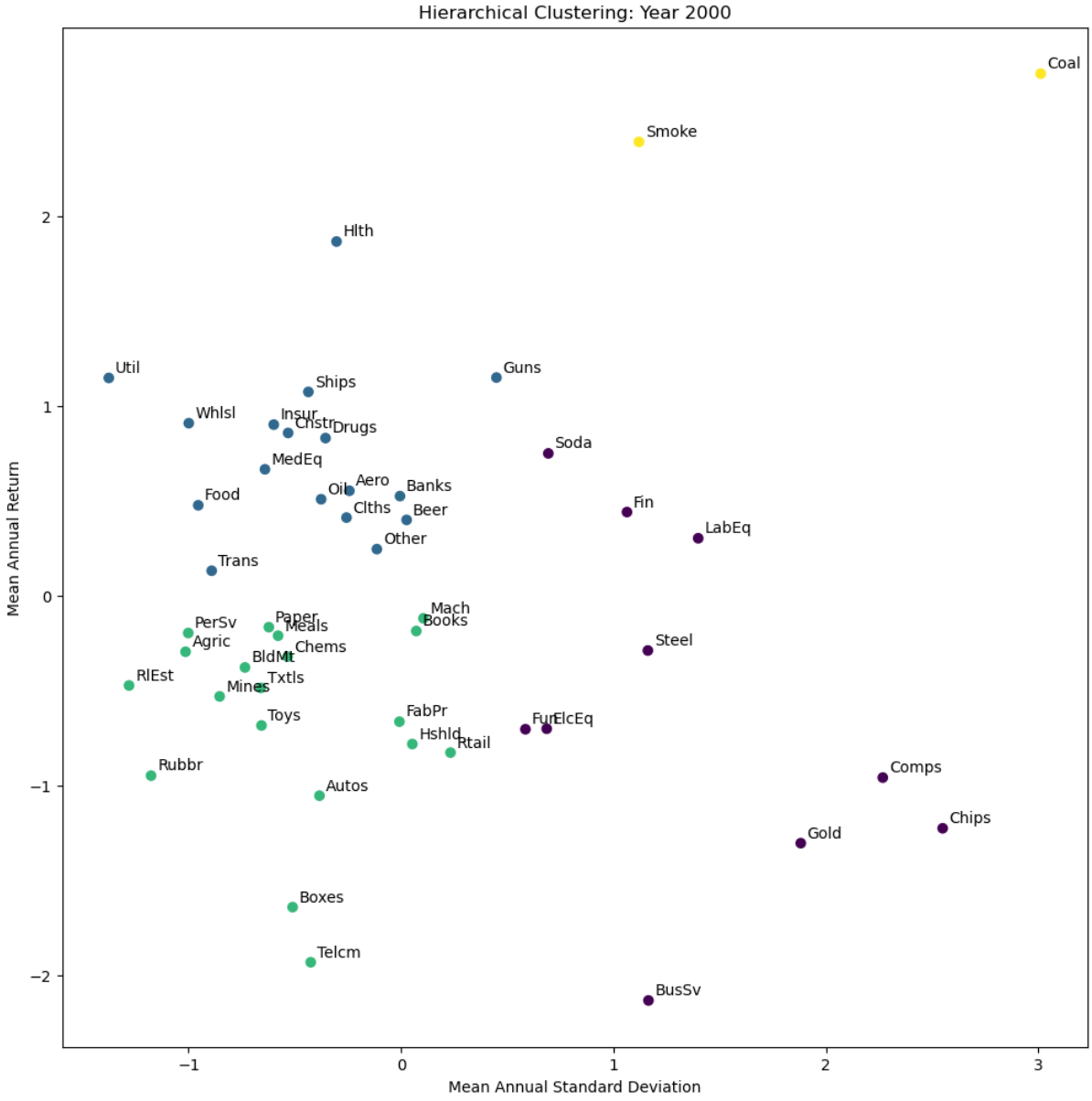
Model: class sklearn.ensemble.RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=50, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0, max_samples=None)

		Precision	Recall	F1-score	Support	TN	FN	FP	TP
Standardisation (oversampled test data)	**0**	0.98	0.95	0.97	2239	2137	39	102	2209
	1	0.96	0.98	0.97	2248				
	Accuracy	-	-	0.97	4487				
	Macro. Avg.	0.97	0.97	0.97	4487				
	Weighted Avg.	0.97	0.97	0.97	4487				
Standardisation (original test data)	**0**	0.96	0.95	0.95	2242	2122	86	120	7
	1	0.06	0.08	0.06	93				
	Accuracy	-	-	0.91	2335				
	Macro. Avg.	0.51	0.51	0.51	2335				
	Weighted Avg.	0.92	0.91	0.92	2335				
Recursive Feature Selection (10 features) (oversampled test data)	**0**	0.98	0.95	0.97	2206	2088	33	118	2248
	1	0.95	0.99	0.97	2281				
	Accuracy	-	-	0.97	4487				
	Macro. Avg.	0.97	0.97	0.97	4487				
	Weighted Avg.	0.97	0.97	0.97	4487				
Recursive Feature Selection (10 features) (original test data)	**0**	1.00	0.98	0.99	2242	2199	1	43	92
	1	0.68	0.99	0.81	93				
	Accuracy	-	-	0.98	2335				
	Macro. Avg.	0.84	0.99	0.90	2335				
	Weighted Avg.	0.99	0.98	0.98	2335				
Optimal Hyperparameters (oversampled test data)	**0**	0.99	0.96	0.97	2206	2109	27	97	2254
	1	0.96	0.99	0.97	2281				
	Accuracy	-	-	0.97	4487				
	Macro. Avg.	0.97	0.97	0.97	4487				
	Weighted Avg.	0.97	0.97	0.97	4487				
Optimal Hyperparameters (original test data)	**0**	1.00	0.99	0.99	2242	2211	0	31	93
	1	0.75	1.00	0.86	93				
	Accuracy	-	-	0.99	2335				
	Macro. Avg.	0.88	0.99	0.93	2335				
	Weighted Avg.	0.99	0.99	0.99	2335				

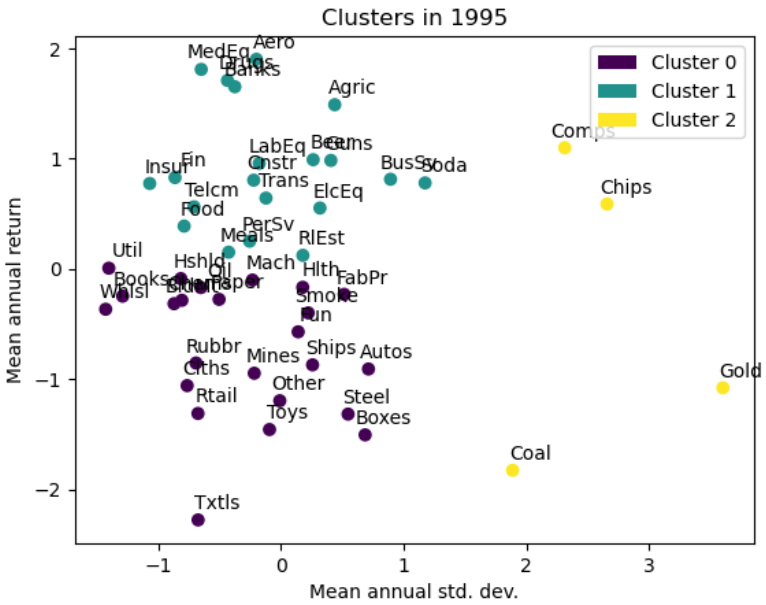
Appendix G: K-Means Clustering Analysis (Year 2000)

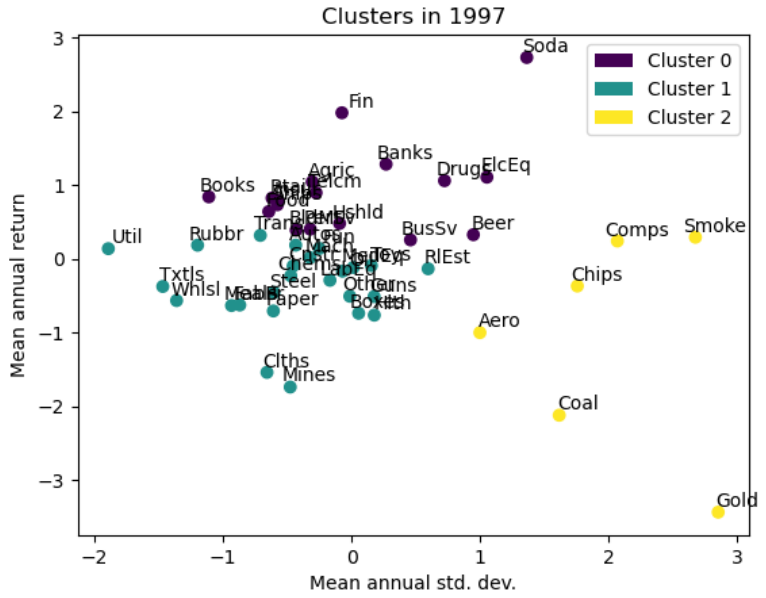


Appendix H: Hierarchical Clustering (Year 2000)



Appendix I: Bull Market Clustering Plots and Cluster Comparison (1995 vs 1997)



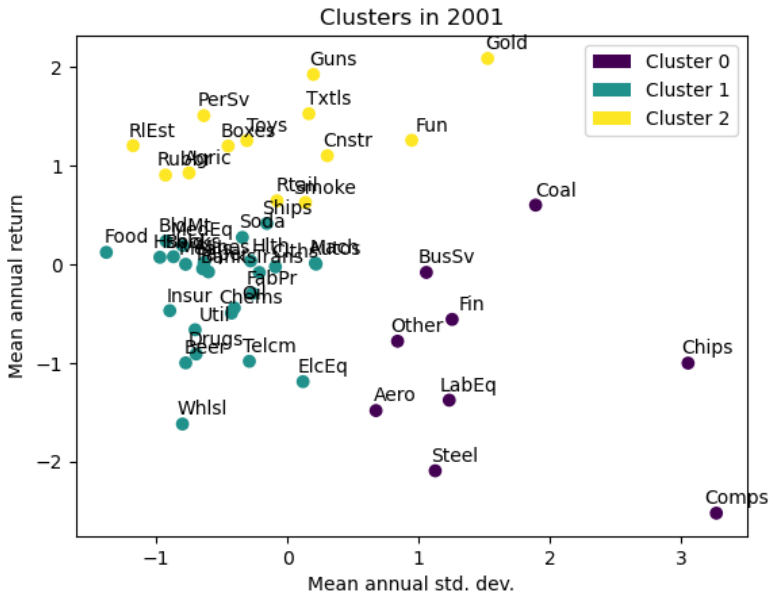


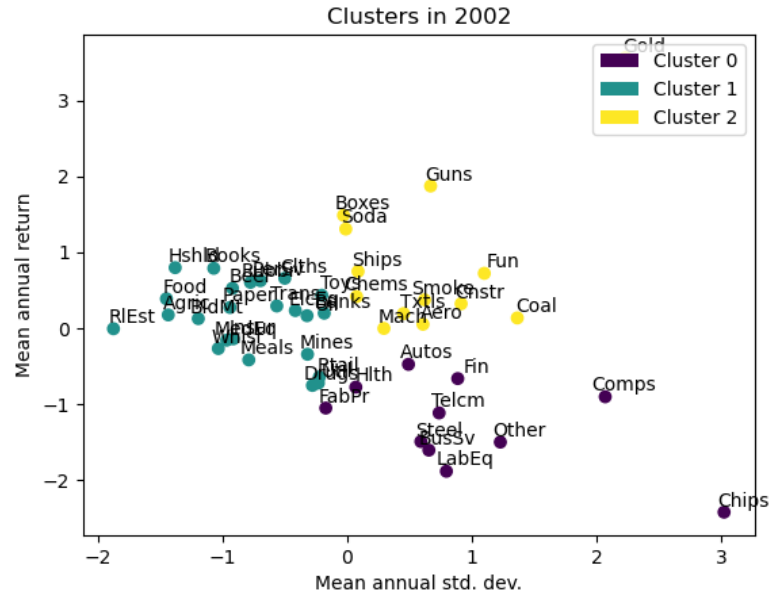

```
Industries in clusters for 1995:
Cluster 0: Smoke, Toys, Fun, Books, Hshld, Clths, Hlth, Chems, Rubbr, Txtls, BldMt, Steel, FabPr, Mach, Autos, Ships, Mines, Oi
l, Util, Paper, Boxes, Whlsl, Rtail, Other
Cluster 1: Agric, Food, Soda, Beer, MedEq, Drugs, Cnstr, ElcEq, Aero, Guns, Telcm, PerSv, BusSv, LabEq, Trans, Meals, Banks, In
sur, RlEst, Fin
Cluster 2: Gold, Coal, Comps, Chips

Industries in clusters for 1997:
Cluster 0: Agric, Food, Soda, Beer, Books, Hshld, Drugs, BldMt, ElcEq, Ships, Telcm, PerSv, BusSv, Rtail, Banks, Insur, Fin
Cluster 1: Toys, Fun, Clths, Hlth, MedEq, Chems, Rubbr, Txtls, Cnstr, Steel, FabPr, Mach, Autos, Guns, Mines, Oil, Util, LabEq,
Paper, Boxes, Trans, Whlsl, Meals, RlEst, Other
Cluster 2: Smoke, Aero, Gold, Coal, Comps, Chips

Duplicate industries in the same clusters:
Cluster 0 in both years share industries: BldMt, Hshld, Ships, Books, Rtail
Cluster 1 in both years share industries: LabEq, RlEst, Cnstr, Trans, MedEq, Guns, Meals
Cluster 2 in both years share industries: Chips, Coal, Comps, Gold
```

Appendix J: Bear Market Clustering Plots and Cluster Comparison (2001 vs 2002)



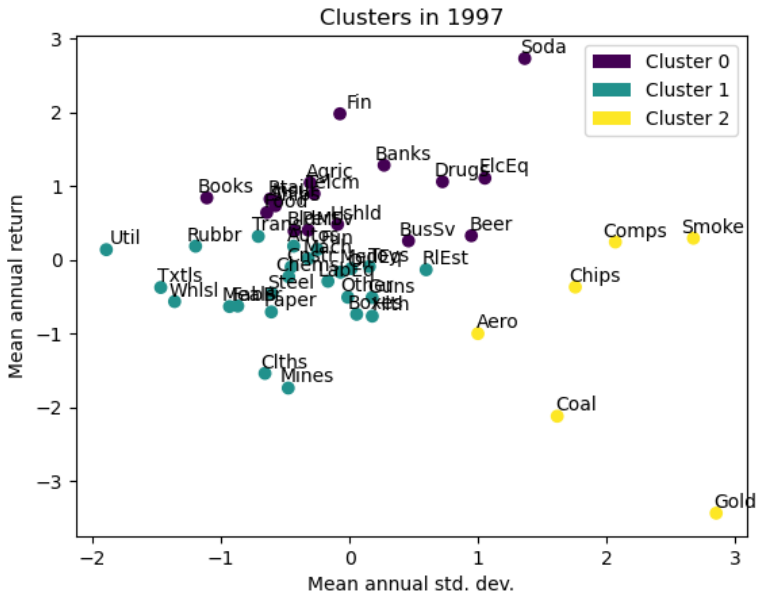


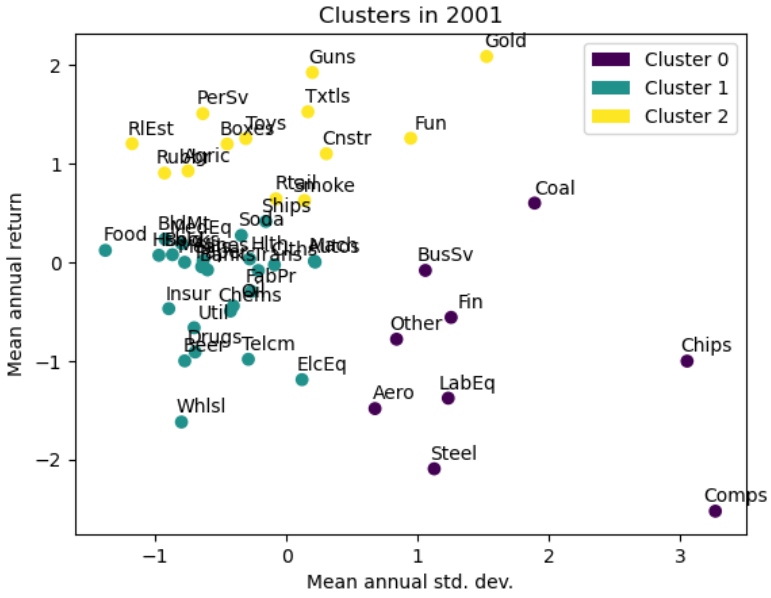

```
Industries in clusters for 2001:
Cluster 0: Steel, Aero, Coal, BusSv, Comps, Chips, LabEq, Fin, Other
Cluster 1: Food, Soda, Beer, Books, Hshld, Clths, Hlth, MedEq, Drugs, Chems, BldMt, FabPr, Mach, ElcEq, Autos, Ships, Mines, Oi
l, Util, Telcm, Paper, Trans, Whlsl, Meals, Banks, Insur
Cluster 2: Agric, Smoke, Toys, Fun, Rubbr, Txtls, Cnstr, Guns, Gold, PerSv, Boxes, Rtail, RlEst

Industries in clusters for 2002:
Cluster 0: Hlth, Steel, FabPr, Autos, Telcm, BusSv, Comps, Chips, LabEq, Fin, Other
Cluster 1: Agric, Food, Beer, Toys, Books, Hshld, Clths, MedEq, Drugs, Rubbr, BldMt, ElcEq, Mines, Oil, Util, PerSv, Paper, Tra
ns, Whlsl, Rtail, Meals, Banks, Insur, RlEst
Cluster 2: Soda, Smoke, Fun, Chems, Txtls, Cnstr, Mach, Aero, Ships, Guns, Gold, Coal, Boxes

Duplicate industries in the same clusters:
Cluster 0 in both years share industries: Fin, Chips, LabEq, Other, BusSv, Steel, Comps
Cluster 1 in both years share industries: Drugs, BldMt, Mines, Hshld, ElcEq, Beer, Clths, Food, Banks, Books, Trans, Oil, Insu
r, MedEq, Util, Paper, Meals, Whlsl
Cluster 2 in both years share industries: Smoke, Boxes, Fun, Cnstr, Txtls, Gold, Guns
```

Appendix K: Bull vs Bear Market Clustering Plots and Cluster Comparison (1997 vs 2001)



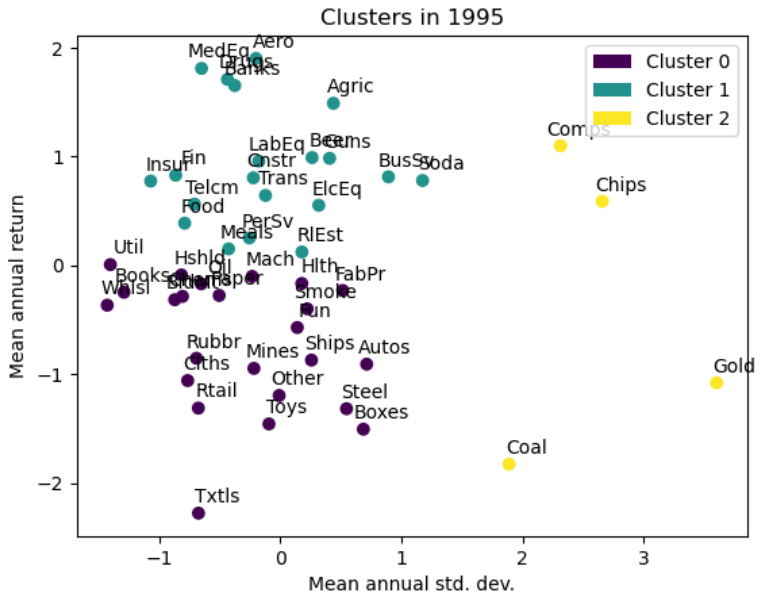


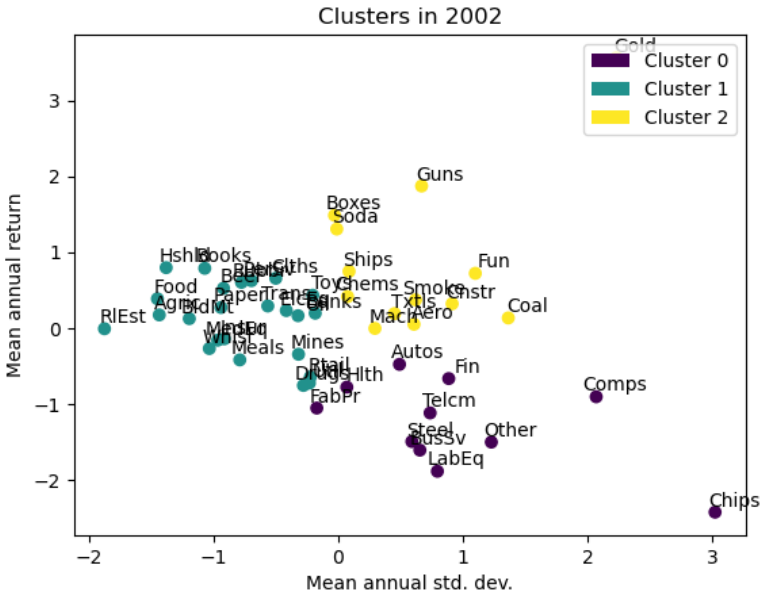

```
Industries in clusters for 1997:
Cluster 0: Agric, Food, Soda, Beer, Books, Hshld, Drugs, BldMt, ElcEq, Ships, Telcm, PerSv, BusSv, Rtail, Banks, Insur, Fin
Cluster 1: Toys, Fun, Clths, Hlth, MedEq, Chems, Rubbr, Txtls, Cnstr, Steel, FabPr, Mach, Autos, Guns, Mines, Oil, Util, LabEq,
Paper, Boxes, Trans, Whlsl, Meals, RlEst, Other
Cluster 2: Smoke, Aero, Gold, Coal, Comps, Chips

Industries in clusters for 2001:
Cluster 0: Steel, Aero, Coal, BusSv, Comps, Chips, LabEq, Fin, Other
Cluster 1: Food, Soda, Beer, Books, Hshld, Clths, Hlth, MedEq, Drugs, Chems, BldMt, FabPr, Mach, ElcEq, Autos, Ships, Mines, Oi
l, Util, Telcm, Paper, Trans, Whlsl, Meals, Banks, Insur
Cluster 2: Agric, Smoke, Toys, Fun, Rubbr, Txtls, Cnstr, Guns, Gold, PerSv, Boxes, Rtail, RlEst

Duplicate industries in the same clusters:
Cluster 0 in both years share industries: Fin, BusSv
Cluster 1 in both years share industries: Mines, Util, Chems, Clths, Trans, Autos, Oil, Mach, FabPr, Hlth, MedEq, Paper, Meals,
Whlsl
Cluster 2 in both years share industries: Smoke, Gold
```

Appendix L: Bull vs Bear Market Clustering Plots and Cluster Comparison (1995 vs 2002)






```
Industries in clusters for 1995:
Cluster 0: Smoke, Toys, Fun, Books, Hshld, Clths, Hlth, Chems, Rubbr, Txtls, BldMt, Steel, FabPr, Mach, Autos, Ships, Mines, Oi
l, Util, Paper, Boxes, Whlsl, Rtail, Other
Cluster 1: Agric, Food, Soda, Beer, MedEq, Drugs, Cnstr, ElcEq, Aero, Guns, Telcm, PerSv, BusSv, LabEq, Trans, Meals, Banks, In
sur, RlEst, Fin
Cluster 2: Gold, Coal, Comps, Chips

Industries in clusters for 2002:
Cluster 0: Hlth, Steel, FabPr, Autos, Telcm, BusSv, Comps, Chips, LabEq, Fin, Other
Cluster 1: Agric, Food, Beer, Toys, Books, Hshld, Clths, MedEq, Drugs, Rubbr, BldMt, ElcEq, Mines, Oil, Util, PerSv, Paper, Tra
ns, Whlsl, Rtail, Meals, Banks, Insur, RlEst
Cluster 2: Soda, Smoke, Fun, Chems, Txtls, Cnstr, Mach, Aero, Ships, Guns, Gold, Coal, Boxes

Duplicate industries in the same clusters:
Cluster 0 in both years share industries: Other, Autos, Hlth, FabPr, Steel
Cluster 1 in both years share industries: Drugs, ElcEq, RlEst, Beer, Food, Banks, Trans, Insur, PerSv, Agric, MedEq, Meals
Cluster 2 in both years share industries: Coal, Gold
```