Enhancing Analysis of Hardware Design Verification Metrics Using Machine Learning & Data Visualization

OSCAR UGGLA
AXEL VOSS

MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Enhancing Analysis of Hardware Design Verification Metrics Using Machine Learning & Data Visualization

Oscar Uggla
`elt15oug@student.lth.se`
Axel Voss
`elt15avo@student.lth.se`

Department of Electrical and Information Technology
Lund University

Supervisor: Erik Larsson

Examiner: Pietro Andreani

May 26, 2021

© 2021
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Closing coverage holes during verification of digital integrated circuits is an iterative process to guarantee all parts are verified before fabrication. The time and manpower it takes to close coverage holes extend with the increasing complexity and size of a digital circuit.

In this thesis unsupervised machine learning is used to cluster together related coverage holes, providing aid for a verification engineer to see a connection between uncovered coverage holes present in a digital RTL design. The coverage metrics are first extracted from a coverage database along with information of the RTL, the data is then pre-processed to prepare it for clustering. The coverage holes are clustered together with the OPTICS, affinity propagation and k-means algorithms.

Observation shows that clustering together the coverage holes can reveal which holes are related to each other and that a potential solution for one of them have a high chance at also fixing the related holes.

This can be used by verification engineers to optimize the task of closing coverage holes. The resources spent on verification can then be reduced to save cost on development.

Populärvetenskaplig Sammanfattning

Integrerade kretsar är en central del i elektronikutrustning som mobiltelefoner, datorer, kameror och andra elektroniska apparater vi använder dagligen. Det har blivit en självklarhet att ha någon form av integrerade kretsar när du producerar modern elektronik. Med hjälp av integrerade kretsar så kan du reducera kostnad, storlek och hastighet jämfört med enskilda komponenter på ett kretskort. Under åren så har antalet transistorer som bygger upp en integrerad krets ökat markant, en krets kan idag bestå av flera miljoner olika transistorer som tillsammans bygger upp kretsens funktion. Förr i tiden så var integrerade kretsar små sett till antalet transistorer, du kunde då skapa en ny krets och började om från början när nästa version skulle ges ut. Nu har kretsarna blivit så avancerade att det inte längre är lönsamt att börja om från början när du designar en ny krets. Istället återanvänder du och delar upp dina integrerade kretsar i så kallade ip-block. När en ny version av en integrerad krets ska utvecklas så återanvänder du dessa ip-block till nästa version. När du återanvänder ip-block i en ny krets så måste du vara säker på att de blocken du använder fungerar som de ska. Detta kräver att kretsen har verifierats grundligt med tydliga tester för att säkerställa att ip-blocket fungerar som det ska.

Verifikationen av integrerade kretsar görs med en simulator. Kretsens beteende simuleras och testas innan den skickas iväg till produktion. Med större och större kretsar så har det blivit omöjligt att testa alla möjliga kombinationer av stimuli en krets simuleras med. Det finns inte tillräckligt med datorkraft att simulera alla möjliga kombinationer. Istället nöjer man sig med att testa en en del av kretsen och spara olika mätvärden från simuleringen. Man nöjer sig med att uppnå dessa mätvärden för att säga att kretsen har testats fullt ut. Du som verifikationsingenjör skapar nya tester och simuleringar för att uppnå dessa mätvärden. Verifikationen av kretsar tar tid, något som är dyrbart när konkurrensen är hård och du vill vara först ut på marknaden med den nya tekniken.

För att minska tiden det tar att uppnå tillräckligt bra simuleringsresultat så undersöker det här projektet om maskininlärning kan hjälpa till som verktyg i verifikationsprocessen. Det syftar till att studera om maskininlärning som ett hjälpmedel kan användas för att snabbare hitta mönster och se samband i en krets för att reducera bearbetningstiden.

Popular Science Summary

Integrated circuits play a vital role in electronics such as mobile phones, computers, cameras and other electronic appliances we use daily. Including integrated circuits in some form is almost a must when producing modern electronics. With the help of integrated circuits, you can reduce the cost, size and speed compared to using conventional digital components on a circuit board. The number of transistors that build up an integrated circuit have increased. One circuit can contain millions of transistors that together give a circuit its function. In the early days, the number of transistors in an integrated circuit was relatively small. You could then start over from scratch when designing the next version. The circuits have become so advanced now that it is no longer profitable to start over when you design a new integrated circuit. Instead, you split up and reuse the design in so-called ip-blocks. Parts of the previously developed ip-blocks are used as a starting point for the next version of your integrated circuit. When you reuse an ip-block you need to be certain that it works exactly as expected according to the specifications. This requires thorough verification with proper testing to ensure that the functionality of the ip-block still that works as it should.

The verification of integrated circuits is performed with a simulator. The circuit's behaviour is simulated and tested before it is sent out for production. With larger and larger circuits it is unfeasible to test all possible combinations of stimuli that a circuit should be able to handle. Instead you randomize the stimuli and use different metrics and scores to measure the completeness of the verification. A circuit is said to be fully covered when all different metrics have reached your desired goal. You as a verification engineer create new test and simulation runs to reach full coverage. The verification of integrated circuits takes time, something that is quite expensive when the competition is fierce and you want to be first on the market with your product.

This project examines if machine learning can be used as a tool to help reduce the time it takes to achieve a satisfactory simulation result. The purpose is to see if machine learning can help a user see patterns and connections between uncovered coverage holes and thus reduce the time dedicated towards verification.

Acknowledgements

We would like to thank Patrik Lislén and Lars Viklund at Axis Communications for composing and proposing this thesis to us. We also thank Lars Viklund and Gabriel Jönsson for giving invaluable support and guidance as our supervisors at Axis Communications.

We also thank Erik Larsson, our supervisor at Lund University that helped us throughout this journey with guidance to complete our thesis.

Table of Contents

1 Introduction **1**
1.1 Hardware Verification Pre-silicon 1
1.2 Related Work 5
1.3 Disposition 7
1.4 Method 7
1.5 Limitations 7

2 Theoretical Background **9**
2.1 Clustering 9
2.2 Dimensionality Reduction 11
2.3 Regularization 15
2.4 Visualization 16
2.5 Feature Engineering 18
2.6 Cluster Evaluation 20

3 Implementation **23**
3.1 Coverage Database 23
3.2 Construction of Dataset 23
3.3 Dimensionality Reduction 28
3.4 Further Dimensional Reduction 30
3.5 Clustering 31

4 Result **33**
4.1 Evaluation 33
4.2 Evaluation Combinations 35
4.3 Evaluation Results 35

5 Discussion and Conclusions **39**
5.1 Experiences 40

References **43**

A Latent Space **47**

B UMAP 67

C PCA 87

List of Figures

1.1 A simple abstraction of a testbench where a DUV is stimulated and the I/O to the DUV is monitored. A scoreboard can tell how well the DUV passed the simulation. 3
1.2 A simple abstraction of coverage driven verification with a test generator providing stimuli to the DUV, a driver that takes a high-level test and translates it to a low-level abstraction for the simulation. The checker detects failures in the design, and the coverage collector records the quality of the generated tests for a coverage model. . . . 4

2.1 Two randomly placed centroids are moved after multiple iterations until they no longer move. 10
2.2 Figure (a) shows responsibility messages being sent out. Figure (b) shows availability commands being sent around. 10
2.3 DBSCAN creates clusters depending on the density ϵ. 11
2.4 Illustration of PCA on a set of data points. 13
2.5 Basic structure of an Autoencoder. 14
2.6 Two cell RNN. 14
2.7 LSTM cell compromising of an forget, input and output gate. 15
2.8 Visualising the sweet spot. 15
2.9 One-hot encoded values for three entries using table2.1. 19

3.1 Illustration of cascading holes for concatenation of two signals and bitwise logic operations. 24
3.2 A simple circuit with 9 signals *A, B, C, D, E, F, G, H, I* and 2x `AND`, 1x `OR` and 1x `XOR` gate. 25
3.3 A tree graph of the signals in the circuit shown in figure 3.2. 25
3.4 Vocabulary created from 4 name entries. 26
3.5 Name entries transformed to feature vectors, here shown in a matrix, coloured squares represent that a word is present for that data point. 27
3.6 Tree graph to binary matrix representation. 27
3.7 Four different feature entries consisting of 8 dimensions and 4 timestamps, t0, t1, t2 and t3. 28
3.8 Layers in the Variational Autoencoder model 29
3.9 Layers used for operator dependencies 29

3.10 Layers used for signal dependencies 30
3.11 Each row in each matrix represent one data point, and each column represents one specific feature. The autoencoders are trained on their respective datasets, and the output in latent space that is combined is from only one data point. 30
3.12 Cumulative explained variance ratio as a function of components. . . 31

A.1 Visualization for feature combination INOS. 52
A.2 Visualization for feature combination INO. 53
A.3 Visualization for feature combination INS. 54
A.4 Visualization for feature combination IOS. 55
A.5 Visualization for feature combination NOS. 56
A.6 Visualization for feature combination IN. 57
A.7 Visualization for feature combination IO. 58
A.8 Visualization for feature combination NO. 59
A.9 Visualization for feature combination IS. 60
A.10 Visualization for feature combination NS. 61
A.11 Visualization for feature combination OS. 62
A.12 Visualization for feature combination I. 63
A.13 Visualization for feature combination N. 64
A.14 Visualization for feature combination O. 65
A.15 Visualization for feature combination S. 66

B.1 Visualization for feature combination INOS. 72
B.2 Visualization for feature combination INO. 73
B.3 Visualization for feature combination INS. 74
B.4 Visualization for feature combination IOS. 75
B.5 Visualization for feature combination NOS. 76
B.6 Visualization for feature combination IN. 77
B.7 Visualization for feature combination IO. 78
B.8 Visualization for feature combination NO. 79
B.9 Visualization for feature combination IS. 80
B.10 Visualization for feature combination NS. 81
B.11 Visualization for feature combination OS. 82
B.12 Visualization for feature combination I. 83
B.13 Visualization for feature combination N. 84
B.14 Visualization for feature combination O. 85
B.15 Visualization for feature combination S. 86

C.1 Visualization for feature combination INOS. 92
C.2 Visualization for feature combination INO. 93
C.3 Visualization for feature combination INS. 94
C.4 Visualization for feature combination IOS. 95
C.5 Visualization for feature combination NOS. 96
C.6 Visualization for feature combination IN. 97
C.7 Visualization for feature combination IO. 98
C.8 Visualization for feature combination NO. 99
C.9 Visualization for feature combination IS. 100

C.10 Visualization for feature combination NS. 101
C.11 Visualization for feature combination OS. 102
C.12 Visualization for feature combination I. 103
C.13 Visualization for feature combination N. 104
C.14 Visualization for feature combination O. 105
C.15 Visualization for feature combination S. 106

List of Tables

2.1 Example showcasing a vocabulary for different operands in digital circuits. 19

3.1 Features and their abbreviations. 25

A.1 Scores for the feature combination IOS. 47
A.2 Scores for the feature combination IO. 47
A.3 Scores for the feature combination IS. 47
A.4 Scores for the feature combination I. 48
A.5 Scores for the feature combination OS. 48
A.6 Scores for the feature combination O. 48
A.7 Scores for the feature combination S. 48
A.8 Scores for the feature combination INOS. 48
A.9 Scores for the feature combination INO. 48
A.10 Scores for the feature combination INS. 49
A.11 Scores for the feature combination IN. 49
A.12 Scores for the feature combination NOS. 49
A.13 Scores for the feature combination NO. 49
A.14 Scores for the feature combination NS. 49
A.15 Scores for the feature combination N. 49
A.16 Scores for Affinity and all feature combinations. 50
A.17 Scores for k-means and all feature combinations. 51
A.18 Scores for OPTICS and all feature combinations. 51

B.1 Scores for the feature combination IOS. 67
B.2 Scores for the feature combination IO. 67
B.3 Scores for the feature combination IS. 67
B.4 Scores for the feature combination I. 68
B.5 Scores for the feature combination OS. 68
B.6 Scores for the feature combination O. 68
B.7 Scores for the feature combination S. 68
B.8 Scores for the feature combination INOS. 68
B.9 Scores for the feature combination INO. 68
B.10 Scores for the feature combination INS. 69

B.11 Scores for the feature combination IN. 69
B.12 Scores for the feature combination NOS. 69
B.13 Scores for the feature combination NO. 69
B.14 Scores for the feature combination NS. 69
B.15 Scores for the feature combination N. 69
B.16 Scores for Affinity and all feature combinations. 70
B.17 Scores for k-means and all feature combinations. 70
B.18 Scores for OPTICS and all feature combinations. 71

C.1 Scores for the feature combination IOS. 87
C.2 Scores for the feature combination IO. 87
C.3 Scores for the feature combination IS. 87
C.4 Scores for the feature combination I. 88
C.5 Scores for the feature combination OS. 88
C.6 Scores for the feature combination O. 88
C.7 Scores for the feature combination S. 88
C.8 Scores for the feature combination INOS. 88
C.9 Scores for the feature combination INO. 88
C.10 Scores for the feature combination INS. 89
C.11 Scores for the feature combination IN. 89
C.12 Scores for the feature combination NOS. 89
C.13 Scores for the feature combination NO. 89
C.14 Scores for the feature combination NS. 89
C.15 Scores for the feature combination N. 89
C.16 Scores for Affinity and all feature combinations. 90
C.17 Scores for k-means and all feature combinations. 90
C.18 Scores for OPTICS and all feature combinations. 91

Chapter 1

Introduction

Today, verifying digital hardware designs pre-silicon is primarily done using simulation and techniques such as coverage-driven constrained random verification. Metrics, including functional, assertion, line, toggle, and condition coverage, play a fundamental role in measuring the verification completeness and progress. Verification engineers study these metrics to answer questions such as:

- Have we verified the design adequately?
- What aspects have not been exercised sufficiently in the design?
- How do we improve test coverage to reach the verification goal?
- What kind of test cases contribute to the most relevant stimuli?
- How much progress are we making?

Ideally, the verification metrics view should allow verification engineers to quickly get a relevant overview of the current status, identify patterns, and see what progress has been made. With the growing complexity of digital hardware designs, the time and effort required to verify a design properly increase. Verification engineers spend a considerable amount of time and resources adequately simulating, verifying, and testing new or refurbished designs.

What can be interesting is to see if this process can be sped up with the advent of machine learning to automate part of the process, thus reducing the time required to reach 100% coverage.

To understand how we can facilitate the digital design verification process and identify the parts where it is possible to apply machine learning, we will introduce the reader to how engineers commonly perform hardware verification for digital circuits before fabrication.

1.1 Hardware Verification Pre-silicon

With more advanced circuits appearing, produced by an ever-growing code base, it is impossible to test and simulate all possible inputs [1] that a circuit should support. It is not feasible to try all potential input vectors as the simulations would consume a significant amount of time and resources. For example, to thoroughly test a 32-bit full adder, all $2^{(64)}$ possible combinations need to be checked. It is easy to understand that this is impractical and a waste of both simulation time and resources. Instead, the design should be thoroughly tested with a satisfactory result that confines to a specification. A design should be verified to make sure that it works under normal operation and that all functions have been tested and verified one way or another.

Traditional verification techniques are based on pre-silicon simulations and formal verification[2]. In a simulation environment, the design is placed in a testbench where input vectors are applied to test the device under verification (DUV). This environment is more often than not design specific, challenging to adapt to another design and takes a considerable amount of time to set up.

Formal verification attempts to prove the correctness of the design with mathematical certainty [1]. The difference between the two is that simulation-based verification *tests* the design while formal verification *verifies* if the written code should behave correctly [1]. In this thesis we focus on the former, how to improve simulation-based verification.

Currently, there exist tools and methods that automate the process based on pseudo-random test generations with coverage metrics [1]. A random generator generates input vectors used to test the DUV, and with a set of constraints, verification engineers can guide the generator to achieve different scenarios. The constraints are fine-tuned until satisfactory results are reached. This process of fine-tuning the constraints and changing the tests goes on until the design has reached sufficient coverage.

Setting up a testing environment has become even more straightforward with the advent of Universal Verification Methodology (UVM). UVM is a framework of classes in SystemVerilog that provides building blocks for test benches [3]. This methodology has the advantages of being portable and reusable. Thus, the environment with random test vectors can be applied to more than one design, reducing the time needed to simulate and verify a new design.

1.1.1 Simulation Based Verification

Simulation-based verification uses test benches to stimulate a DUV and monitor how it behaves and responds. A simple setup can be seen in figure 1.1. Engineers must carry out extensive simulations to capture all possible bugs that may exist, and manual tweaking is necessary to capture the complete functionality. They also need to take much care in finding corner cases that may break the design's functionality.

Coverage driven verification is a form of functional verification by simulation of a DUV. It is a systematic approach where tests are generated to provide closure of coverage holes in the design [4]. The design is placed in a testbench containing the various parts, as in figure 1.2.

The test generator provides different test scenarios as input to the system. For example, this can be a picture from a camera sensor representing a surveillance camera's real-world scenario. The stimulus driver creates stimuli suited for a particular DUV. The stimuli driver acts as the outer environment encapsulating the DUV. The DUV is monitored by a checker that detects possible failures during the simulation The coverage collector records the quality of the test to be reviewed.

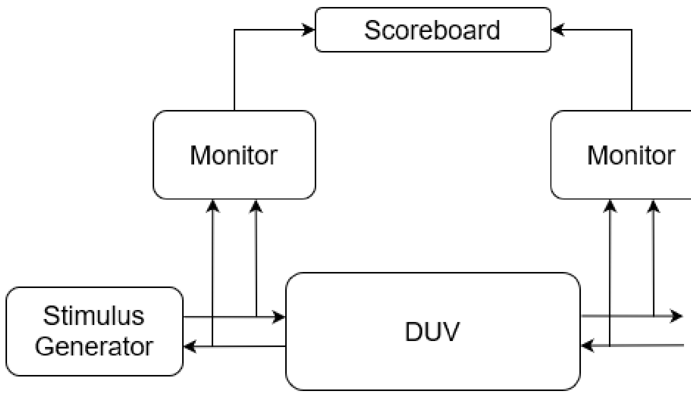

Figure 1.1: A simple abstraction of a testbench where a DUV is stimulated and the I/O to the DUV is monitored. A scoreboard can tell how well the DUV passed the simulation.

There exist different coverage metrics that provide information on how well a design has been verified. The recorded metrics are used to verify if the design is fully covered or not and act as a sort of input, deciding whether to run more tests or create a new directed test to close the remaining coverage holes.

It is an iterative process to review the coverage recordings and update the test generator to close even more coverage holes. When enough coverage holes have been closed and the coverage report reports an acceptable level of coverage, the design is fully covered.

1.1.2 Tests and Coverage Bins

A verification engineer must write tests to verify a design, providing constraints for random test vectors and, in some cases, create directed tests. Separate tests with custom constraints are created to test as much as possible of the design. It is often unnecessary and complex for a test to check the design's full functionality. Instead, multiple tests that focus on subsets of features can be combined to provide test coverage for the whole design.

The input vectors and the design's behaviour can be compared against a golden model to confirm the written RTL code's functional properties. This can be time-consuming to set up and adequately verify. Coverage metrics provide a report on the coverage holes that were detected during a simulation. The coverage metrics are recorded in different coverage bins, containing all information about when and how a signal, module, or design achieved a metric. There are various coverage metrics available in a verification engineers toolbox. Some of them are explained below.

1.1.3 Functional Coverage

Functional coverage is the metric on how much of the functionality of the design has been executed [1]. A functionality could, for example, be a design connected

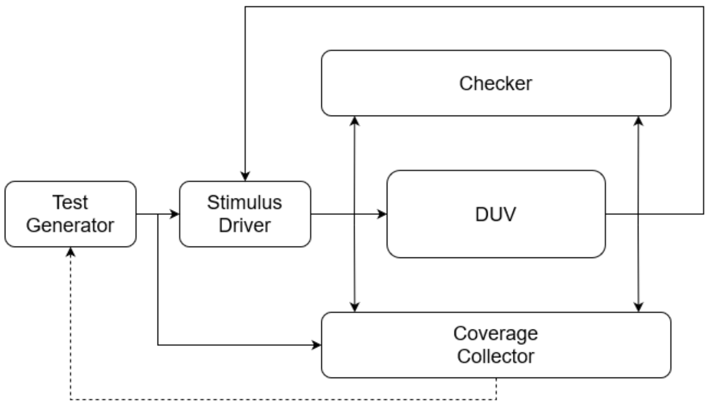

Figure 1.2: A simple abstraction of coverage driven verification with a test generator providing stimuli to the DUV, a driver that takes a high-level test and translates it to a low-level abstraction for the simulation. The checker detects failures in the design, and the coverage collector records the quality of the generated tests for a coverage model.

to a bus, then you need to exercise all the features of the bus protocol. Functional coverage does not prove that the features were exercised correctly or for the right reason, only that the functionality was executed somehow.

1.1.4 Structural Coverage

Structural coverage, or code coverage, ties into the implementation and representation of the design. Below are a few types of structural coverage explained in more detail.

Line Coverage

Line coverage is a metric that indicates which lines have been executed during the simulations. Line coverage that is not fully covered can imply that there is dead or unreachable code present in the design [1]. A higher level of coverage can be achieved by running more tests, writing directed tests to target previously unreached parts, removing unused parts or excluding them as they might not be used in the current configuration.

Toggle Coverage

The toggle coverage metric collects information whether a signal or latch in the design has toggled from high to low or vice versa [1]. It is up to the verification engineer to go through the toggle metric and see whether a signal should be able to toggle or not. Signals that have not toggled might indicate that more directed tests are needed to cover these toggles. It can also be possible that this is expected behaviour and that some signals should never toggle. In that case, such an occurrence shall be excluded from the report. Toggle coverage is seen as the most uncomplicated structural coverage metric to implement in a verification flow [1].

Condition Coverage

Conditional coverage is a metric that indicates if we have been able to exercise all possible combinations of conditional statements during simulations. Just as with line and toggle coverage, we might need to run more tests to get sufficient coverage of this metric. Exclusions might be needed to suppress warnings for when two signals cannot reach a particular combination in the design [1].

A design is said to be closed when it has reached 100% coverage in all these metrics. The way to get there differs from design to design. The random generator is not intelligent and does not know what to test. A verification engineer has to look at the missed coverage bins and see if they should be excluded, fixed in code or if directed tests should be written to cover these uncovered coverage holes. The time required to reach 80% of the coverage only requires 20% of the total verification time. Thus the last 20% can take up to 80% of the development time to reach [2]. In this thesis, we strive to improve the verification process to reduce the time spent on design verification.

1.2 Related Work

There has been a drive to improve the verification process in various stages to reduce the required engineering effort. Many of these solutions use artificial intelligence or machine learning to automate the process and create an effective verification process with minimal engineering effort.

E. E. Mandouh et al. tries to simplify the coverage analysis by clustering functional coverage goals that share similar items [5]. They do this in a two-stage process. First, they look at the association between cover-crosses by making a binary connectivity matrix. After that, K-means clustering, with Jaccard similarity for the distance, is applied and groups together highly correlated cover-crosses. Then another clustering iteration is made to sub-group the previous clusters so that clusters with low coverage ratios are selected first for coverage hole analysis.

A. Wahba et al. used machine learning on regression data from previously failed simulations and their signatures to cluster faults together [6]. They used machine learning and rule learning to find signatures in faulty code and prioritize RTL simulation defects over verification check failures. The idea was that prioritizing RTL-defects would yield better RTL quality and more efficient design development cycles.

D. Maksimovic et al. combined affinity propagation and a support vector machine to find the root cause of a failure in source code [7]. They use a three-part system to clustering together potential faults. They look at faults in the RTL code combined with possible faulty commits in version control and regression testing to pinpoint where the root cause exists in the defective code. By transforming the affected start- and end-lines to a point graph for each affected file, the defective RTL code could be clustered together with affinity propagation and Euclidean distance as the distance function. The clusters could then be reviewed collectively as a starting point for the verification engineers debugging process.

Golagha, Mojdeh et al. conducted a study to find faults without looking at coverage data. Instead, using sources not tied to the RTL code to cluster failing tests [8]. They used agglomerative clustering to group failing tests based on non-coverage data such as Jira log history, general features and fail/pass-history. Using historical data, they transform each test into a binary matrix representation and use agglomerative clustering to cluster the faults together. They based the number of clusters on historical data of several failed tests and confirmed faults.

There has also been a review by Eder, Kerstin I. et al. showcasing the possibilities of using *coverage Derived test Generation*, coverage derived test generation, powered by machine learning to speed up the testing of digital circuits [2]. They discuss how AI-driven randomization of input stimuli can help cover hard to reach coverage points faster and more efficiently than just using pure randomization input, as seen in figure 1.2. Their proposed methods can capture and cover coverage holes that would otherwise have been missed in a normal simulation.

Digital circuits are generated from a top-level RTL description synthesized into a netlist and then placed during routing on silicon or an FPGA. The netlist is a description of the circuit and its function. We hypothesize that some features shared between missing coverage holes can be captured from this. Therefore, it is also interesting to look at how previous work has tried to measure the similarity between circuits. Some of the proposed methods could hopefully be applicable to our model.

Two separate papers try to find similarities in circuits to facilitate place, route [9] and IP reuse theft protection [10]. Both papers discuss a method of representing the circuit as a graph and use graph comparisons to find similar circuits or similarities in a subset of a circuit. Graph comparison is complex and takes excessive computational time when applied to larger circuits and should, therefore, be avoided when employed as a one-to-many search. Instead, it is more sustainable to opt for simpler matching and a smaller subsection of the design. Zeng discusses how different synthesized netlists can look different but have the same functionality [10], a simple *XOR* gate may be realized in different ways, which is something that needs to be considered when trying to find circuit similarity. Both papers suggest that deconstruction of a circuit down to a graph-representation can help find similarities among circuits or sub-circuits.

We were unable to find anyone applying machine learning on the holes in the coverage metrics from our research. We hypothesize that the current verification process can be optimized by applying machine learning to cluster uncovered coverage points to find similarities among them. Data visualization techniques can then be used later to facilitate the analysis of the machine learning algorithms results.

1.3 Disposition

The disposition of the thesis is as follows. In the first chapter, we provide a general introduction to the verification of digital circuits and the methodology that exists today. In chapter two, we provide the reader with a theoretical background of the subject. This chapter will go through all methods used in the final prototype and focus on three major topics: cluster algorithms, dimensionality reduction techniques and evaluation. The development of our proposed implementation will be explained in chapter three. In this chapter, we will give the reader a complete understanding of our proposed solution. An evaluation of the said solution is given in chapter four, with the result covering the performance. A discussion of the thesis and our conclusions can subsequently be found in chapter five.

1.4 Method

Here we will briefly explain our method consisting of the phases: preparation and research, data exploration, feature extraction, prototype implementation and evaluation of the said prototype.

First, we studied the coverage database and the currently available tools used to analyze coverage from simulations at Axis. From this foundation, we gathered information on different clustering algorithms used in a similar setting that could cluster the coverage information. We also went through possible visualization methods during this phase and how different clustering algorithms could be visualized in a later stage.

Next, we explored what information and features could be extracted directly from the coverage database as input to a clustering algorithm. We investigated in what different ways we could represent holes in the coverage and how we, at a later stage, could compare the representations to each other both individually and in combinations. We wanted to conduct a study to evaluate which features best portray similarities between the coverage holes. A working prototype implementation was developed with this in mind.

Together with engineers from Axis, we chose to evaluate a design that had reached 100% coverage. By looking at a design with full coverage, we could use a backward approach to iteratively remove some of the tests and manually induce holes in the design to evaluate the results.

1.5 Limitations

We have limited our scope to only focus on toggle coverage and no other coverage metrics. Our thought process is, if it works for toggle coverage, which is a simple coverage metric, it can be further expanded to include more, if not all, coverage metrics mentioned above.

Chapter 2

Theoretical Background

In this chapter, we will discuss the theoretical background of our work. We will begin explaining machine learning and, more explicitly, clustering. We will go through what clustering is, and some commonly used clustering algorithms. We also go through dimensional reduction methods, as they play a crucial part in producing valid results since most cluster algorithms work better in a lower dimension. After that, we go through how a dataset should be produced and encoded in order to facilitate clustering. Following is an overview of how to evaluate a clustering method and how it can be given a tangible success score.

2.1 Clustering

There exist many methods for clustering a dataset, each with its pros and cons. In this section, we go through some of the more prominent in the machine learning field. The basics for a clustering algorithm are to group similar data points together and see connections and similarities between different data points, something that can aid us in finding related coverage holes. A data point is often represented as a feature vector, and multiple data points form a dataset. A feature vector contains tangible values that can be compared against one another to judge how close or far apart two data points are.

2.1.1 k-means

k-means clustering is one of the simplest clustering methods. It is based on picking k randomly selected centroids, the centre of an imaginary cluster. Each data point is assigned to be part of the nearest cluster. After that, the centre of the centroid is moved to the mean position of all data point in its cluster. This is repeated until no changes happen between two consecutive runs [11]. This method's major drawback is that it is not good at detecting complex shapes of scattered data points. The k variable also needs to be set, which means that one must know the expected number of clusters before evaluation. Therefore, k-means is only suitable if you have some knowledge of the expected number of clusters beforehand and is ill-suited for a clustering problem with an unknown number of real clusters.

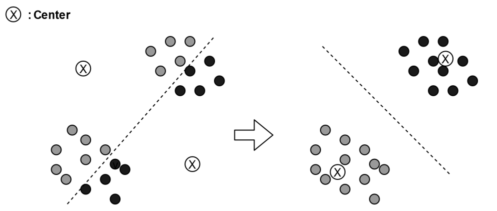

Figure 2.1: Two randomly placed centroids are moved after multiple iterations until they no longer move.

2.1.2 Affinity Propagation

Affinity propagation is a clustering algorithm [12] where data points, called nodes, send messages between each other about their preferences. The nodes pass two types of messages around: "responsibility" and "availability". A responsibility message tells the receiving node the accumulative evidence for it being suitable as a potential *exemplar* from the sending node. The availability message goes in the opposite direction and tells the receiving node the accumulative evidence that the receiving node should take the sending node as an *exemplar*. The messages will update each node's preference until close to no change can be seen in the network or until a predetermined number of iterations are reached. This algorithm does not need to know a predetermined number of clusters beforehand, as the network will determine by itself how suited each node is to be an exemplar. All points sharing the same exemplar belong to the same cluster. Affinity propagation has some good characteristics. The data points do not need to be in continuous space, as each node only sends a metric of evidence about the suitability to another node. The major drawback of this design is that the method will operate with the time complexity $\mathcal{O}(n^2)$ [12].

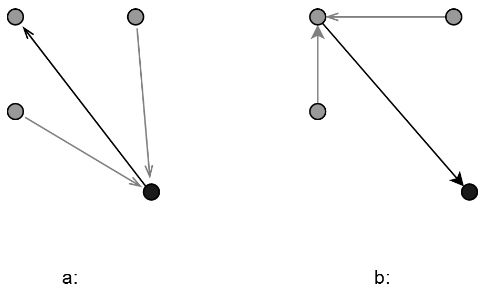

Figure 2.2: Figure (a) shows responsibility messages being sent out. Figure (b) shows availability commands being sent around.

2.1.3 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) [13], has the major advantage that it does not need to know the number of clusters beforehand.

It works with two input parameters, ϵ and *minPts* specifying the radius of a points neighbourhood and the minimum number of neighbour points for it to be considered a core point. A border point is a point ϵ distance from a core point but with fewer neighbouring points than the threshold for a core point. The algorithm is based on three key definitions, *directly density-reachable*, *density-reachable* and *density-connected*. A cluster is made out of one or more core points *density-connected* to each other and possibly one or more border points *directly density-reachable* from a cluster's core point. Points not *directly density-reachable* from any core point are considered noise and not included in any cluster. DBSCAN can find any arbitrarily shaped clusters and even clusters inside other clusters (if the clusters are separated with a distance greater than ϵ). It can also ignore clustering noise points [13].

The disadvantage of DBSCAN is that it is not deterministic. If a point can be a border point to more than one cluster, it is not clear to which cluster the point should belong. DBSCAN also depends on the distance function and is therefore prone to high-dimensional data and ill-suited for datasets with a significant difference in densities [14]. The performance of the algorithm will degrade with varying densities.

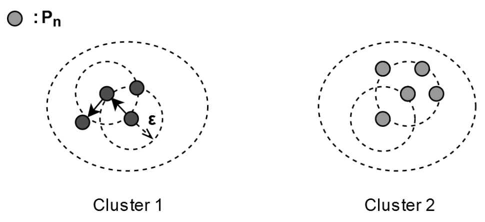

Figure 2.3: DBSCAN creates clusters depending on the density ϵ.

2.1.4 OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) can be seen as an extension to DBSCAN that tries to fix some of its flaws. The algorithm can detect clusters with varying densities, something that is a major drawback with DBSCAN. It achieves this by linearly ordering the database so that spatially close points become neighbours in the ordered database [15].

OPTICS only requires two input parameters, ϵ_{max} and *minPts*. With these, the algorithm calculates the core-distance and the *reachability-distance* for each point. This metric can be shown in a reachability-plot where clusters can be seen in the valleys and where dense clusters have deeper valleys than scattered clusters.

2.2 Dimensionality Reduction

One common problem among clustering algorithms is what's called *the curse of dimensionality*, that a clustering algorithm's performance will degrade with an ever-increasing dimensional dataset [16]. The problem arises when using Euclidean distance-based measurements between two data points. With increasing dimensions, the total volume will increase to the point where all data points in the dataset appear to be sparse [17], thus making it harder to classify two points as belonging to the same cluster. There are two major ways to reduce the number of features in a dataset, feature selection and feature extraction. Feature selection focuses on reducing the number of features that represent a data point. It is undesired to have a feature that only produces noise and no meaningful value when clustered. It will degrade the performance for most learning algorithms [18]. This has a more significant focus on supervised learning problems compared to unsupervised, as there is no clear way to know the relevance of a feature without knowing the label of the data point [18]. The number of clusters for unsupervised clustering is interrelated with the features, making the selection even harder [18]. Feature extraction (also known as feature projection) is based on mapping features in the high-dimensional space to a lower-dimensional space and at the same time minimize the information loss [19]. One of the more well-known methods is the Principal component analysis (PCA), an unsupervised method that projects data into a subspace of the original dimensions and hopefully manages to reduce it. Using autoencoders is a different method that comes in various variations and uses neural networks to reduce the number of dimensions. The customization with autoencoders has its roots in the neural network with one or more hidden layers. It is easy to stack hidden layers on top of each other, reducing the dimensions even more than with just one layer. The ability to customize the hidden layers can help with achieving accurate dimensionality reduction for a specific application.

2.2.1 Principal Component Analysis

Principal component analysis (PCA) aims to reduce a dataset's dimensionality while still retaining the variations. A set of observations is described with a set of principal components that retain the maximum variance of the original observations [20]. The first principal component is a line that best fits all observations in the high dimensionality space. Each other principal component is made to fit all observations while being orthonormal to each previous principal component. The observations can then be projected onto the axes to get a reduced number of dimensions representing each observation.

The good thing with PCA for reducing the dimensionality is that it retains the similarity between two points if the clustering is done with Euclidian distance [20]. The distance between two points will be unchanged if there are the same number of principal components used as the number of dimensions in the original observations. By projecting the observation on the first m principal components, the pairwise similarity between two different points becomes an approximation to the similarity metric in the original dimensions.

2.2.2 Autoencoders

An autoencoder reduction algorithm can be described as five different parts; *input layer, encoder, latent space, decoder and output layer*, connected in that order as seen in figure 2.5. It uses neural networks (NN) to build up both its encoder and

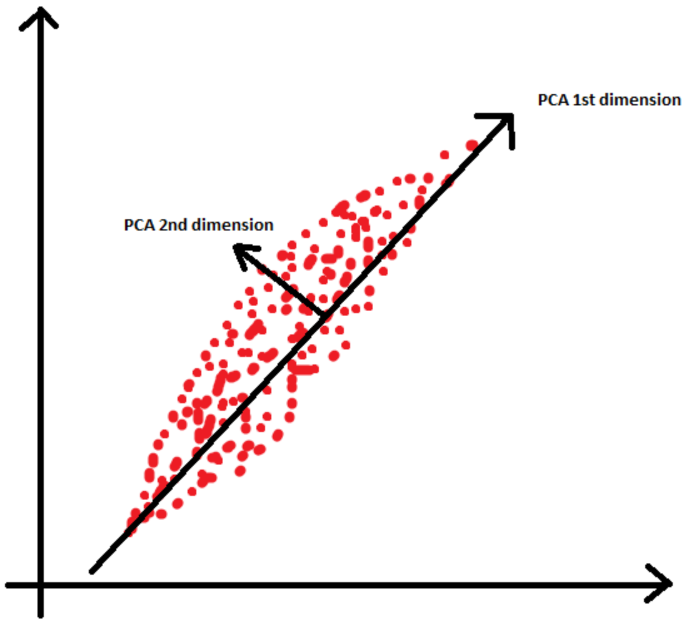

Figure 2.4: Illustration of PCA on a set of data points.

decoder. The encoder will take the input data, $x \in R^N$, and reduce the number of features to a more manageable size, $z \in R^M$ where $M < N$. This compressed data in latent space should be a new way to describe the original data fed through the encoder. The decoders job is to do the reverse. It tries to generate an accurate representation $\tilde{x} \in R^N$ of x from the encoded data z in the latent space. The encoder and decoder are trained on a training dataset to produce an accurate reconstruction of $\tilde{x}$ for all data points. With the encoder trained, a data point feed through the network will get a dimensionality reduced representation in the latent space. Clustering algorithms can then use the reduced dimensions in latent space to find similarities between other dimensionality reduced data points. The use of autoencoders as dimensionality reduction algorithm has been used for single-cell RNA analysis [21] and have proven good results in making complex data available to view in a lower-dimensional space.

2.2.3 RNN and LSTM

A recurrent neural network (RNN) is a neutral network that behaves the same as an ordinary neural network but contains a memory of the last output, that is to be passed as an input to the network. This makes it possible to train on a sequence of data assumed to depend on each other of an indefinite length. This means the RNN can train on variable input sequences without a problem and can be represented as non-sequence data in latent space. However, RNN has a problem that is called vanishing gradient decent.

As the RNN is trained with backpropagation and gradient-based methods, the

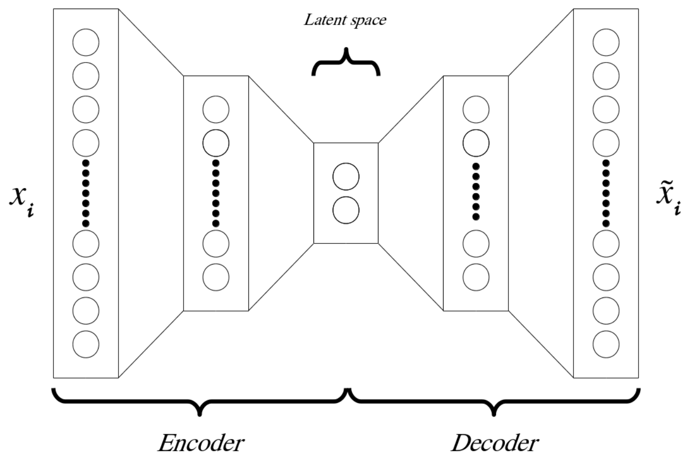

Figure 2.5: Basic structure of an Autoencoder.

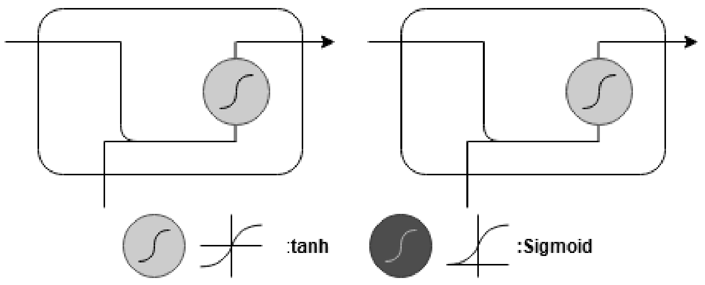

Figure 2.6: Two cell RNN.

update to the weights will tend to go to zero or infinity. The weights get updated by a portion of the derivative of the error during a training iteration. The gradient can sometimes be minimal with stacking cells, and thus the gradient will be close to zero, preventing changes to the weights. In figure 2.6, you can only see two cells. In such an example, this would not be a problem, but with more and more cells after each other, the vanishing gradient descent problem occurs.

Long short-term memory (LSTM) is a solution to the problem mentioned above of RNN. It uses the *tanh* and *sigmoid* functions to transform the input while keeping its cell state and hidden state. An LSTM cell can be seen in figure 2.7 with the hidden state and cell state entering the cell from the left and respective state exiting on the right. The current input will enter from below. LSTM works by forgetting irrelevant information from the input and the hidden layers with a sigmoid function. After that, the cell state is updated with only relevant information, squashed in the range $-1to1$ with the *tanh* function. The output stage decides what parts of the hidden state should be carried over to the next cell from the current cell state. The cell state is passed through a *tanh* function again to keep the output bounded.

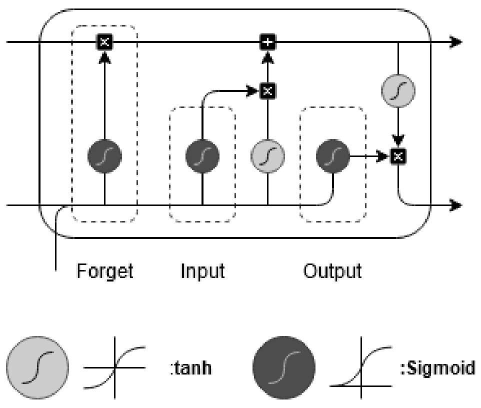

Figure 2.7: LSTM cell compromising of an forget, input and output gate.

2.2.4 LSTM Autoencoder

As an autoencoder uses a neural network as an encoder, we can exchange it for an LSTM network instead to support sequences of data. The LSTM takes the regular fully connected layers place in the encoder to combine the positive sides of autoencoders with the advantage of handling the sequences supported by LSTM. The two principles are the same but must be implemented differently in practice when realizing a system.

2.3 Regularization

One common challenge with neural networks is over-fitting. When training a network, one often strives to achieve a good but generalized model. A model with too much capacity might lead to it being overfitted or overtrained on the specific training data, whereas a model with too little capacity cannot learn the problem at all. The challenge lies in finding this sweet spot.

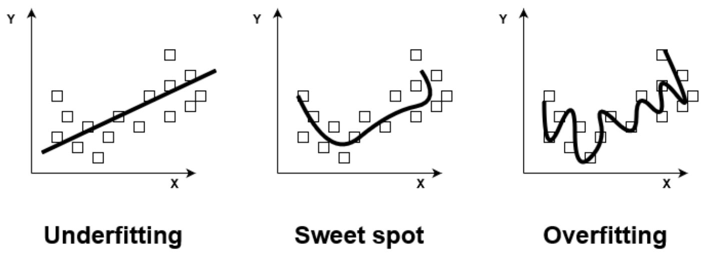

Figure 2.8: Visualising the sweet spot.

Many methods exist in aiding with this problem. The easiest is to simplify the model and constrain the overall complexity by reducing the number of neurons directly in the layers or the actual layers themselves. However, for complicated problems, this may be impossible and lead to the model being unable to learn anything. Another straight forward approach is to train the model on more data, but this places demands on data availability which could be challenging in a real-world scenario. Thankfully we have methods such as “dropout”, where a random amount of neurons are dropped each iteration, forcing generalization by training multiple subsets of the full network.

2.3.1 Variational Autoencoders

One way to regularize autoencoders is to use a variational model. The variational autoencoder tackles the over-fitting problem using a distribution where a data point might be in latent space instead of a single point. During training, the input point’s position to the decoder in latent space will differ from its actual position, as it is sampled from a distribution around the point. This means that the autoencoder is trained to reconstruct the data point from the latent space distribution instead of the actual point. Variational autoencoders have the effect that quite similar data points will appear closer together in latent space than a regular autoencoder where two completely different data points could be situated at a close proximity [22]. Variational autoencoders have also been proven to reduce the number of dimensions for datasets where the number of dimensions is far greater than the number of data points in the dataset [23].

2.3.2 Regularizing LSTM Autoencoders

To reduce over-fitting in a recurrent network, dropout can be applied. One approach is a Variational LSTM that applies the dropout at each time step [24].

2.4 Visualization

There exist dimensionality reduction methods that are adapted to visualize the result for the human eye. They specialize in transforming high-dimensional data down to just a few dimensions suitable to produce visually pleasing results. Two of them are mentioned below, t-SNE and the follow-up adaptation UMAP. They are both destructive in keeping density information from the original dataset and should therefore be used for their purpose of visualization.

2.4.1 t-SNE

The t-SNE method is specialized in turning high dimensionality data into 1, 2 or 3 dimensions that can be plotted. The algorithm works by taking all data points and randomly placing them on a line (in 1-D) or a graph (in 2-D). Now all points are evenly distributed in the plane, and similar points might be either close or far away. Iteratively each point is moved in the direction with the highest positive force. Similar data points attract each other, while unrelated data points repel each other. This process will go over all data points until all data points have reached their desired place in the graph, where similar points are closer together than dissimilar points.

For one point p_i the force between each other point in the data set needs to be calculated. This is done by using a Gaussian distribution. Each other data point will have a certain distance from p_i and will be placed in that distribution. The closer a data point is to p_i, the higher the density value it will be assigned. This density value is the force relationship between two different points in the original high dimensional space. These density values are all normalized to combat the problem where one point has a large number of close neighbours while other neighbours are far away. The normalization is done so that the combined density for one point p_i will always sum up to 1.

In the low dimensional space, each point's relationship does not use the Gaussian distribution but instead, the t-distribution. T-distribution prevents most of the data points from gathering at the centre and forcing them to spread out, making it easier and more transparent for visualization. The t-SNE algorithm tries to minimize the Kullback–Leibler divergence, how much one distribution differs from another, between the distributions in high and low dimensional space. This fitted t-distribution is then used, as described above, to move the points in the low dimensional space around until each data point is close to its neighbours [25].

2.4.2 UMAP

Uniform Manifold Approximation and Projection (UMAP) [26] is an algorithm with its base in maths as a dimensionality reduction algorithm. Just as t-SNE, it tries to tackle the problem of reducing a high dimensionality data set into fewer dimensions and still retaining valuable relation between data points in the new feature dimension [26].

As the name suggests, UMAP takes advantage of manifold approximation and learning techniques. It boils down to the thought that the dimensionality is artificially high. Instead, each data point can be described with only a few parameters representing the actual dimensions of the data set. Meaning a high dimensionality feature set should only depend on a small number of parameters. UMAP construct this manifold by producing local approximations and then stitch them together, constructing a topological approximation. After the manifold approximation has been evaluated, UMAP constructs *fuzzy sets* that can be represented on the manifold. The high dimensionality topological representation can then be put down into a low dimensional representation by reducing the error between the two topological representations. This can be reduced to two or three dimensions to help visualize the data for the human eye.

Each point on the manifold represents another point in space for each data point in the dataset. To find the connection between two data points and their relation, UMAP tries to create geometric blocks of simplices. A geometric shape that is of k dimensions is called k-simplex. A 0-simplex is a point, 1-simplex is a line, 2-simplex is a triangle, and a 3-simplex is a tetrahedron and so forth. This structure can be easily represented by a set of objects and their faces. On their own, these simplices do not provide any valuable information for the algorithm.

It comes from when two simplices are put together with their faces against each other and provides a simplicial set. We choose not to delve deeper into the maths behind the simplicial set, as it will have little to no use for this thesis's point. These simplicial set will be the building block between the data points. How these points get connected will be covered later. First, we need to look at how these should be connected on the manifold. A Simplex should be created of the nearest neighbours between data points. This is done by looking in a radius around each point, to judge which points are close to each other. The problem arises in empirical data when data points are clumped together. This will result in a high dimensional simplex, which is undesired and defeats the purpose of what we are trying to achieve. It is solved by assuming that all points are uniformly distributed on the manifold, then there won't be a problem to pick the perfect radius to avoid these high dimension simplices. In the real world, the data points won't be uniformly distributed. Therefore the radius for each point needs to be chosen individually to reach the k-th nearest neighbour. In a theoretical 2-dimensional space, this radius from a data point represents a projection on the manifold in what can be euclidean space, if points are separated by euclidean distances, of the distance between two points. Two neighbouring points with an overlapping patch in the high dimensional space will also have an overlapping patch on the manifold. Each point can then assume that the points it can see are a certain length away.

Now with the concept of the radius, we replace it with a fuzzy cover to give a meaningful distance from one point to another. This fuzzy cover provides a value between 0 and 1 for how far away a data point is from the source point, instead of just labelling if a data point is *inside* or *outside* the source point radius. The fuzzy cover of a point needs to reach at least its closest neighbour. The reason to use this fuzzy cover with a variable length is to combat the curse of dimensionality. The distance between each point in the high dimensional space will not be affected by it. Instead, the normalized distance between the data points will not depend on the number of dimensions.

After the fuzzy sets have been created with the simplices, they can be transformed into a lower-dimensional space, using parts of t-SNE and other repelling forces between data points to create a reduced feature dimension representation of the data set. For a deeper knowledge of how UMAP work, readers are advised to take a look at the paper [26] for UMAP, or the informative talk by one of the authors [27].

UMAP should have an advantage over t-SNE on high dimensional data set according to the authors [26] and is fast and reliable to use as a dimensionality reduction stage in a clustering problem. The implementation of UMAP that is used in this thesis comes from the authors own python module [28].

2.5 Feature Engineering

A clustering algorithm or other machine learning algorithms can theoretically be crafted to produce a pleasing result. They do, however, depend heavily on the dataset with its features. A machine learning problem and solution can only be as good as the input it is provided with. With faulty or incorrect data, the result is not satisfactory and should be discarded as a solution.

A machine learning model must be carefully constructed based on the design and availability of the data set. While it is possible to provide any number of features, the question still stands how relevant these features are. Are they character-defining and should be kept, or should they be dropped completely?

2.5.1 Encoding

Most clustering algorithms only work with numerical input. Therefore, some datasets need to be transformed and encoded to comply. Some necessary transformations need to be done.

1. Convert non-numerical data into numerical.
2. Resize input vectors to match the rest of the samples if dissimilarities in size exist.

Non-numerical data, such as categorical data, have to be represented as a number in some way. One example is primitive operands such as AND, OR and XOR that exist in digital circuits. They all represent one type of operation, but there is no way of measuring a meaningful numerical distance between the three. The naïve approach would be to assign each operand a number $\{1, 2, 3\}$, but that would assume an ordinal relationship between the operands and could lead to unwanted results. One solution to this problem is to use a one-hot encoding. In a one-hot representation of the data, one bit represents one value in the vocabulary. The vocabulary contains a list of all the possible values a categorical feature can take. The previous example can be seen in table 2.1, where part of a vocabulary of operands is illustrated.

Table 2.1: Example showcasing a vocabulary for different operands in digital circuits.

AND	0
OR	1
XOR	2
...	

A feature indicating that the AND operation was present is expressed as a vector with a value of "1" on the operands index in the vocabulary. In figure 2.9 the first entry represents the AND operand, the second XOR and the last entry is an OR.

```
[[  1.0  0.0  0.0  ],
 [  0.0  0.0  1.0  ],
           ...
 [  0.0  1.0  0.0  ]]
```

Figure 2.9: One-hot encoded values for three entries using table2.1.

The combination of the presence of both an AND and an OR operation can be binary encoded into the following vector `[ 1.0 1.0 0.0]`.

There are additional transformations that can be made to produce a better dataset. By normalizing numerical features to be in the range $[0, 1]$ or $[-1, 1]$ the performance of the algorithm can be enhanced [29]. Normalizing features also removes the possibility that some feature becomes too dominant over others.

2.6 Cluster Evaluation

To determine if a clustering algorithm succeeded to cluster relevant data points together and to assess its overall performance, one needs to perform an evaluation analysis. This analysis can be done either by looking at the internal structure or using external information to validate the result. The internal cluster structure can be evaluated using methods such as C index, Calinski-Harabasz, Davies-Bouldin, Dunn, Silhouette, Xie-Beni [30]. These methods give a score on how good the algorithms performed from a pure clustering perspective. The clusters can be externally evaluated by looking at which tests hit a coverage bin for a given signal in a cluster. The more similar the covering tests for each data point in a cluster are, the better. The theory is if one test hits multiple signals in a cluster, there would be a greater chance that a new tests or a slightly changed test for one signal would also cover the other signals in the same cluster. It should be noted that this only works if you look at a 100% covered design.

2.6.1 Davies-Bouldin

The Davies-Bouldin index gives a metric on how well-spaced clusters are and how dense the clusters themselves are [31], a lower score indicating good clustering characteristics. This index is chosen to give a score for how good the 2-D visualization will be. The score is defined as

$$DB = \frac{1}{n}\sum_{i=1}^{n} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right) \tag{2.1}$$

where n is the number of clusters; σ_i and σ_j is the dispersion of clusters i and j; and finally $d(c_i, c_j)$ is the distance between the two clusters i and j. The distance metric is evaluated in euclidean space, as the clustering will use euclidean distance.

2.6.2 Silhouette

The Silhouette index is, at its core, a tool to evaluate and aid in graphical validation of clusters [32]. The original paper used the Silhouettes method to create dendrograms to visualize the clusters. Later, it has become used as an index coefficient to measure a cluster algorithm's performance and to indicate if the number of clusters is acceptable [33]. Silhouette is a distance-based metric that compares the mean distance from a data point i and all other data points (equation 2.2) and the smallest mean distance to other clusters (equation 2.3).

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j) \tag{2.2}$$

$$b(i) = \min_{k \neq i} \frac{1}{|C_i|} \sum_{j \in C_k} d(i,j) \tag{2.3}$$

where C_i defines the cluster that point i is in; $d(i,j)$ is the distance between two points i and j in the same cluster; and C_k is any cluster that point i is not part of. The score for one point is then given by

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, if|C_i| > 1 \tag{2.4}$$

which will give a score $[-1, 1]$ for point i. A value close to 1 indicates a strong similarity between other points in the same cluster as point i. If point i is more similar to another cluster and thus probably should belong to that cluster, the value will go down towards -1. The silhouette coefficient is defined as:

$$SC = \max_k \tilde{s}(k) \tag{2.5}$$

where $\tilde{s}(k)$ is the mean of $s(i)$ for the chosen number of clusters k [33]. This score can help indicate if the clustering algorithm performed a good clustering with distinct silhouettes.

2.6.3 Multiple Site Similarity

In biology, it is often interesting to calculate how similar the composition of species in different locations are. One way of doing this is with the Sørensen similarity index [34], which defines the similarity between two sites of species A and B by

$$C_s = \frac{2ab}{a + b} \tag{2.6}$$

where ab is the number of shared species between the two sites A and B; a and b is the number of species found in A and B respectively. The index is bound within $[0, 1]$. The index gives valuable information about how alike the populations on different sites are. This can be applied to our problem with how similar two signals are depending on the test that hit a related coverage bin for the respective signals.

The proposed solution for using Sørensens similarity index for more than one site [35] can be used to find multi-class similarity for all data points in a cluster. The equation is given by

$$C_s = \frac{T}{T - 1} \left(1 - \frac{S_T}{\sum_i a_i} \right) \tag{2.7}$$

where T is the number of sites used to bound the index to $[0, 1]$; a_i the number of species in site i; and S_T is

$$S_T = \sum_i a_i - \sum_{i<j} a_{ij} + \sum_{i<j<k} a_{ijk} - ... \tag{2.8}$$

where a_{ij} is the number of shared species between the two sites i and j. The same applies to a_{ijk} which is the number of shared species in all three sites i, j and k.

Chapter 3

Implementation

This chapter will explain our implementation of the prototype we developed to examine different clustering algorithms and feature combinations. First, we will go through the available coverage database, the information it contains and a simplified netlist created for the simulation of a design. From there on, we will describe how the dataset was constructed and how the chosen features were extracted, encoded, and reduced, both for clustering and visualization.

3.1 Coverage Database

The coverage information exists in a coverage database. This database is filled with information about the coverage metrics; functional coverage, line coverage, toggle coverage, branch and condition coverage for a set of simulations on a design. It is from this database that the coverage information is extracted. The database recorded every hit a test caused during a simulation when a line or branch was executed, a condition was satisfied or when a bit toggled back and forth. From this information, we could get the signal names and the status of each coverage bin. There was also information on whether a signal should be excluded from the report. An excluded signal is ignored in the total score of the coverage report. This information only covered the names of the signals and the signal sizes and indexes for signal vectors. From this information, connections and potential clusters could be identified for a fully covered design with the help of the test results. However, this information was not enough to facilitate the creation of a dataset with present coverage holes. There was too little information to make valid connections between coverage holes. Instead, we chose to look at a netlist representation of the design to make connections between coverage holes. This was done with the hypothesis that the design and the netlist could represent the relations between signals and signal bits. This netlist was used to further expand our dataset that would be used for clustering.

3.2 Construction of Dataset

To facilitate clustering, distinct features of the signals had to be both extracted and constructed. We extracted information regarding the name, size, and bit position directly from the coverage database for each signal. From the netlist, we managed to construct a graph representation of both signal and operator dependencies.

From the coverage database, both the size and bit position information were interesting. In the past, at Axis, there had been noted that the least significant bit of a signal vector of a certain size was never covered, as the vector represented an even number. Another case is when big ranges of bits in some signals are left uncovered because of the same underlying cause. It is interesting to see if this behaviour can be seen on multiple signals throughout the design and later clustered together using the size and index as a feature.

We also looked at the signal names and how they were constructed—this metric varied on a design basis. In our initial study, we found that certain designs had a strict naming scheme for determining the signal's use and purpose. For such designs, we wanted to see if this information could further enhance similarities between the signals. Names constructed with multiple words were particularly interesting, as each word could give us a clue to the signals full use and relations.

The signal and operator dependencies were our way of representing relationships and similarities between signals on a “deeper level" compared to just the naming, size and index information that could be extracted directly from the coverage database. Our idea was to capture any possible prerequisite signals or operators present in multiple holes throughout the design, as an uncovered coverage point could propagate to other coverage points, see figure 3.1.

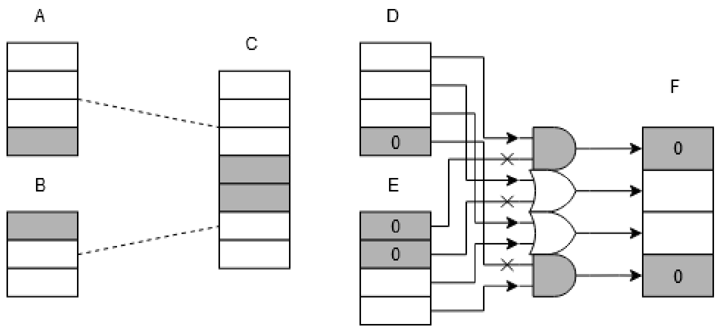

Figure 3.1: Illustration of cascading holes for concatenation of two signals and bitwise logic operations.

When traversing the netlist, the signal and operator dependencies can be represented as a tree graph with the selected signal as the root and each operator dependency as a leaf with the signal dependency as the branches connecting the leaves. This tree grows exponentially as each dependency can branch out in an unknown number of branches. This is problematic concerning both the time it takes to extract the features and the resulting feature vector's size. As mentioned in section 2.2, the curse of dimensionality says that in high dimensional space, all data appear to be sparse and objects are dissimilar [16]. This negatively affects any clustering algorithms ability to produce a trustworthy result.

Figure 3.2 shows a simple circuit that can be translated into a tree graph with different branches as seen in figure 3.3.

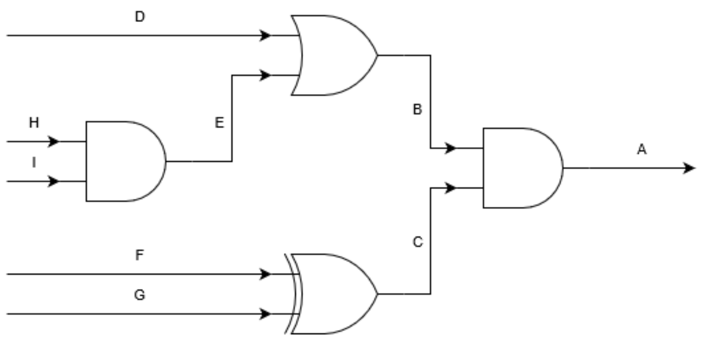

Figure 3.2: A simple circuit with 9 signals *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I* and 2x AND, 1x OR and 1x XOR gate.

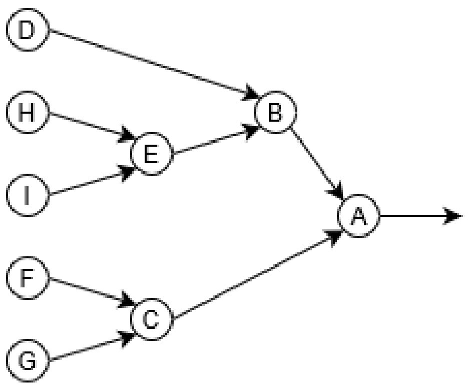

Figure 3.3: A tree graph of the signals in the circuit shown in figure 3.2.

Tracing every signal back to its origin would also cause every signal to share a large subset of features even if they don't have a close relationship. We decided to limit our dependency search backwards to the last occurring register or latch to avoid this.

The features we decided to use are listed in table 3.1 with abbreviations that will be used later for presenting the result.

Table 3.1: Features and their abbreviations.

Feature description	Abbrv.
Signal information (Size and Index)	I
Name	N
Operator dependencies for 1 CC	O
Signal dependencies for 1 CC	S

3.2.1 Encoding

Since most of our researched cluster algorithms work by comparing distances between points, some of our feature sets had to be transformed or encoded to a metric data type. It did not make sense to compare the distance between points represented by nominal or ordinal data. We had two types of problematic datasets, the name set and the two dependency sets. The size and index features are already discrete numerical by nature and did not need to be transformed. In this section, we will explain how the problematic datasets were encoded.

Name Encoding

As mentioned before, the names features was extracted directly from the coverage database. With the robust naming scheme present for certain designs, we hoped that we would be able to cluster the signals based on these features. Along with the need to transform the data into a metric data type, another problematic aspect that needed to be addressed was the fact that the names are unique.

Using purely unique names would give rise to a dataset equal to an identity matrix where each entry in the set would be equally separated from all other entries. A clustering algorithm operating on this set would produce the same number of clusters as there are entries, something that is unwanted. In the designs we had at our disposal to investigate, each name comprised of one or many shorter, nonunique words with an underscore "_" as a delimiter. By dividing names into shorter words, we could successfully construct a dataset with shared features between the entries.

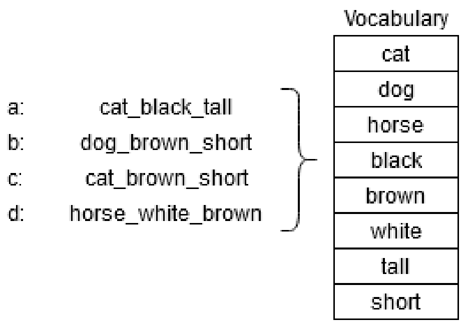

Figure 3.4: Vocabulary created from 4 name entries.

As the last step, we transformed this nominal data into something more usable for our algorithms by encoding it in a binary matrix. The resulting matrix is of the dimension $N \times M$, where N is the number of entries in the original dataset, and M is the size of the total vocabulary found across all entries when splitting the names. The feature vector for the name has a "**1**" in the resulting locations representing the presence of a certain word, while a "**0**" marks the absence of a word in the name.

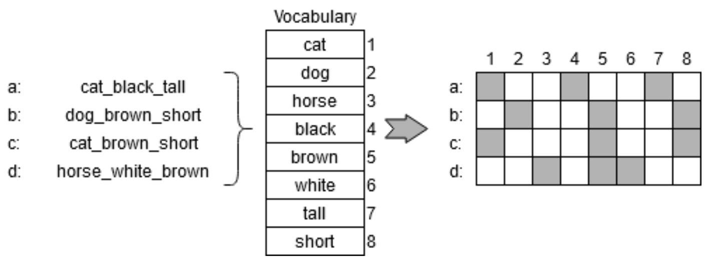

Figure 3.5: Name entries transformed to feature vectors, here shown in a matrix, coloured squares represent that a word is present for that data point.

Signal and Operation Encoding

Just as with the name encoding, a vocabulary was created for both signal and operation dependencies. The vocabulary for signal dependencies referred to specific signals in the design, while the operation dependencies were encoded by their type, as in the example in section 2.5.1. Both feature vectors also used binary encoding to represent the dependencies. The difference between the name and dependency feature sets is that the latter has an added dimension. The purpose of this is to try to capture both the sequence and branching of the dependencies. Suppose a signal has multiple dependencies on other signals and operators, as seen in figure 3.2. In that case, we want to capture the order in which these signals and operators relate to each other. For each step backwards in a signal's dependencies, we extract the immediate relations and construct the resulting binary matrix from the vocabulary, as seen in figure 3.6. Then, we merge the matrices from each branch at the same distance from the original signal into one. The result is a matrix of dimension $N \times M \times K$ where N is the number of entries in the original dataset, M is the size of the total vocabulary found across all entries, and K is the number of steps taken backwards in the dependencies, an example of this can be seen in figure 3.7.

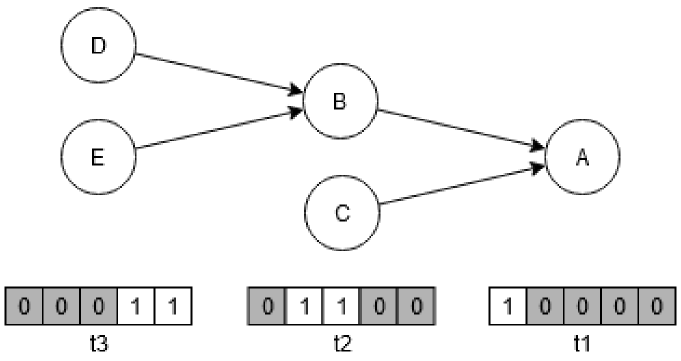

Figure 3.6: Tree graph to binary matrix representation.

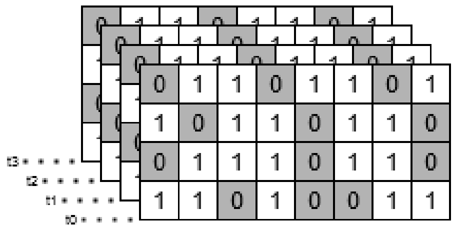

Figure 3.7: Four different feature entries consisting of 8 dimensions and 4 timestamps, t0, t1, t2 and t3.

3.3 Dimensionality Reduction

Before running the cluster algorithms on our datasets, we wanted to reduce the number of dimensions in the sets. Dimensionality reduction is a common methodology to deal with the curse of dimensionality. Our approach was to reduce dimensions with the help of autoencoders. Two types of autoencoder models were used for the two different types of complex datasets. We omitted this technique on the simple set, consisting of just the size and index, since it already had a low dimension and did not have to be encoded in the previous step. Many other dimensionality reduction techniques exist, such as PCA and LDA, to name a few [36]. However, using autoencoders on our complex sets with sequences also had the added trait of simplifying and homogenizing the timestamped feature vector's representation to a reduced vector representation. This was helpful as the different sets of features could be combined freely in all possible combinations.

3.3.1 Variational Autoencoder

We used a deep variational autoencoder model to reduce the features to a more manageable number for the name set. The motivation for using a regularized model with a variational approach was to prevent overfitting and force the network to create meaningful connections between the features in latent space. Since our dataset's size was very limited and the fact that it was not possible to split the dataset into the typical train, validation and test sets, we had to tackle this problem in another way. We determined that not having the regularization in place would be a too naïve approach. The variational autoencoder was made in Python with Tensorflow Keras [37] Python library, a library for realizing and solving machine learning problems.

3.3.2 LSTM Autoencoder

For the two sequential datasets of operator and signal dependencies, we used a recurrent network model with long short-term memory (LSTM). This architecture is similar to the Seq2Seq technique often used in language translation [38].

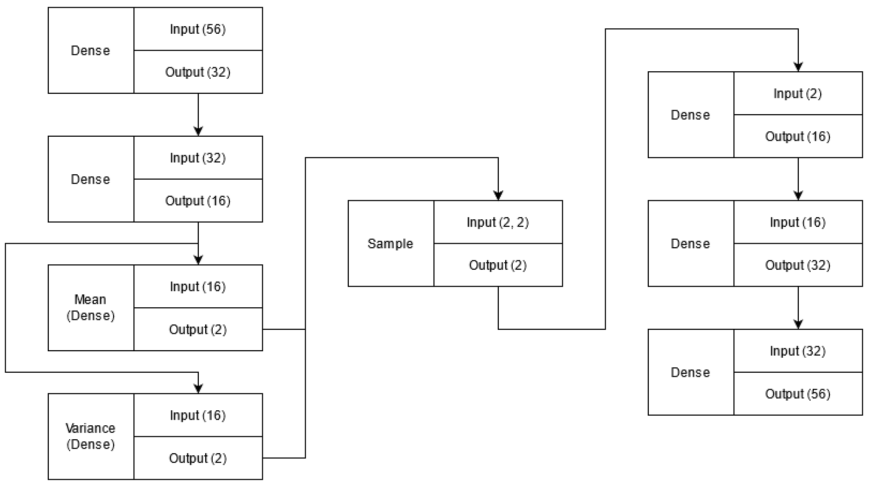

Figure 3.8: Layers in the Variational Autoencoder model

Our LSTM Autoencoder was trained to produce the input dependency sequence in reverse, as it has been proved to yield better learning results for Seq2Seq [38]. To avoid overfitting, we used a recurrent dropout regularization approach [39].

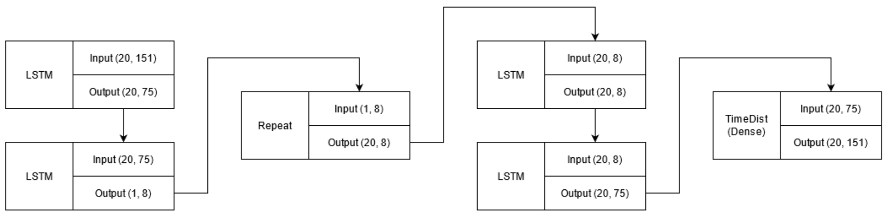

Figure 3.9: Layers used for operator dependencies

3.3.3 Handling Multiple Features

With the complex sets transformed into a simpler and homogeneous one-dimensional representation, they could be combined freely. The representations in the latent space were also normalized to give the same weight to each kind of feature. This is illustrated in figure 3.11 where a vector representation in latent space from each autoencoder is appended together with the simpler set, creating reduced feature representation for a data point.

As the last step in this section, we also performed some analysis with PCA. We looked at the cumulative explained variance ratio as a function of each com-

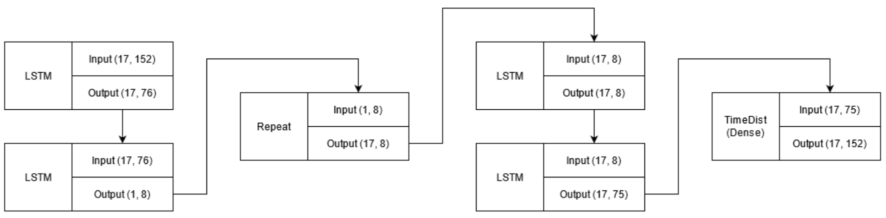

Figure 3.10: Layers used for signal dependencies

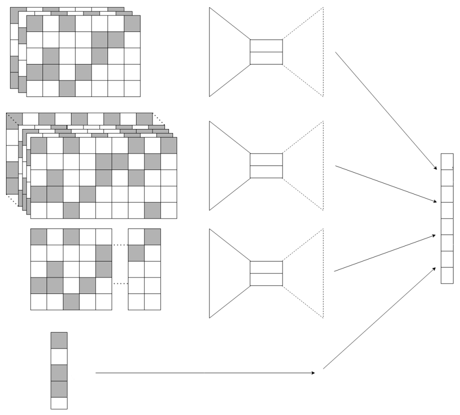

Figure 3.11: Each row in each matrix represent one data point, and each column represents one specific feature. The autoencoders are trained on their respective datasets, and the output in latent space that is combined is from only one data point.

bination's number of components. The plot can be seen in figure 3.12 and also in Appendix C. From these plots, we could conclude that a large amount of variance is contained within a smaller subset of components. We could thus, advantageously, use PCA on our datasets in the latent space to give a fairer comparison in the later evaluation with every combination having the same dimension. In all feature combinations, the PCA with five components were able to explain over 85% of the variance.

3.4 Further Dimensional Reduction

The reduced number of features present a lower feature dimension than before. However, the feature space was still too large and had to be reduced further to be visualized. The dimensionality reduction algorithm UMAP was used to reduce the number of features down to two, ideal for visualization. The implementation of UMAP that we used comes from the creators of the original paper [28] with the

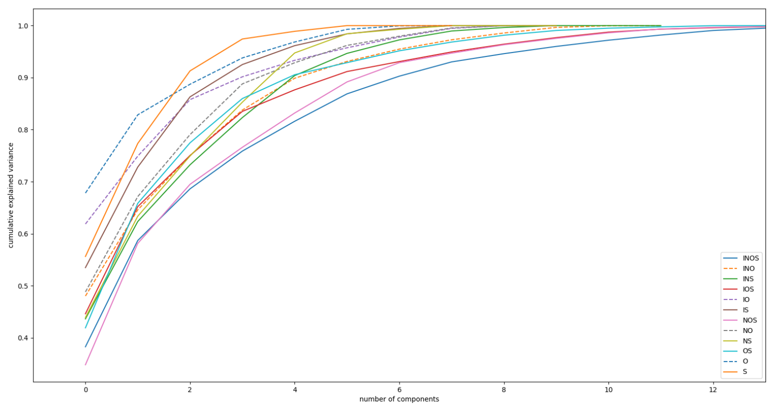

Figure 3.12: Cumulative explained variance ratio as a function of components.

implementation of their algorithm in Python.

3.5 Clustering

Any clustering algorithm that requires the number of clusters to be pre-defined beforehand is not recommended to use as the number of clusters is unknown. If a clustering algorithm needs a pre-defined number of clusters, then that information must come from another source. We chose to cluster with OPTICS and affinity propagation from *scikit-learns* [40]. We also clustered with k-means using the number of clusters found by OPTICS as k. We did this to examine if OPTICS characteristic to not cluster noise points affected the results or not.

Chapter 4
Result

4.1 Evaluation

To evaluate the clustering, we wanted to see if the clustering created well-defined clusters and if they contained signals with shared similarities. To measure the distinctness of the produced clusters, the two indexes Davies-Bouldin [31] and Silhouette [32] were selected. We chose to use these two indexes to measure how distinct and separated the different clusters were, with the motivation that more spaced-out and dense clusters will make the visualization better.

With the multiple site similarity index [34], we could see if the same tests had hit the same coverage points in a cluster. With this index, we could measure both the similarities within a cluster and the dissimilarity of other clusters.

4.1.1 Davies-Bouldin and Silhouette

Both Davies-Bouldin and Silhouette were used to obtain valuable information on the clustering. Neither algorithm needs any additional information about the data, as they are both internal cluster evaluation indexes. Silhouette rates the clusters with a score between -1 and 1, where a higher value represents a more distinct clustering. Davies-Bouldin, on the other hand, gives a score close to 0 on a well-separated clustering. We chose these two metrics as the problem is geared towards visualization, and having more distinct clusters makes the result of this project more palpable. We could not look at or validate the clusters in higher dimensions so these indexes were used as a score to confirm if the output was reasonable or if we had to perform further tweaking.

4.1.2 Multiple Site Similarity

While examining the clusters solely based on their structure and distinctiveness from each other is fairly straight forward, validating their actual data and retrieving information is not.

In most cases, the need for a *ground truth* or a *label* is inevitable to prove that the clusters contain data points that belong to each other and that they are not just placed close to each other.

In our case, we did not have any labels or categories of the clustered data points at hand. If we had, there would not have been a need for this thesis. Instead, we had to develop a way of verifying the similarities of our data points from an outside perspective.

Multi Label Problem

We created pseudo-labels by using the tests that hit each data point and hypothesized that similar tests hit similar data points, or more generally, that there will be a more significant overlap of data points hit by certain tests within the same cluster. This hypothesis can also be inverted when viewing and comparing the actual clusters, where the overlap instead should be kept low.

These pseudo labels set the stage for a multi-label problem, where each data point in the cluster could belong to multiple labels where each label represents one test. Conventional methods for evaluating classification based on multi-labels need to perform a comparison against a ground truth. This would not be a problem in a classification problem since the classification algorithm itself would assign the labels to each signal. On the other hand, clustering does not predict what labels a certain cluster represents, and thus we have to look elsewhere to evaluate the clusters based on labels.

As we want a high similarity among present tests in a cluster, we strove to calculate the similarity among all data points in a cluster. The Multiple site similarity, as explained in section 2.6.3, seeks to compare two or more sites against each other and see how similar they are to each other. We treat each data point in our cluster as a site with the covering tests acting as its species. We can then get a score for each cluster representing the intra-cluster similarities. We calculated an average score of the internal similarity for each cluster. The scores for all the individual clusters were then combined to get an average score of the clusters' internal similarities. We did this as we want the score of a cluster to be independent of the number of points in it, preventing one big cluster from dominating the score negatively or positively.

Inter Cluster Similarity

It is also interesting to compare how different the clusters are to each other, indicating if certain tests toggled only a small number of signals in few clusters or if they toggled all signals in every cluster. If the inter-cluster similarity is high, it may suggest that some of the tests managed to toggle a large number of the signals in a design. If the similarities between clusters are high, that may suggest that they could be merged into one cluster.

To calculate the inter-cluster similarity, we applied multiple site similarity on all clusters to determine how similar the found clusters are. To do this, we now represent a cluster as a site with its species. We used multi-class labels of the different sets of test combinations present in a cluster to represent the species, from which an inter-cluster similarity could be calculated when applying the multiple site similarity algorithm.

4.2 Evaluation Combinations

The clustering results depended heavily on the choice of both selected features and cluster algorithm. The four feature sets, seen in table 3.1 presents 15 different combinations of feature vectors that were evaluated for each clustering algorithm:

- OPTICS
- Affinity propagation
- k-means

A combined average score for MSS was calculated with the following formula:

$$MSS_{tot} = \frac{MSS_{Internal} + (1 - MSS_{External})}{2} \tag{4.1}$$

4.3 Evaluation Results

We evaluated our implementation on a design that had reached full toggle coverage, all signals had either been toggled or been excluded from toggling, making it possible to evaluate with our MSS approach. We extracted scores from three stages in our implementation:

- Directly on the latent space.
- After applying UMAP to reduce the features to two dimensions.
- Before UMAP but reduced feature space with PCA.

Complete tabular with all the experiments' scores can be found in appendixes A, B, and C.

4.3.1 Latent space

When clustering on the latent space from the autoencoders, we can observe that:

- The **O** feature set often produces a better result than **S** for all clustering algorithms, see table A.16, A.17 and A.18.
- Affinity propagation produces a greater number of clusters compared to other clustering algorithms, see table A.16.
- For OPTICS, in table A.18, the best score is 0.85 and is achieved with the features **INOS**, closely followed by **IO** and **OS**. Notably, **OS** produces better results when combined instead of just **S** or **O** features which differs from Affinity and k-means in A.16 and A.17.
- Using the feature set **I** more often than not gave a better result than not using it. As seen in A.16, A.17, A.18.
- Using the feature set **N**, more often than not, actually produced worse results than leaving it out with affinity A.16 and k-means A.17. The same does not apply to OPTICS A.18.
- Overall, clustering with OPTICS produced the best result, as seen both with internal and external MSS score.

4.3.2 After UMAP

Moreover, when using autoencoder and clustering after applying UMAP for dimensionality reduction, we can observe:

- When using affinity propagation, see table B.16, the feature combination **INO**, **IS**, and **IO** all produce high MSS scores. With DB and silhouette scores in mind, **IO** gives the best result.
- The features **O** and **S** should not be combined as a feature. See table B.17 with k-means and feature combination **IOS**, **OS**, **S**, and **O**. In all cases **S** and **O** produce better results when they are not combined. We can see that there are better results if the **I** feature is combined with another feature combination and must be included to get a good score. With the above two observations in mind, we can see that only six combinations are relevant, **IO**, **IS**, **I**, **INO**, **INS**, and **IN**. From these six feature combinations, we can see that solely using the **I** feature will net the best score for DB and silhouette and give a good MSS score, but produces fewer clusters.
- The feature **I** gives the best MSS score overall when using OPTICS, see table B.18, and produces the lowest number of clusters, 4, compared to the average of 8-9 clusters. The second best one is **IN**, followed by **IO**, **IS**, and **INS**.
- All feature combinations produced better results when paired with the **I** set. This is true for all cluster algorithms B.16, B.17 and B.18.
- The feature set **N** produced worse results paired with other combinations than when it was left out for all different cluster algorithms B.17, B.17 and B.18.

4.3.3 After PCA

By homogenizing the latent space features with PCA, we can observe that:

- With using affinity propagation we observe a large number of clusters compared to the other experiments, see table C.16. The scores lack the external MSS score that can be explained by the high number of clusters.
- We can see that the cluster scores for k-means clustering in table C.17 have better scores for Davies Bouldin and Silhouette than OPTICS (see table C.18). However, the MSS scores show better results when using OPTICS.
- The highest total MSS score was reached with the **INO** feature combination when using PCA and clustering with OPTICS.
- With all cluster algorithms, including the feature set **I** in the combinations almost always produced a better result.
- Using the **N** features produced a worse result in some combinations with Affinity and k-means but produced a better result with OPTICS. Overall, the performance drop caused by the **N** set were lower when PCA was applied.

- Using both **O** and **S** together often gave a performance gain compared to just using one of them in the combinations. This differs from the previous result where clustering algorithms were applied without PCA directly on the latent space, with and without UMAP.
- The **N** feature contributed to the spread of the features. Figure C.12 shows only the **N** feature and how all data points are scattered around. Compared to C.11 that contains small isolated clusters in distinct corners.

Overall, we observe that the **I** feature played a vital role in a signal's characteristics as they appear at the top for all three feature combinations in all three result sets in appendix A, B, and C. There is no clear winner with an excellent MSS that also gives good Davies Bouldin and silhouette indices. From figures A.1 to A.15 and figures C.1 to C.15, we can see that the visualization for the different feature combinations is, as expected, quite different. Some feature combinations give distinct clusters that the human eye can easily distinguish, see figure C.6 for the **IO** features, while the best MSS score is given by the **INO** combination in table C.18, see figure C.2, which has a more evenly spread out distribution of data points. From the figures, we also note that the clustering algorithms found clusters that were relatively close to each other as in figure C.1, C.2, and C.3, thus reducing the effectiveness and relevance of both the Davies-Bouldin and silhouette scores.

Chapter 5

Discussion and Conclusions

We can conclude from the recorded cluster evaluation that some features were better to use for clustering than others. If we consider only the clustering evaluated with the MSS score, then **INO** using Optics and PCA is the best method to cluster together coverage holes for hole analysis, see table D.18. This method outperformed the other feature combination and clustering algorithms. Moreover, from the visualization in figure D.2 we can conclude optics were able to find possible clusters inside the two separated groups of points. We believe this combination may be the best one. However, there needs to be further experiments on other designs of various sizes to see if the result differs. The name feature on its own did not provide any good result for the clustering. We studied the name feature because it is one of the few information sources directly available in the coverage database. Evaluating if the name feature is good or not all comes down to the design that is to be clustered. An RTL design can have utterly unique, nonsensical names and still synthesize to a netlist. Therefore, the value of the name features is questionable. It all depends on a design philosophy with coherent and strict coding guidelines, as a deviation to them could make the clustering start to identify other coding patterns. For example if different designers are working on a project.

An interesting observation from our experiments is that PCA managed to remove noise from the latent space when combining both the **O** and **S** features. The reason might be that the underlying features these sets describe are very similar, and combining them does not provide any new information, only raise the number of dimensions. Applying PCA on the latent space provided a boost to the clustering result and should be investigated more as a final dimensionality reduction method. Using UMAP to produce clusters and identify coverage holes has not proved to be sufficient from a pure visualization standpoint. The results are vastly different depending on the features selected and the randomness of the algorithms. We could get completely different images from a set of latent space features, and as mentioned by UMAPs authors, this is normal for the algorithm. However, the nature of UMAP could lead to clusters tearing apart, and in some cases, even the creation of new imaginary clusters. These issues are one of the caveats with trying to find a reduced dimensional representation. All figures in appendix A, B and C are just one variation of possibly endless two-dimensional representations that could be generated from UMAP, and the perfect visualization will always be unknown. The clustering directly on the latent space or after PCA reduction does not have the same issues as they preserve density and do not depend on the randomness of the algorithm. However, they fail to deliver a two-dimensional visualizable image. During the late stages of this thesis, we had a thorough discussion about the goal of the visualization aspect of the prototype. In the beginning, we had the idea that a graphical representation could aid a verification engineer in structuring the work left to close coverage by pinpointing holes that should be analyzed together or even prioritize what to focus on first. For example, the graphical representation could be read left to right or larger clusters first followed by the smaller ones. In retrospective, we are not sure that this is the most effective way of providing helpful input. A simple priority list with sizes of the clusters, their members and anomalies would probably suffice in achieving this, hence removing the problem of further dimensionality reduction completely.

5.1 Experiences

We would like to briefly talk about our experiences encountered during our work on this thesis, the problems we have faced and possible solutions we have not been able to reach.

5.1.1 Creating a Dataset

We tackled the problem as a cluster-based problem at first, that machine learning should be the core in discovering the connection between uncovered coverage holes. What we discovered during our work on this thesis was the difficulty of accurately representing the connection between two different signals in the same circuit. Should the connection be drawn from the netlist, the RTL, extract data before, during or after simulations. This made it hard to construct a meaningful dataset that we thought could be used to accurately represent the relations in a circuit. We strongly suggest doing future work examining the best possible way of representing similarities and connections in digital circuits for machine learning problems. One suggestion is to try Graph Neural Networks (GNN) [41].

When we started to look at the toggle coverage, we hypothesized that this would also be the most straightforward coverage metric to cluster. Though basing the model solely on the available coverage database, the name, signal size, and index position would not be enough information to cluster the toggle coverage holes. When we started extracting the signal and operational dependencies from the netlist, we had to condense the information and reduce it to a more manageable size for the clustering algorithms. Using an autoencoder was not a certainty when we started looking at this problem. Instead, we started using autoencoders during our work with this project as they solved multiple issues, e.g. reducing the number of features and provided a numerical representation of categorical features. It could be interesting to investigate a similar model with autoencoders on line, condition or branch coverage with a similar approach. Such a method could theoretically be able to gather information from nearby lines and statements. Consider a "nested if" statement that may have prevented one part of the code from execution and thus not yielding either condition, toggle or code coverage for that branch. It is probably easier to capture why a branch, condition or line metric is uncovered by looking at the surrounding environment rather than looking for general toggle coverage points.

The LSTM autoencoders that we used do not capture any branching and the performance could potentially be enhanced by upgrading it to a variational graph autoencoder [42], thus capturing more of the depth of the connections in a netlist.

5.1.2 Creating a General Clustering Solution

One of the core features that we strove to achieve with our implementation was simplicity to use and to quickly give verification engineers an overview of the current closure status without too much work. By simplicity, we mean that a person should be able to use the tool and find it helpful without being an expert in both machine learning and hardware verification. In a way, both fields are an art as well as a science. Creating good models or analyzing and closing coverage requires experience and extensive knowledge about the domain in question. In the beginning, we found it hard to know what kind of features we could extract and even harder to determine which of them could be interesting. With the autoencoder concept, we attempted to overcome this barrier and avoid the need to become proficient in another field outside our usual work area.

We initially thought and later confirmed that there had to be a considerable amount of hyperparameter tuning to achieve a good result with the clustering algorithms and the overall model. Even getting a result in the first place required a significant chunk of fine-tuning for our specific dataset. We have proven that there is some underlying connection between the coverage points that could be captured and highlighted what type of features worked out for us. However, since we have only proven this for one particular design, which required substantial extra work, it is safe to say that we were unable to create a generalized and easy-to-use solution. Integrating this into a verifications engineers everyday toolbox is not worthwile yet. At the end of this section, we will discuss this further and propose alternative solutions.

5.1.3 Actual Categorisation

At the end of the thesis, we came over more detailed information for how the verification progress proceeded. We got to know better how these problems where analyzed and grouped them up into three categories.

1. Coverage holes that have been excluded. Holes that a designer do not care for. The signal might be from a configuration register and will not change in the current design configuration. It might be a combination of signals that can never happen in the design.

2. Coverage holes that rely on more targeted testing. The random testing generator might not have been able to fully cover what it is intended to do. Therefore, the number of test iterations might need to be increased to hit all coverage holes.

3. Coverage holes that require new tests to be made. A completely new test might be needed with a specific configuration to target one part of the design.

Our goal was to aid the verification engineer with identifying and solving these root causes simultaneously. We had seen before that [2] with their Coverage Driven test Generation could use ML to help close coverage holes in the second point before even reaching the stage we analyzed them at, meaning that by employing CDG the effort could theoretically be focused on fixing the first and third points instead.

Formal tools are often used to accelerate coverage closure. These tools builds mathematical models to prove that certain coverage points are impossible to cover and should be excluded. The tools very powerful but not perfect and may not be able to prove all the coverage points. The verification engineer would analyze the code and realize that the automatically excluded signals might be connected to other uncovered coverage points further down into the design. These coverage points can also be excluded. Our implementation could potentially aid with this analysis, since it could be easy to identify the uncovered coverage holes missed by the formal coverage analysis. Even more static analysis of this problem should be done to further enhance the automatic exclusion process and give a more complete answer.

We believe the third point is what the next version of this kind of tool should be more focused on. Here the similarities between signals and coverage bins would play a central part in reducing the number of new tests that have to be made.

References

[1] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[2] C. Ioannides and K. I. Eder, "Coverage-directed test generation automated by machine learning – a review," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, Jan. 2012.

[3] "Uvm introduction." `https://www.chipverify.com/uvm/uvm-introduction`. Accessed: 2020-10-31.

[4] D. Araiza-Illan, D. Western, A. Pipe, and K. Eder, "Coverage-driven verification —," in *Hardware and Software: Verification and Testing* (N. Piterman, ed.), (Cham), pp. 69–84, Springer International Publishing, 2015.

[5] E. E. Mandouh, A. Salem, M. Amer, and A. G. Wassal, "Cross-product functional coverage analysis using machine learning clustering techniques," in *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, pp. 1–2, 2018.

[6] A. Wahba, J. Hohnerlein, and F. Rahman, "Expediting design bug discovery in regressions of x86 processors using machine learning," in *2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV)*, pp. 1–6, Dec 2019.

[7] D. Maksimovic, A. Veneris, and Z. Poulos, "Clustering-based revision debug in regression verification," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pp. 32–37, 2015.

[8] M. Golagha, C. Lehnhoff, A. Pretschner, and H. Ilmberger, "Failure clustering without coverage," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, (New York, NY, USA), p. 134–145, Association for Computing Machinery, 2019.

[9] X. Shi, D. Zeng, Y. Hu, G. Lin, and O. Zaïane, "Enhancement of incremental design for fpgas using circuit similarity," in *2011 12th International Symposium on Quality Electronic Design*, pp. 1 – 8, 04 2011.

[10] K. Zeng, *An Exploration of Circuit Similarity for Discovering and Predicting Reusable Hardware.* PhD thesis, Virginia Tech, 2016.

[11] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, (Berkeley, Calif.), pp. 281–297, University of California Press, 1967.

[12] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231, AAAI Press, 1996.

[14] P. H. Ahmad and S. Dang, "Performance evaluation of clustering algorithm using different datasets," *International Journal of Advance Research in Computer Science and Management Studies*, vol. 3, pp. 167–173, 2015.

[15] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, (New York, NY, USA), p. 49–60, Association for Computing Machinery, 1999.

[16] M. Steinbach, L. Ertöz, and V. Kumar, "The challenges of clustering high dimensional data," in *New directions in statistical physics*, pp. 273–309, Springer, 2004.

[17] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012.

[18] M. H. C. Law, M. A. T. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[19] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy, "Relational autoencoder for feature extraction," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 364–371, 2017.

[20] K. Y. Yeung and W. L. Ruzzo, "An empirical study on principal component analysis for clustering gene expression data," *Bioinformatics*, vol. 17, no. 9, pp. 763–774, 2001.

[21] E. Lin, S. Mukherjee, and S. Kannan, "A deep adversarial variational autoencoder model for dimensionality reduction in single-cell rna sequencing analysis," *BMC bioinformatics*, vol. 21, no. 1, pp. 1–11, 2020.

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[23] M. S. Mahmud and X. Fu, "Unsupervised classification of high-dimension and low-sample data with variational autoencoder based dimensionality reduction," in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 498–503, 2019.

[24] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” 2016.

[25] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[26] L. McInnes, J. Healy, and J. Melville, “Umap: Uniform manifold approximation and projection for dimension reduction,” 2018.

[27] L. McInnes, “Umap uniform manifold approximation and projection for dimension reduction,” 2018. Scipy2018, `https://www.youtube.com/watch?v=nq6iPZVUxZU`.

[28] L. McInnes, J. Healy, N. Saul, and L. Grossberger, “Umap: Uniform manifold approximation and projection,” *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[29] “Normalization,” Oct 2 2020. Accessed on: Nov 23-2020. [Online]. Available: https://developers.google.com/machine-learning/data-prep/transform/normalization.

[30] W. Shao, F. Salim, A. Song, and A. Bouguettaya, “Clustering big spatiotemporal-interval data,” *IEEE Transactions on Big Data*, vol. 2, pp. 190 – 203, 07 2016.

[31] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979.

[32] P. J. Rousseeuw, “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,” *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[33] L. Kaufman, P. Leonard Kaufman, and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* A Wiley-Interscience publication, John Wiley & Sons, 1990.

[34] T. Sørensen, *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons.* Biologiske skrifter, I kommission hos E. Munksgaard, 1948.

[35] O. H. Diserud and F. Ødegaard, “A multiple-site similarity measure,” *Biology letters*, vol. 3, no. 1, pp. 20–22, 2007.

[36] A. Sarveniazi, “An actual survey of dimensionality reduction,” *American Journal of Computational Mathematics*, vol. 04, pp. 55–72, 01 2014.

[37] F. Chollet *et al.*, “Keras.” `https://keras.io`, 2015.

[38] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, pp. 3104–3112, Curran Associates, Inc., 2014.

[39] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," *arXiv preprint arXiv:1603.05118*, 2016.

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

[41] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2021.

[42] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

Appendix A

Latent Space

IOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	14	0.72	0.47	0.84	0.81	0.48
k-means	6	0.87	0.47	0.67	0.83	0.58
OPTICS	6	1.76	0.13	0.52	0.79	0.64

Table A.1: Scores for the feature combination IOS.

IO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	12	0.64	0.52	0.79	0.85	0.53
k-means	4	0.92	0.51	0.64	0.90	0.63
OPTICS	4	1.26	0.19	0.34	0.90	0.78

Table A.2: Scores for the feature combination IO.

IS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	11	0.82	0.46	0.80	0.84	0.52
k-means	7	0.77	0.54	0.73	0.86	0.57
OPTICS	7	1.86	0.13	0.71	0.78	0.53

Table A.3: Scores for the feature combination IS.

I	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	0	N/A	N/A	N/A	N/A	N/A
k-means	3	0.39	0.90	0.35	0.94	0.79
OPTICS	3	0.44	0.97	0.35	0.94	0.80

Table A.4: Scores for the feature combination I.

OS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	17	0.52	0.52	0.87	0.66	0.39
k-means	5	0.84	0.50	0.76	0.81	0.53
OPTICS	5	2.04	0.05	0.27	0.81	0.77

Table A.5: Scores for the feature combination OS.

O	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	0.48	0.59	0.83	0.77	0.47
k-means	4	0.60	0.61	0.73	0.82	0.55
OPTICS	4	1.41	0.15	0.67	0.61	0.47

Table A.6: Scores for the feature combination O.

S	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	11	0.50	0.57	0.84	0.71	0.44
k-means	6	0.73	0.50	0.81	0.74	0.47
OPTICS	6	1.83	0.06	0.57	0.78	0.60

Table A.7: Scores for the feature combination S.

INOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	17	0.82	0.38	0.86	0.77	0.45
k-means	4	1.09	0.38	0.70	0.82	0.56
OPTICS	4	2.16	-0.05	0.20	0.91	0.85

Table A.8: Scores for the feature combination INOS.

INO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	16	0.92	0.30	0.83	0.80	0.48
k-means	4	1.25	0.33	0.64	0.90	0.63
OPTICS	4	1.72	-0.11	0.14	0.69	0.77

Table A.9: Scores for the feature combination INO.

INS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	0.84	0.39	0.81	0.82	0.51
k-means	6	0.96	0.36	0.63	0.89	0.63
OPTICS	6	1.93	0.04	0.65	0.80	0.58

Table A.10: Scores for the feature combination INS.

IN	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	9	0.95	0.34	0.65	0.89	0.62
k-means	5	0.84	0.44	0.53	0.91	0.69
OPTICS	5	1.67	-0.06	0.65	0.91	0.63

Table A.11: Scores for the feature combination IN.

NOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	18	0.62	0.44	0.88	0.61	0.37
k-means	4	1.07	0.37	0.72	0.82	0.55
OPTICS	4	2.44	-0.12	0.29	0.92	0.81

Table A.12: Scores for the feature combination NOS.

NO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	15	0.71	0.35	0.85	0.72	0.43
k-means	4	1.20	0.28	0.77	0.76	0.49
OPTICS	4	1.78	-0.11	0.40	0.54	0.57

Table A.13: Scores for the feature combination NO.

NS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	15	0.73	0.46	0.84	0.72	0.44
k-means	6	0.95	0.31	0.72	0.74	0.51
OPTICS	6	1.84	0.03	0.76	0.85	0.55

Table A.14: Scores for the feature combination NS.

N	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	9	0.72	0.42	0.79	0.78	0.50
k-means	6	0.74	0.41	0.75	0.76	0.51
OPTICS	6	1.32	0.03	0.76	0.70	0.47

Table A.15: Scores for the feature combination N.

Affinity	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	14	0.72	0.47	0.84	0.81	0.48
IO	12	0.64	0.52	0.79	0.85	0.53
IS	11	0.82	0.46	0.80	0.84	0.52
I	0	N/A	N/A	N/A	N/A	N/A
OS	17	0.52	0.52	0.87	0.66	0.39
O	13	0.48	0.59	0.83	0.77	0.47
S	11	0.50	0.57	0.84	0.71	0.44
INOS	17	0.82	0.38	0.86	0.77	0.45
INO	16	0.92	0.30	0.83	0.80	0.48
INS	13	0.84	0.39	0.81	0.82	0.51
IN	9	0.95	0.34	0.65	0.89	0.62
NOS	18	0.62	0.44	0.88	0.61	0.37
NO	15	0.71	0.35	0.85	0.72	0.43
NS	15	0.73	0.46	0.84	0.72	0.44
N	9	0.72	0.42	0.79	0.78	0.50

Table A.16: Scores for Affinity and all feature combinations.

k-means	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	0.87	0.47	0.67	0.83	0.58
IO	4	0.92	0.51	0.64	0.90	0.63
IS	7	0.77	0.54	0.73	0.86	0.57
I	3	0.39	0.90	0.35	0.94	0.79
OS	5	0.84	0.50	0.76	0.81	0.53
O	4	0.60	0.61	0.73	0.82	0.55
S	6	0.73	0.50	0.81	0.74	0.47
INOS	4	1.09	0.38	0.70	0.82	0.56
INO	4	1.25	0.33	0.64	0.90	0.63
INS	6	0.96	0.36	0.63	0.89	0.63
IN	5	0.84	0.44	0.53	0.91	0.69
NOS	4	1.07	0.37	0.72	0.82	0.55
NO	4	1.20	0.28	0.77	0.76	0.49
NS	6	0.95	0.31	0.72	0.74	0.51
N	6	0.74	0.41	0.75	0.76	0.51

Table A.17: Scores for k-means and all feature combinations.

OPTICS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	1.76	0.13	0.52	0.79	0.64
IO	4	1.26	0.19	0.34	0.90	0.78
IS	7	1.86	0.13	0.71	0.78	0.53
I	3	0.44	0.97	0.35	0.94	0.80
OS	5	2.04	0.05	0.27	0.81	0.77
O	4	1.41	0.15	0.67	0.61	0.47
S	6	1.83	0.06	0.57	0.78	0.60
INOS	4	2.16	-0.05	0.20	0.91	0.85
INO	4	1.72	-0.11	0.14	0.69	0.77
INS	6	1.93	0.04	0.65	0.80	0.58
IN	5	1.67	-0.06	0.65	0.91	0.63
NOS	4	2.44	-0.12	0.29	0.92	0.81
NO	4	1.78	-0.11	0.40	0.54	0.57
NS	6	1.84	0.03	0.76	0.85	0.55
N	6	1.32	0.03	0.76	0.70	0.47

Table A.18: Scores for OPTICS and all feature combinations.

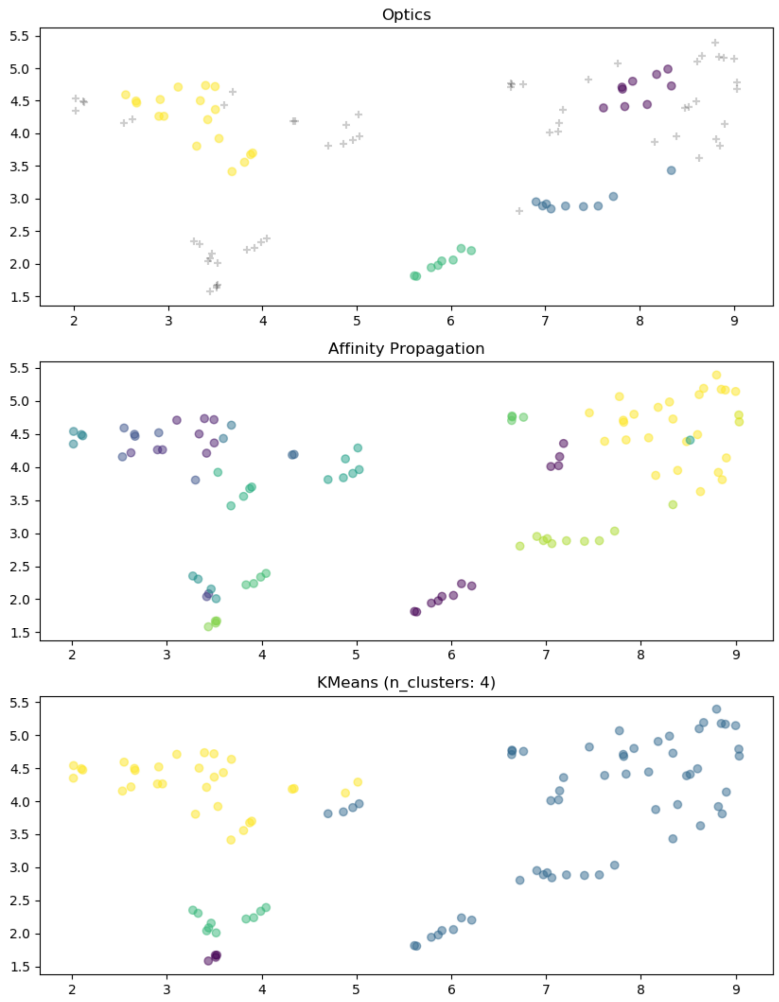

Figure A.1: Visualization for feature combination INOS.

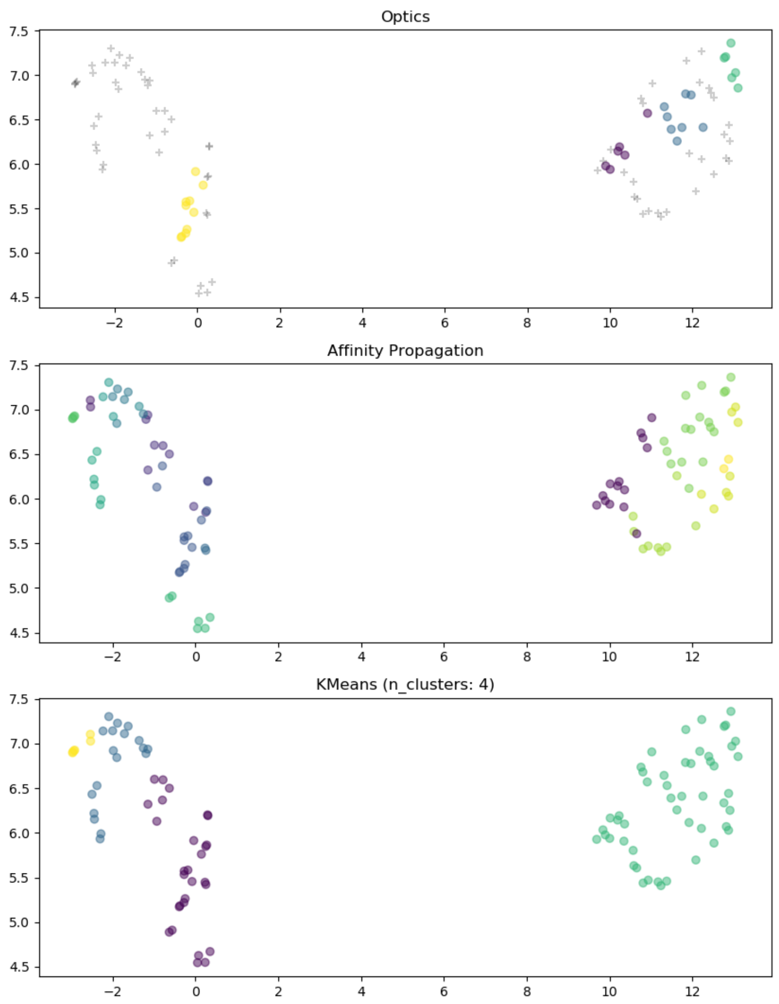

Figure A.2: Visualization for feature combination INO.

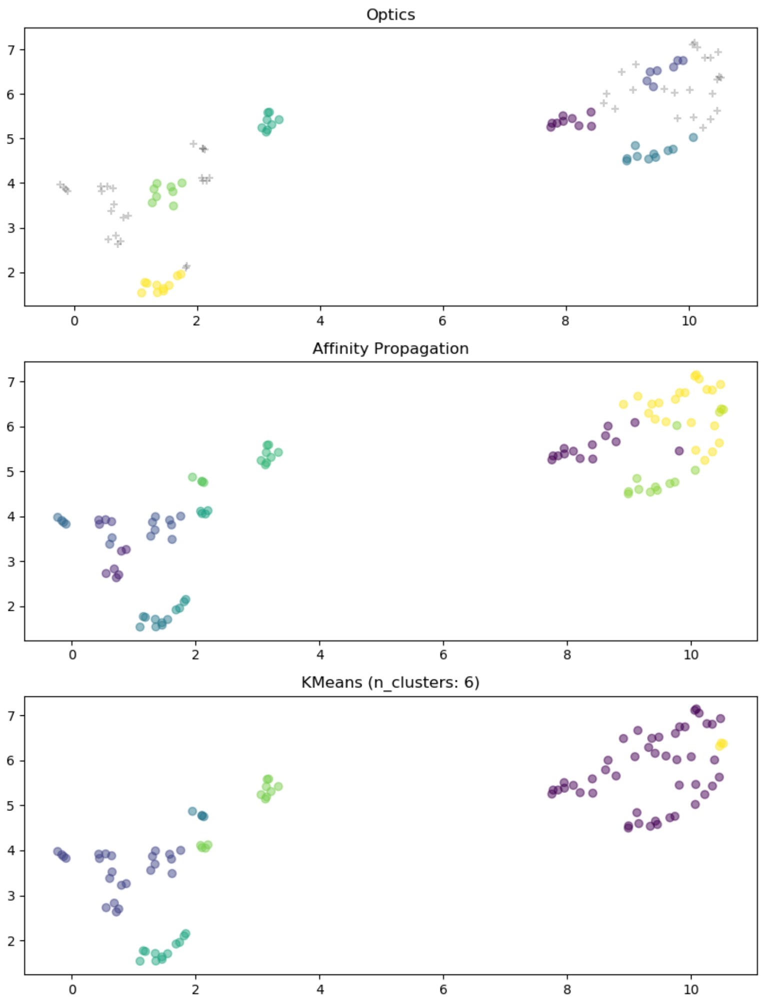

Figure A.3: Visualization for feature combination INS.

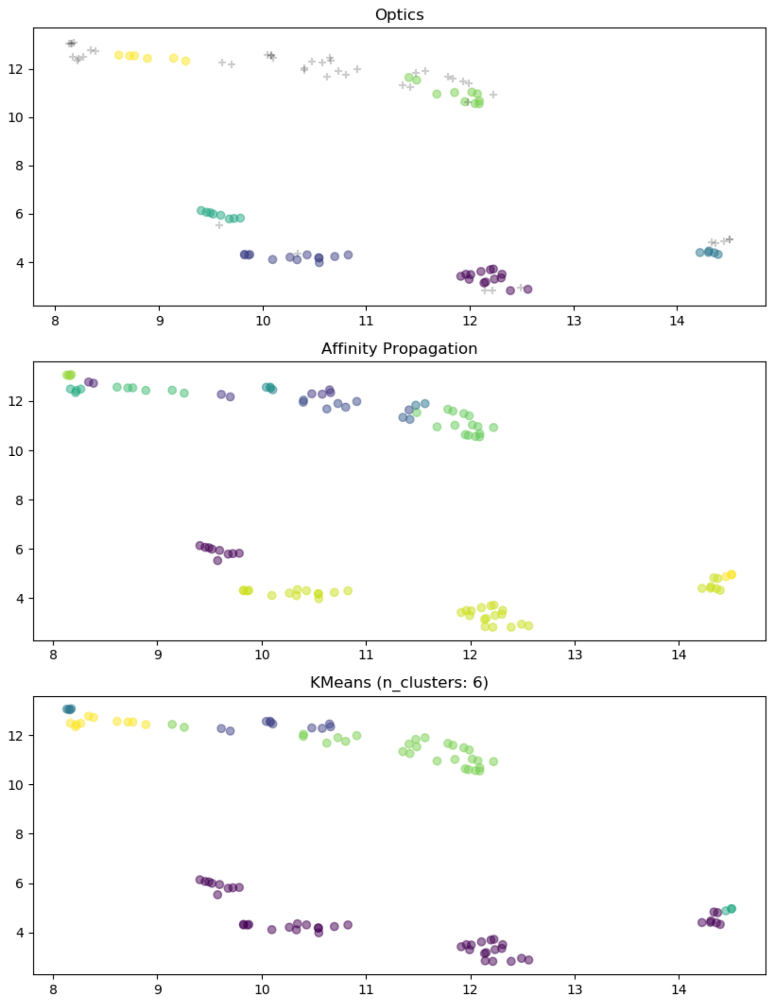

Figure A.4: Visualization for feature combination IOS.

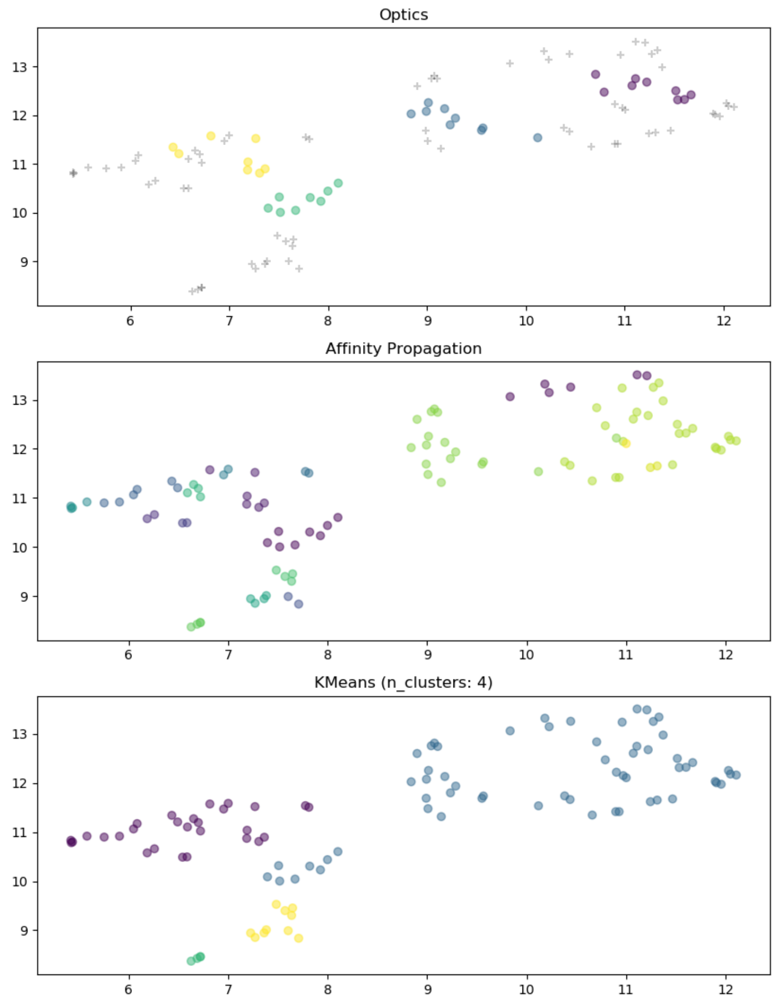

Figure A.5: Visualization for feature combination NOS.

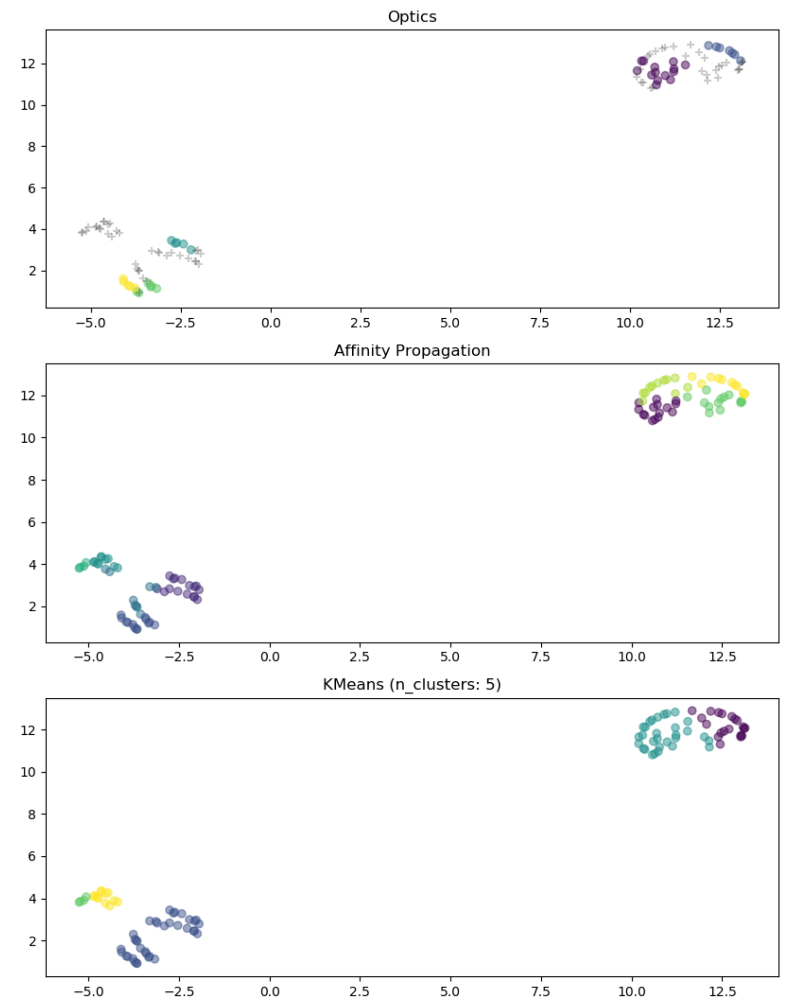

Figure A.6: Visualization for feature combination IN.

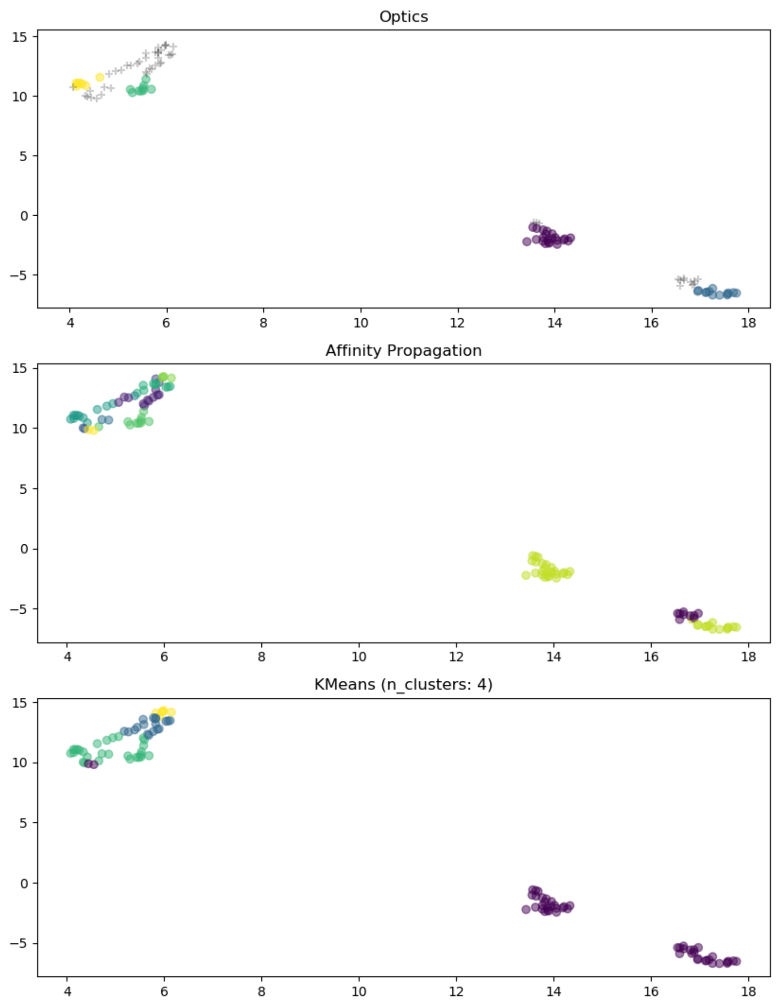

Figure A.7: Visualization for feature combination IO.

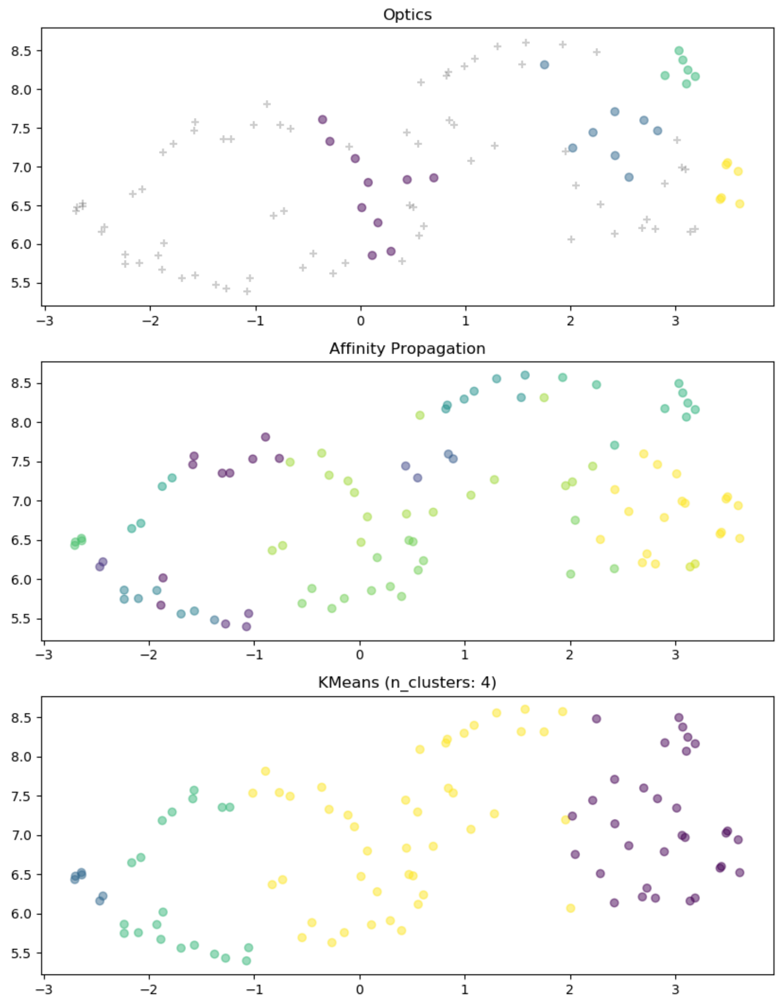

Figure A.8: Visualization for feature combination NO.

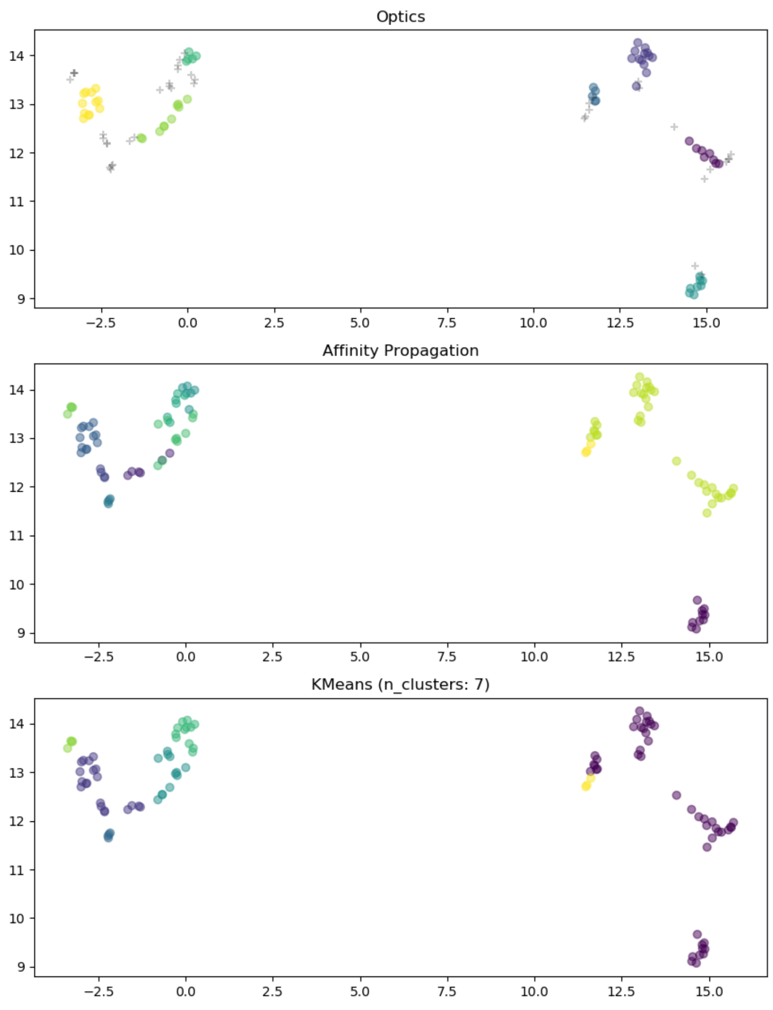

Figure A.9: Visualization for feature combination IS.

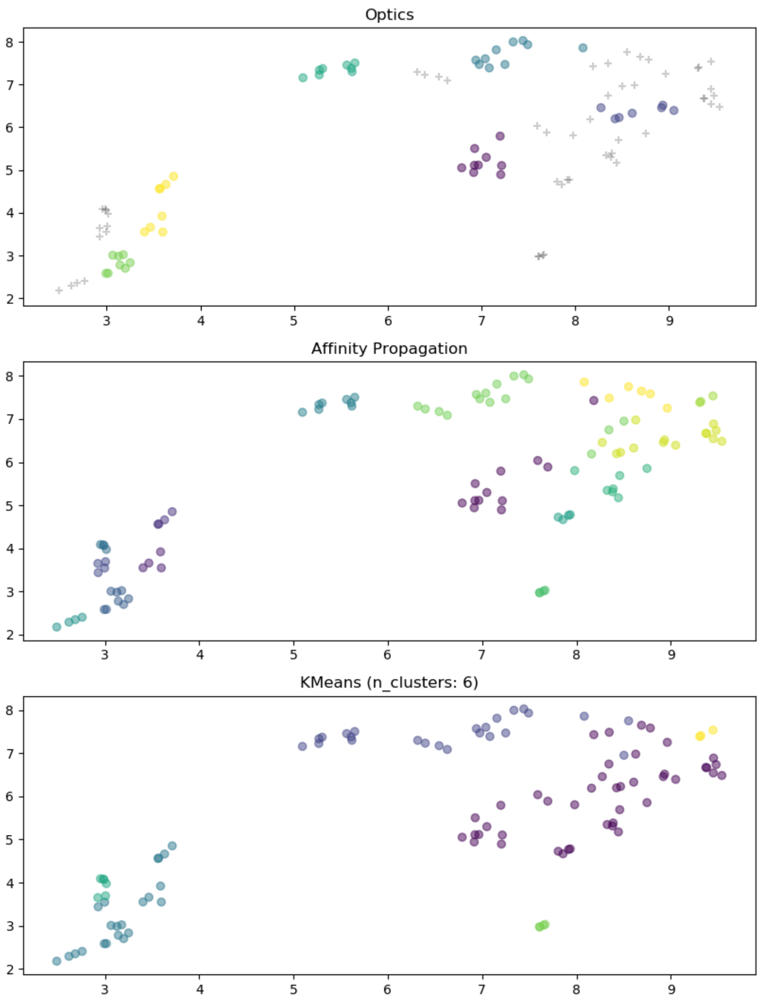

Figure A.10: Visualization for feature combination NS.

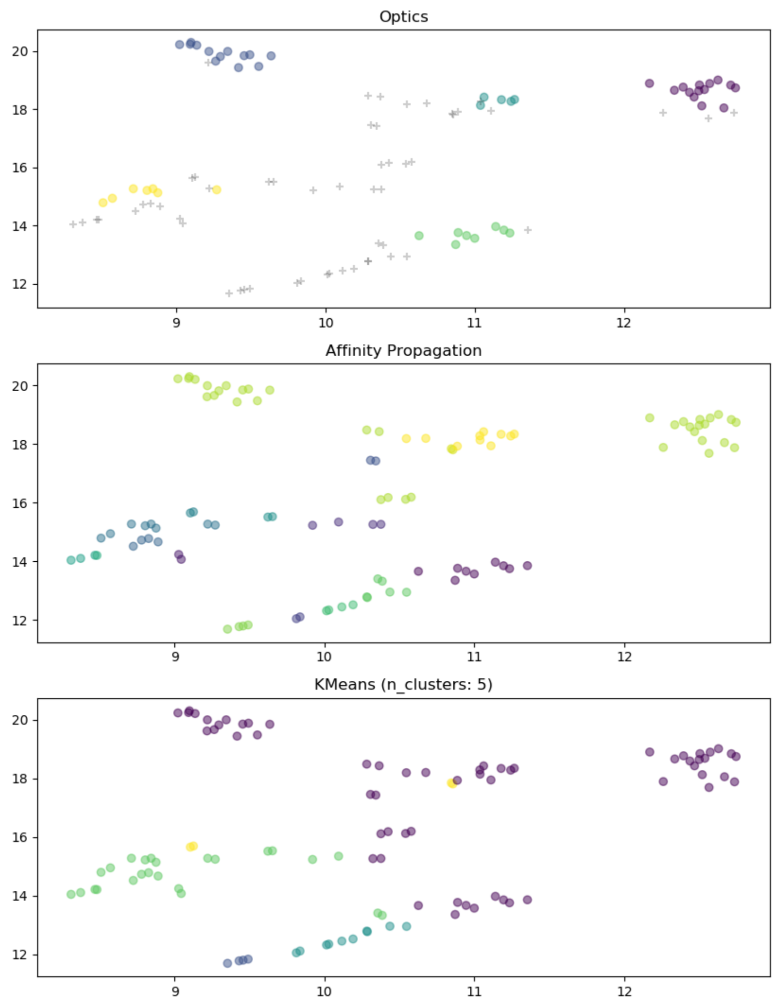

Figure A.11: Visualization for feature combination OS.

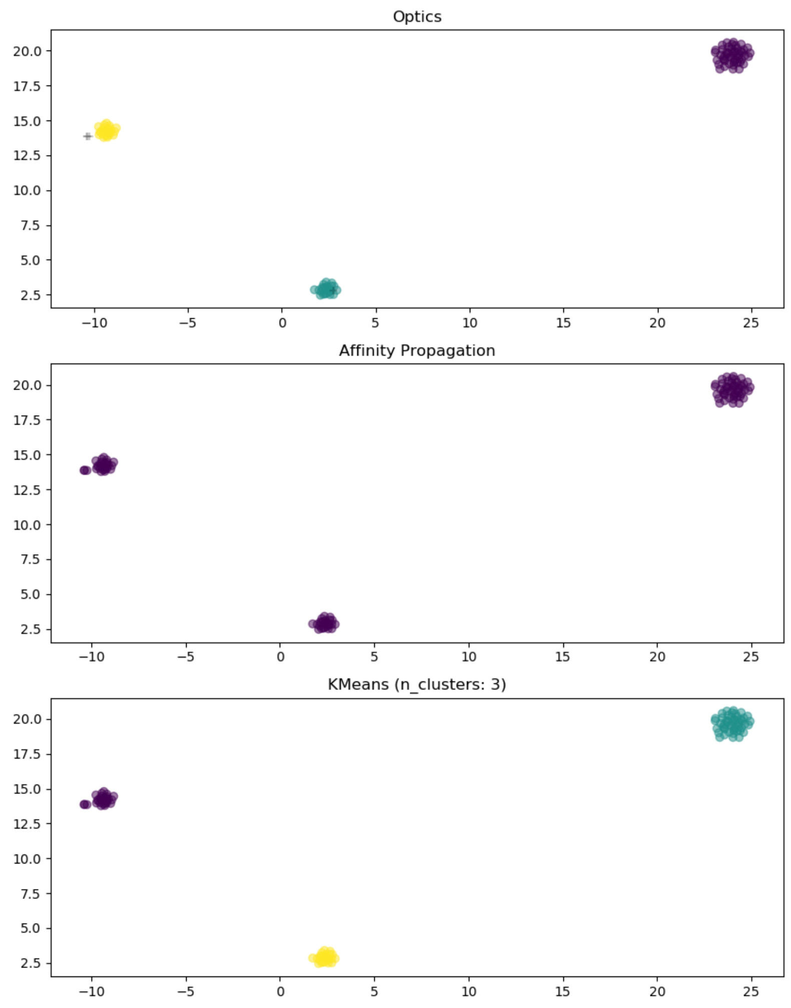

Figure A.12: Visualization for feature combination I.

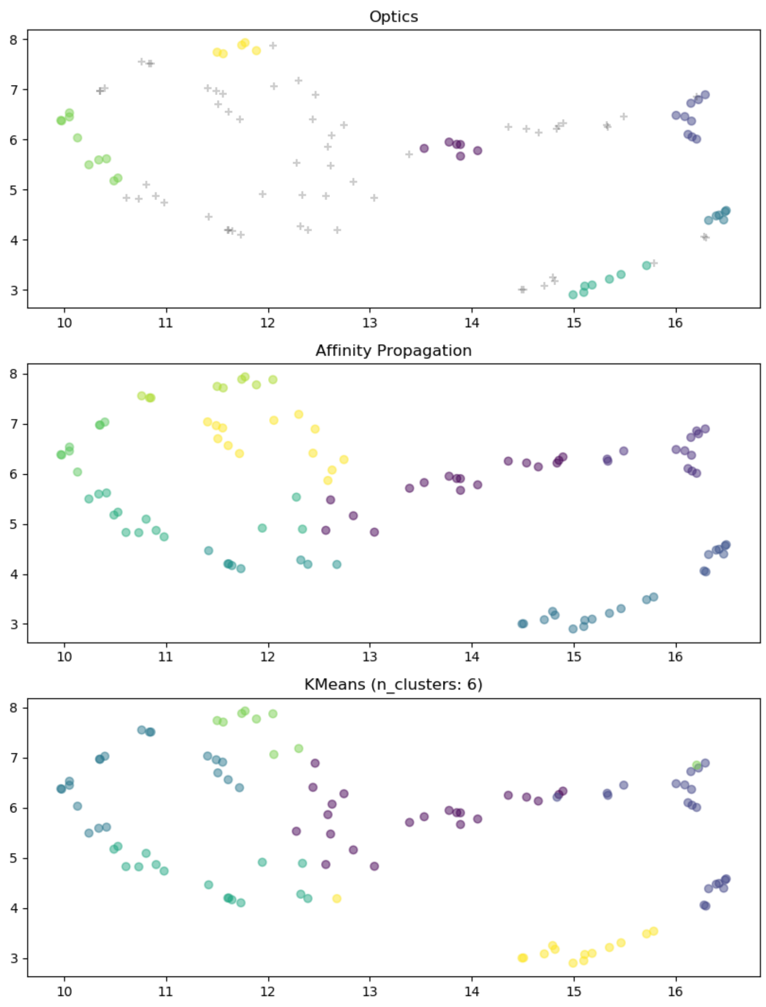

Figure A.13: Visualization for feature combination N.

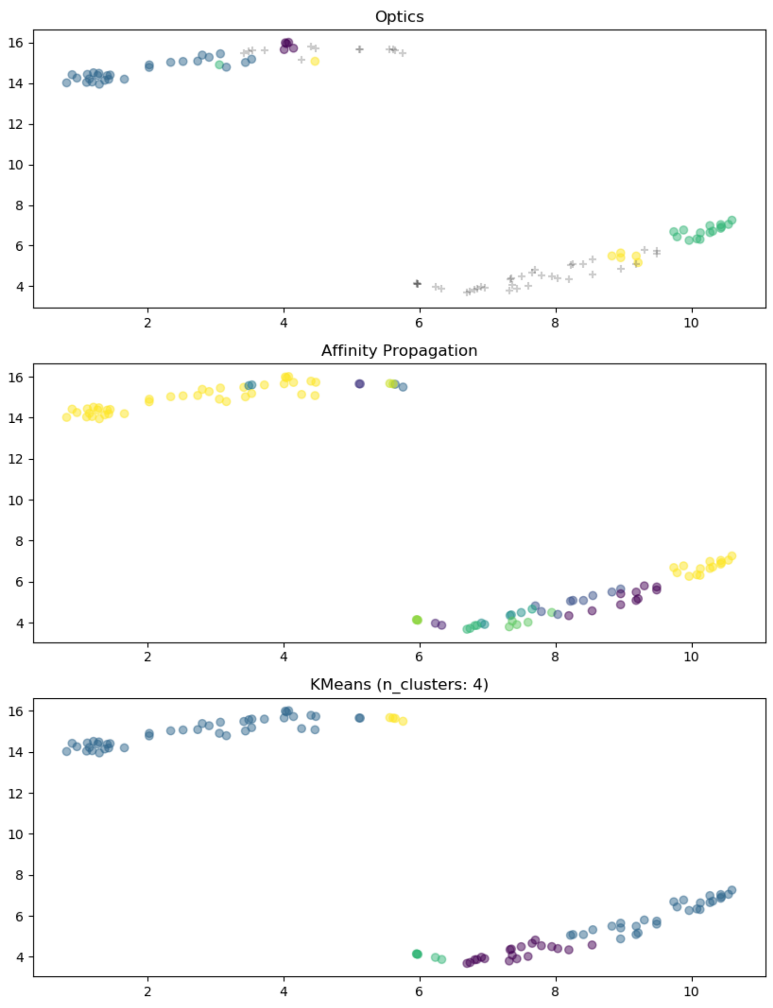

Figure A.14: Visualization for feature combination O.

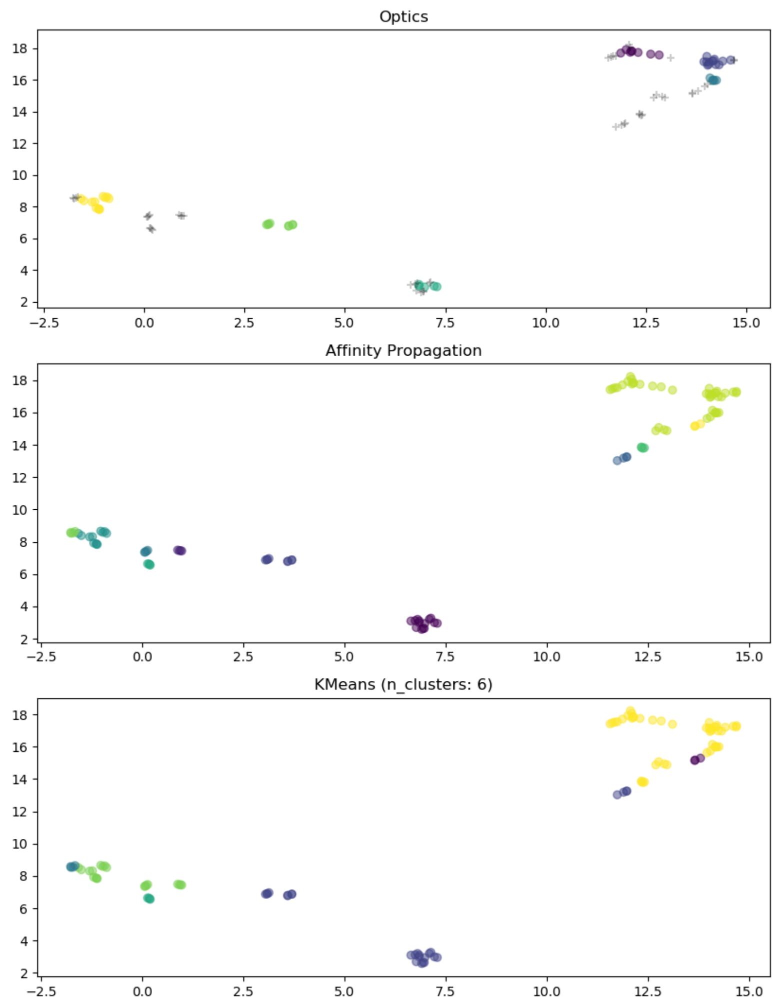

Figure A.15: Visualization for feature combination S.

Appendix B

UMAP

Table B.1: Scores for the feature combination IOS.

IOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	6	0.44	0.67	0.68	0.86	0.59
k-means	6	0.44	0.67	0.68	0.86	0.59
OPTICS	9	1.22	0.44	0.72	0.76	0.52

Table B.2: Scores for the feature combination IO.

IO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	4	0.37	0.73	0.54	0.90	0.68
k-means	4	0.37	0.73	0.54	0.90	0.68
OPTICS	9	1.20	0.29	0.71	0.85	0.57

Table B.3: Scores for the feature combination IS.

IS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	5	0.43	0.68	0.53	0.91	0.69
k-means	5	0.43	0.68	0.53	0.91	0.69
OPTICS	8	0.97	0.60	0.74	0.85	0.55

Table B.4: Scores for the feature combination I.

I	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	0	N/A	N/A	N/A	N/A	N/A
k-means	3	0.04	0.97	0.35	0.94	0.79
OPTICS	4	1.05	-0.13	0.34	0.85	0.75

Table B.5: Scores for the feature combination OS.

OS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	7	0.52	0.62	0.79	0.78	0.50
k-means	7	0.50	0.63	0.79	0.78	0.50
OPTICS	6	1.04	0.34	0.70	0.79	0.55

Table B.6: Scores for the feature combination O.

O	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	5	0.51	0.59	0.73	0.80	0.53
k-means	5	0.48	0.61	0.73	0.79	0.53
OPTICS	9	1.03	0.20	0.80	0.71	0.46

Table B.7: Scores for the feature combination S.

S	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	5	0.51	0.62	0.78	0.82	0.52
k-means	5	0.45	0.64	0.78	0.84	0.53
OPTICS	8	1.30	0.35	0.74	0.81	0.54

Table B.8: Scores for the feature combination INOS.

INOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	7	0.62	0.55	0.71	0.84	0.56
k-means	7	0.61	0.56	0.71	0.83	0.56
OPTICS	8	1.11	0.22	0.71	0.74	0.51

Table B.9: Scores for the feature combination INO.

INO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	4	0.62	0.56	0.54	0.91	0.69
k-means	4	0.62	0.56	0.54	0.91	0.69
OPTICS	10	1.08	0.05	0.71	0.74	0.51

Table B.10: Scores for the feature combination INS.

INS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	5	0.69	0.52	0.66	0.90	0.62
k-means	5	0.69	0.52	0.66	0.90	0.62
OPTICS	9	1.28	0.27	0.68	0.79	0.55

Table B.11: Scores for the feature combination IN.

IN	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	5	0.55	0.61	0.60	0.90	0.65
k-means	5	0.56	0.61	0.60	0.90	0.65
OPTICS	9	1.38	0.15	0.63	0.82	0.59

Table B.12: Scores for the feature combination NOS.

NOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	7	0.74	0.45	0.79	0.76	0.48
k-means	7	0.72	0.47	0.78	0.78	0.50
OPTICS	9	1.37	0.24	0.75	0.64	0.45

Table B.13: Scores for the feature combination NO.

NO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	8	0.71	0.43	0.80	0.79	0.50
k-means	8	0.73	0.43	0.80	0.77	0.49
OPTICS	8	1.54	0.21	0.72	0.61	0.45

Table B.14: Scores for the feature combination NS.

NS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	8	0.61	0.52	0.79	0.77	0.49
k-means	8	0.59	0.53	0.78	0.77	0.49
OPTICS	9	1.42	0.34	0.72	0.74	0.51

Table B.15: Scores for the feature combination N.

N	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	8	0.55	0.55	0.79	0.79	0.50
k-means	8	0.54	0.54	0.79	0.79	0.50
OPTICS	9	1.08	0.24	0.81	0.71	0.45

Table B.16: Scores for Affinity and all feature combinations.

Affinity	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	0.44	0.67	0.68	0.86	0.59
IO	4	0.37	0.73	0.54	0.90	0.68
IS	5	0.43	0.68	0.53	0.91	0.69
I	0	N/A	N/A	N/A	N/A	N/A
OS	7	0.52	0.62	0.79	0.78	0.50
O	5	0.51	0.59	0.73	0.80	0.53
S	5	0.51	0.62	0.78	0.82	0.52
INOS	7	0.62	0.55	0.71	0.84	0.56
INO	4	0.62	0.56	0.54	0.91	0.69
INS	5	0.69	0.52	0.66	0.90	0.62
IN	5	0.55	0.61	0.60	0.90	0.65
NOS	7	0.74	0.45	0.79	0.76	0.48
NO	8	0.71	0.43	0.80	0.79	0.50
NS	8	0.61	0.52	0.79	0.77	0.49
N	8	0.55	0.55	0.79	0.79	0.50

Table B.17: Scores for k-means and all feature combinations.

k-means	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	0.44	0.67	0.68	0.86	0.59
IO	4	0.37	0.73	0.54	0.90	0.68
IS	5	0.43	0.68	0.53	0.91	0.69
I	3	0.04	0.97	0.35	0.94	0.79
OS	7	0.50	0.63	0.79	0.78	0.50
O	5	0.48	0.61	0.73	0.79	0.53
S	5	0.45	0.64	0.78	0.84	0.53
INOS	7	0.61	0.56	0.71	0.83	0.56
INO	4	0.62	0.56	0.54	0.91	0.69
INS	5	0.69	0.52	0.66	0.90	0.62
IN	5	0.56	0.61	0.60	0.90	0.65
NOS	7	0.72	0.47	0.78	0.78	0.50
NO	8	0.73	0.43	0.80	0.77	0.49
NS	8	0.59	0.53	0.78	0.77	0.49
N	8	0.54	0.54	0.79	0.79	0.50

Table B.18: Scores for OPTICS and all feature combinations.

OPTICS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	9	1.22	0.44	0.72	0.76	0.52
IO	9	1.20	0.29	0.71	0.85	0.57
IS	8	0.97	0.60	0.74	0.85	0.55
I	4	1.05	-0.13	0.34	0.85	0.75
OS	6	1.04	0.34	0.70	0.79	0.55
O	9	1.03	0.20	0.80	0.71	0.46
S	8	1.30	0.35	0.74	0.81	0.54
INOS	8	1.11	0.22	0.71	0.74	0.51
INO	10	1.08	0.05	0.71	0.74	0.51
INS	9	1.28	0.27	0.68	0.79	0.55
IN	9	1.38	0.15	0.63	0.82	0.59
NOS	9	1.37	0.24	0.75	0.64	0.45
NO	8	1.54	0.21	0.72	0.61	0.45
NS	9	1.42	0.34	0.72	0.74	0.51
N	9	1.08	0.24	0.81	0.71	0.45

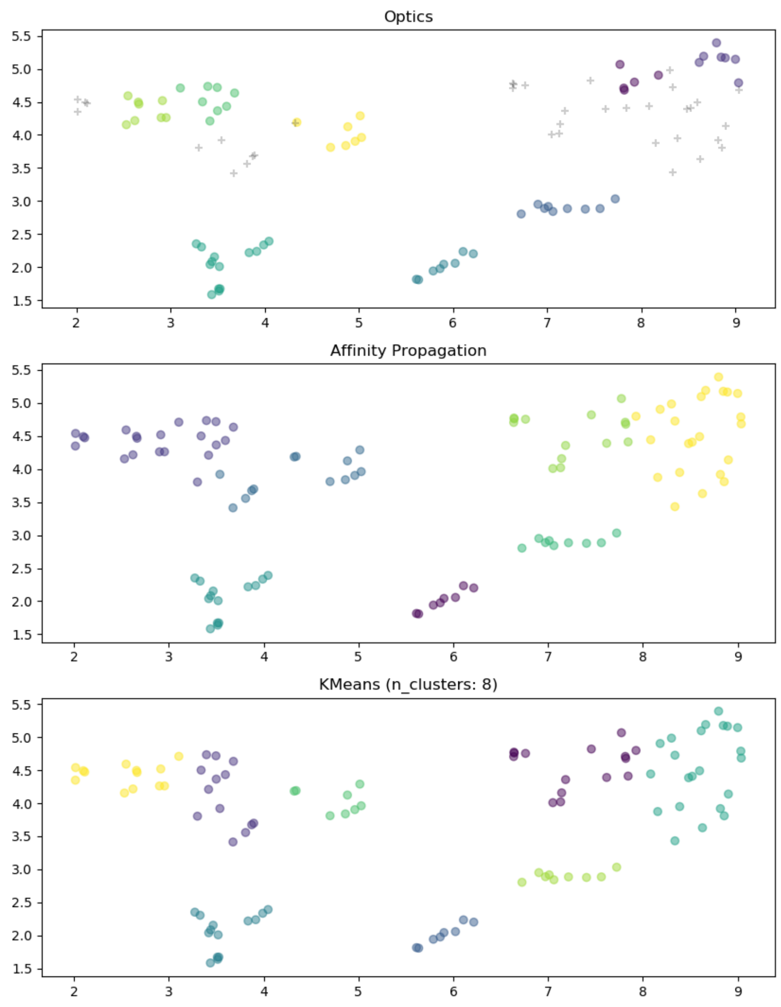

Figure B.1: Visualization for feature combination INOS.

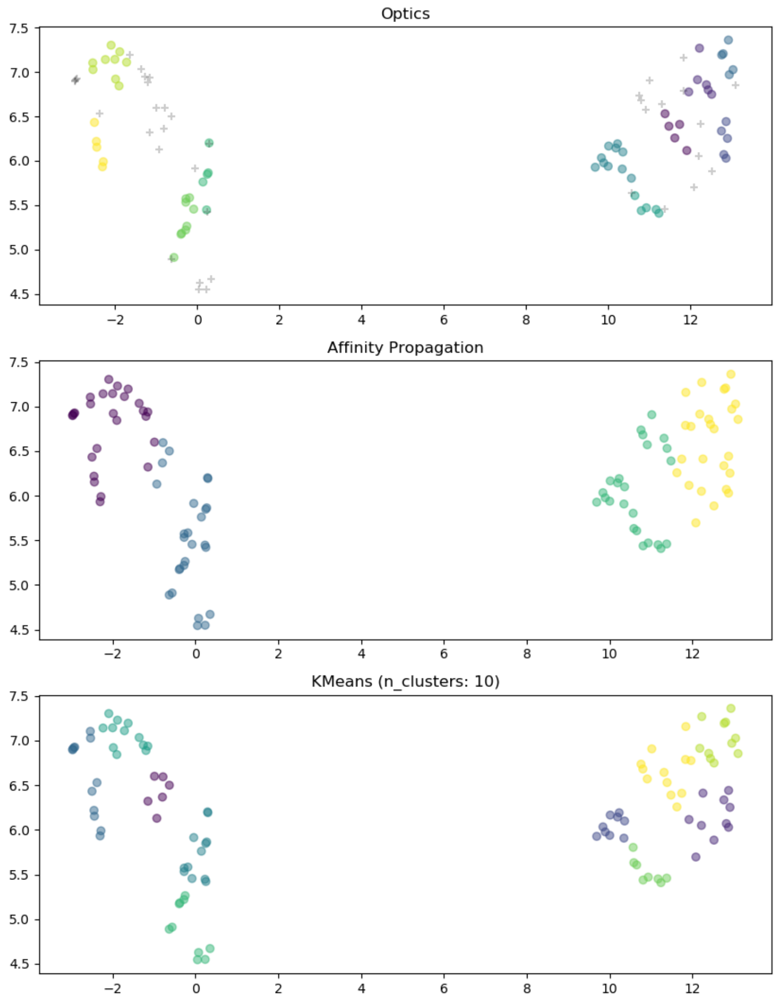

Figure B.2: Visualization for feature combination INO.

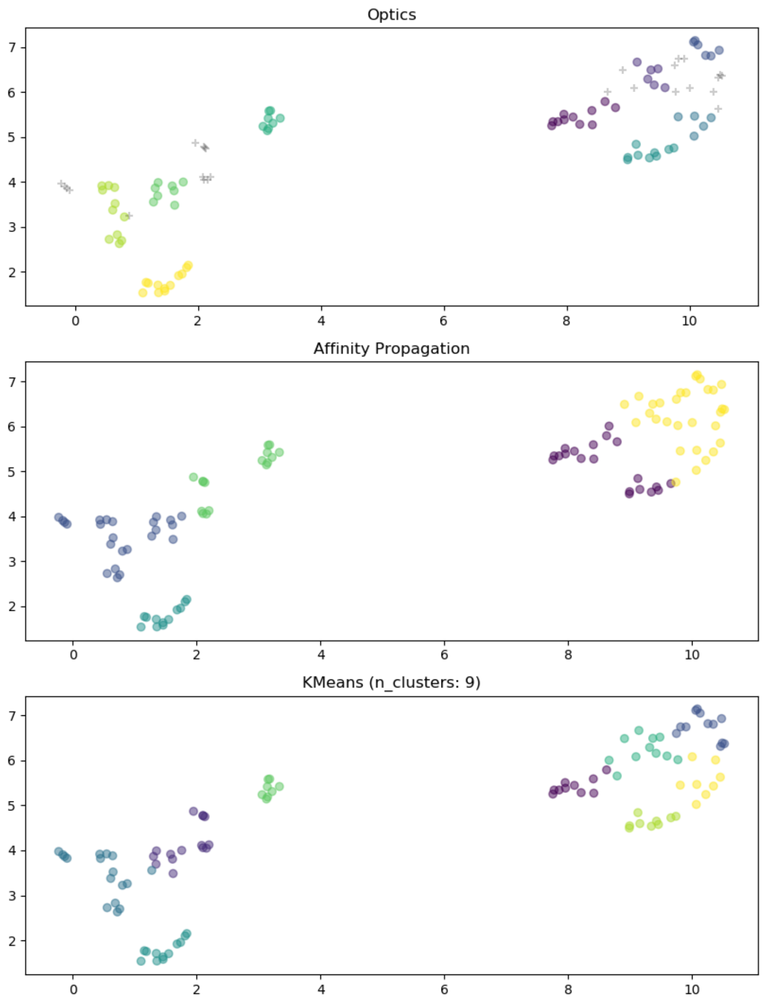

Figure B.3: Visualization for feature combination INS.

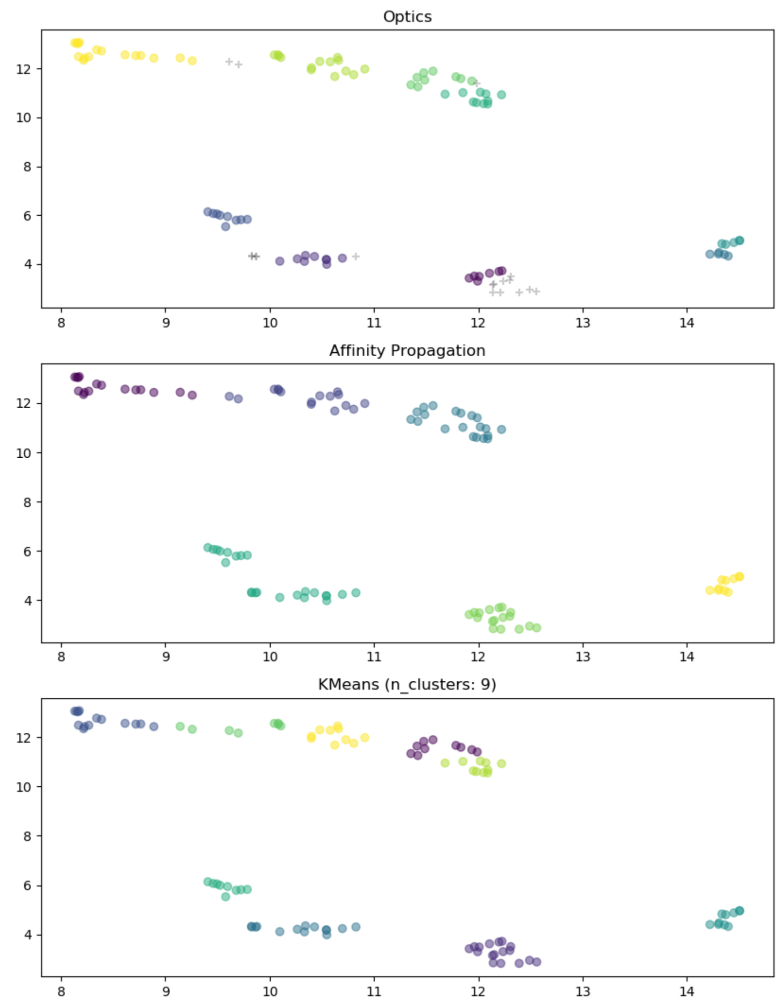

Figure B.4: Visualization for feature combination IOS.

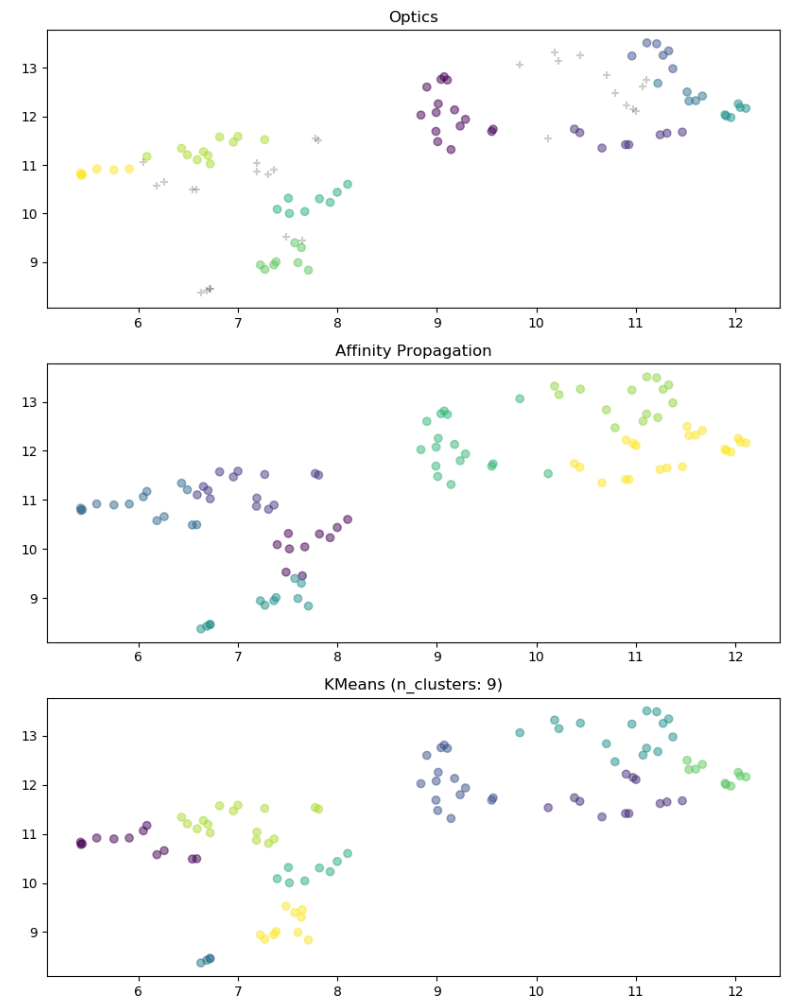

Figure B.5: Visualization for feature combination NOS.

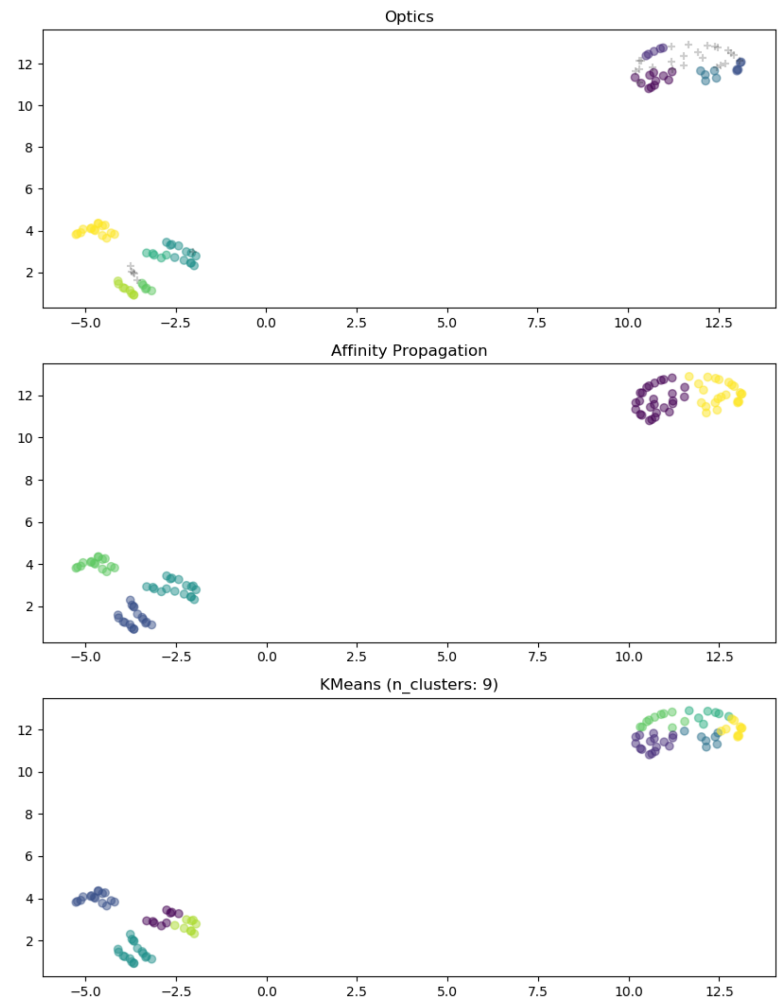

Figure B.6: Visualization for feature combination IN.

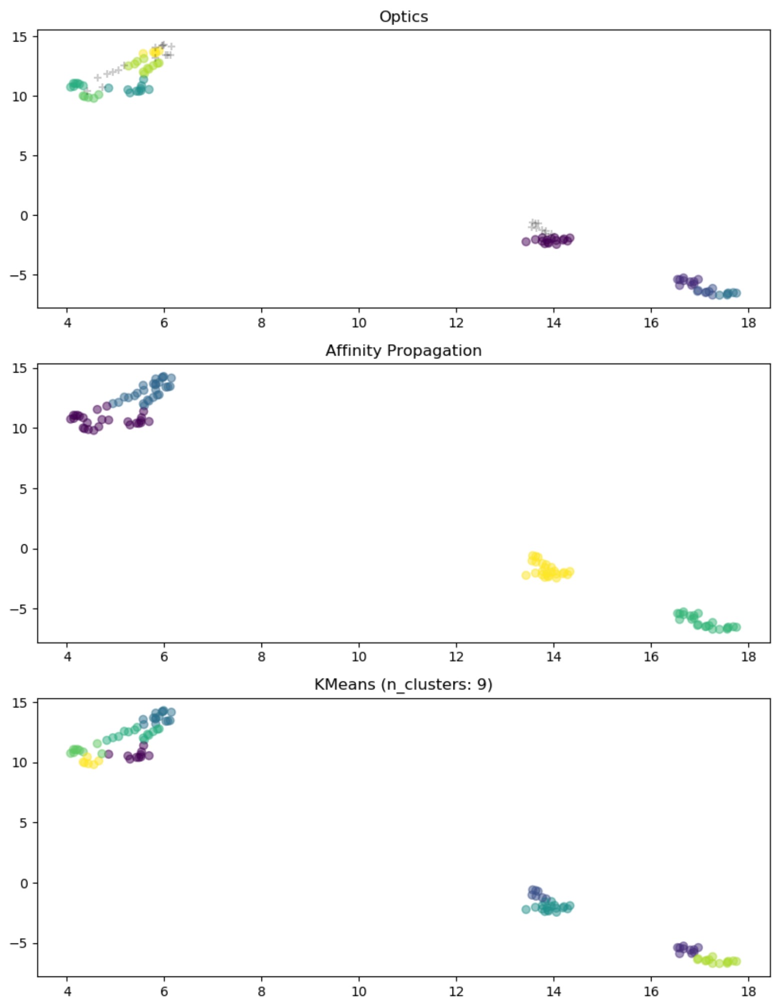

Figure B.7: Visualization for feature combination IO.

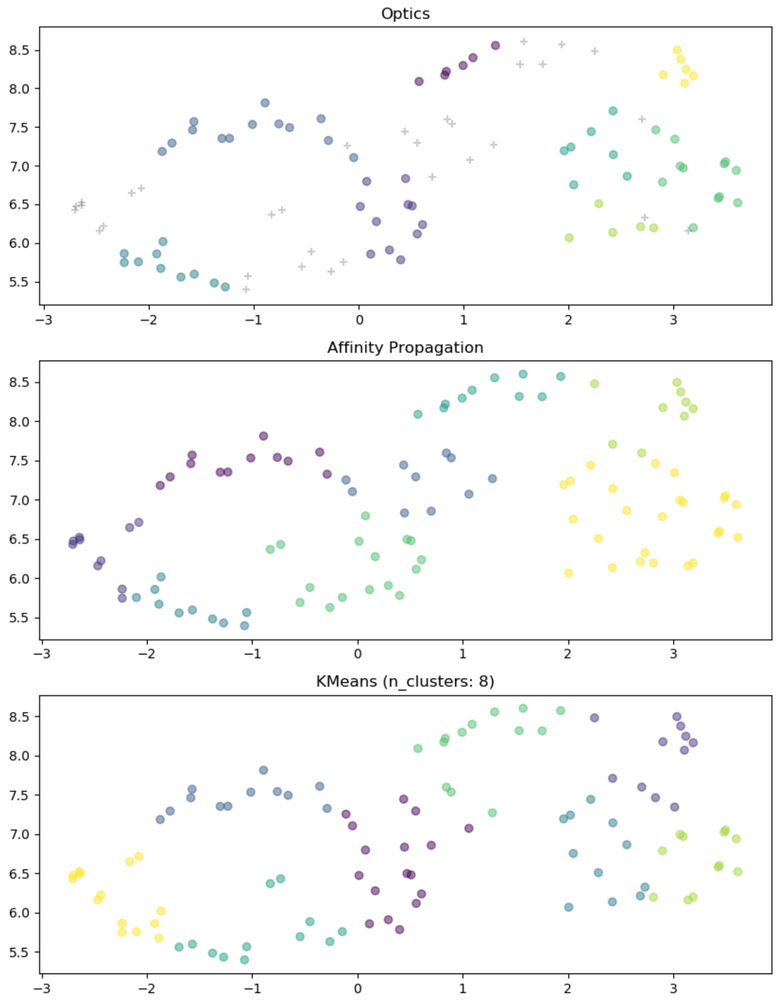

Figure B.8: Visualization for feature combination NO.

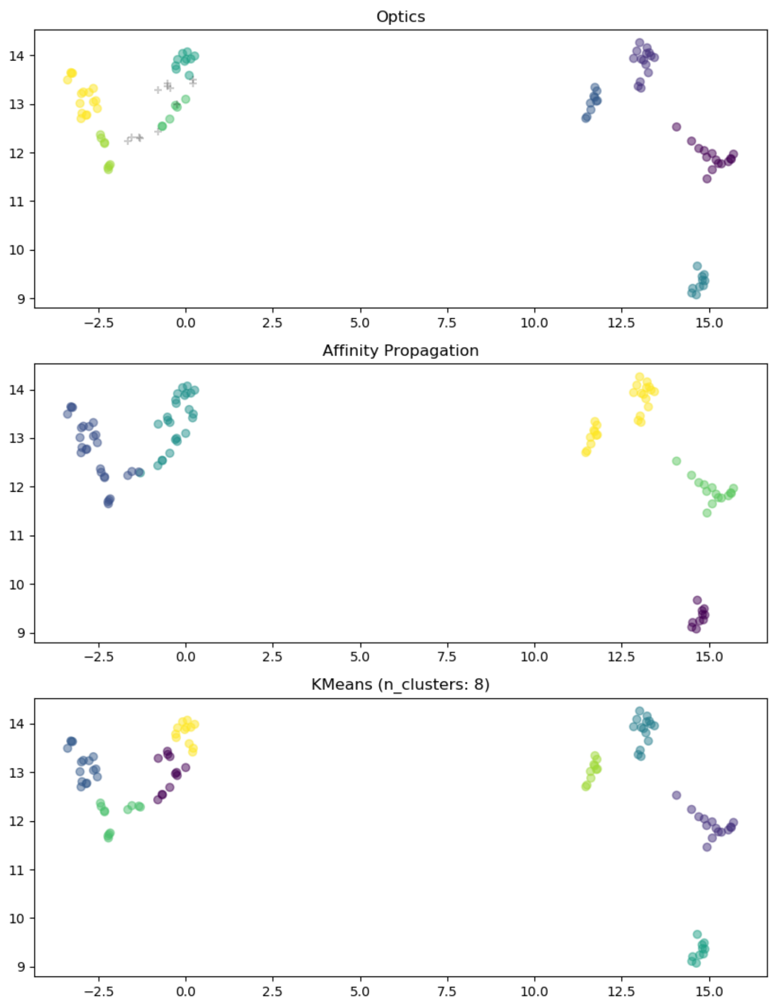

Figure B.9: Visualization for feature combination IS.

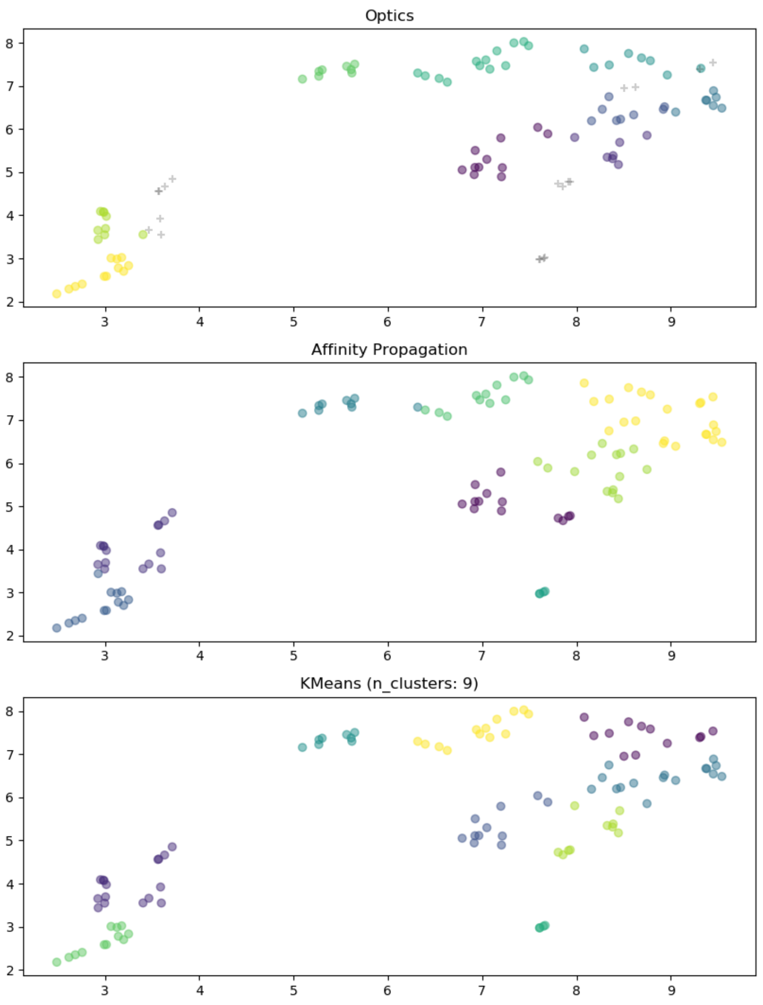

Figure B.10: Visualization for feature combination NS.

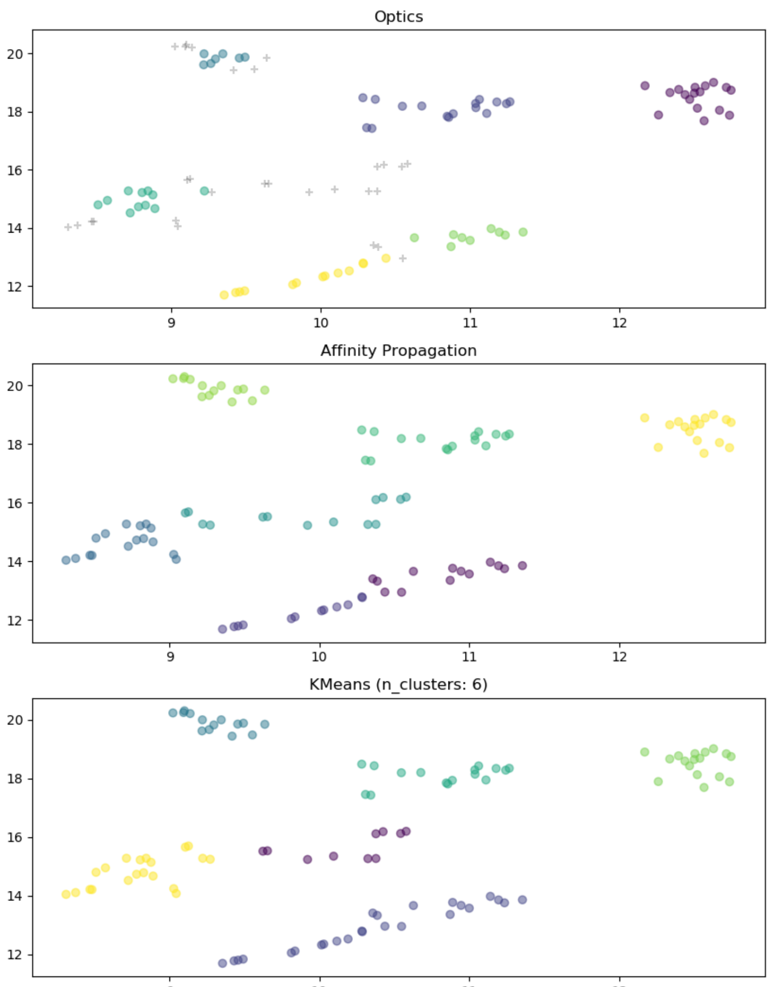

Figure B.11: Visualization for feature combination OS.

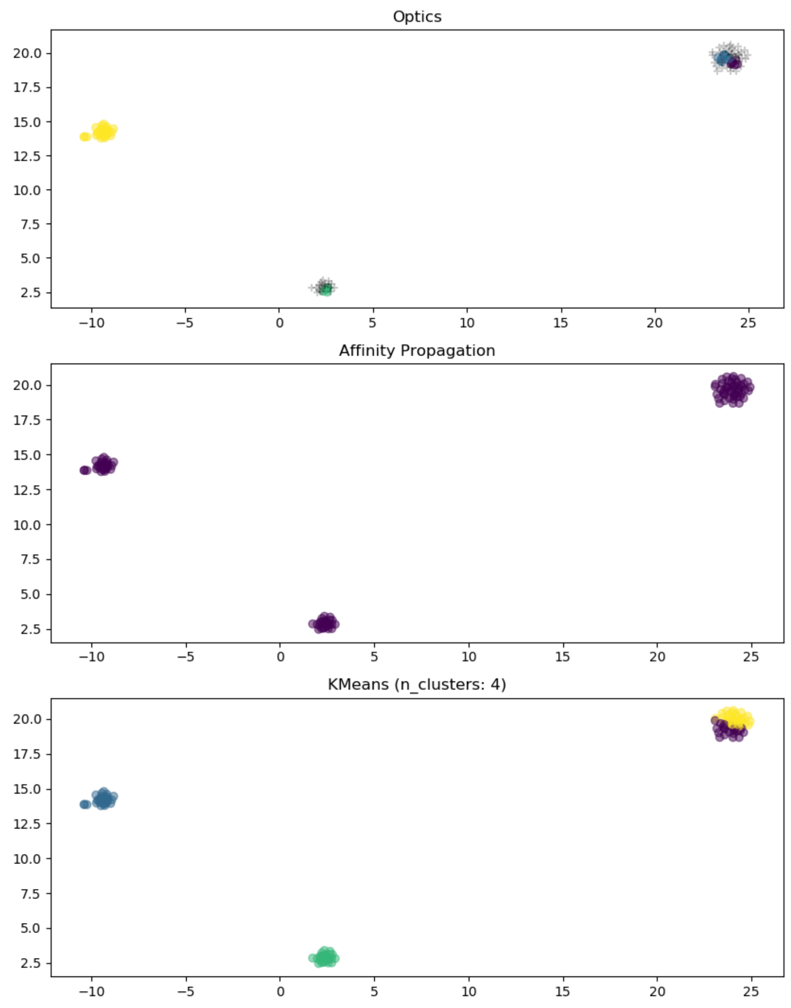

Figure B.12: Visualization for feature combination I.

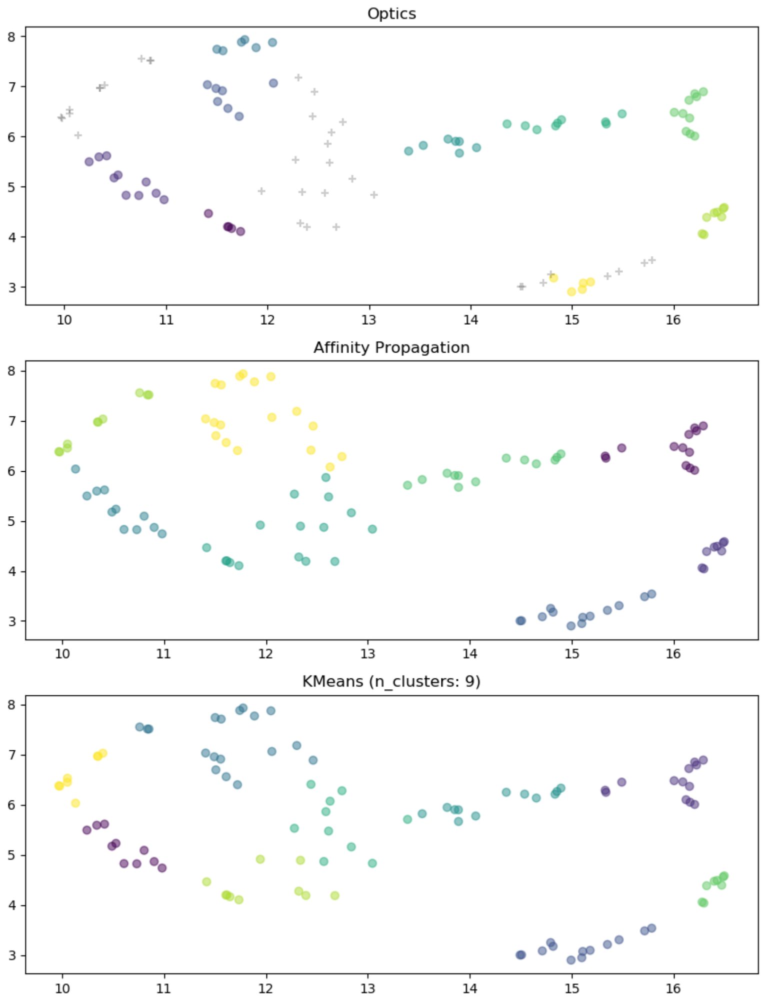

Figure B.13: Visualization for feature combination N.

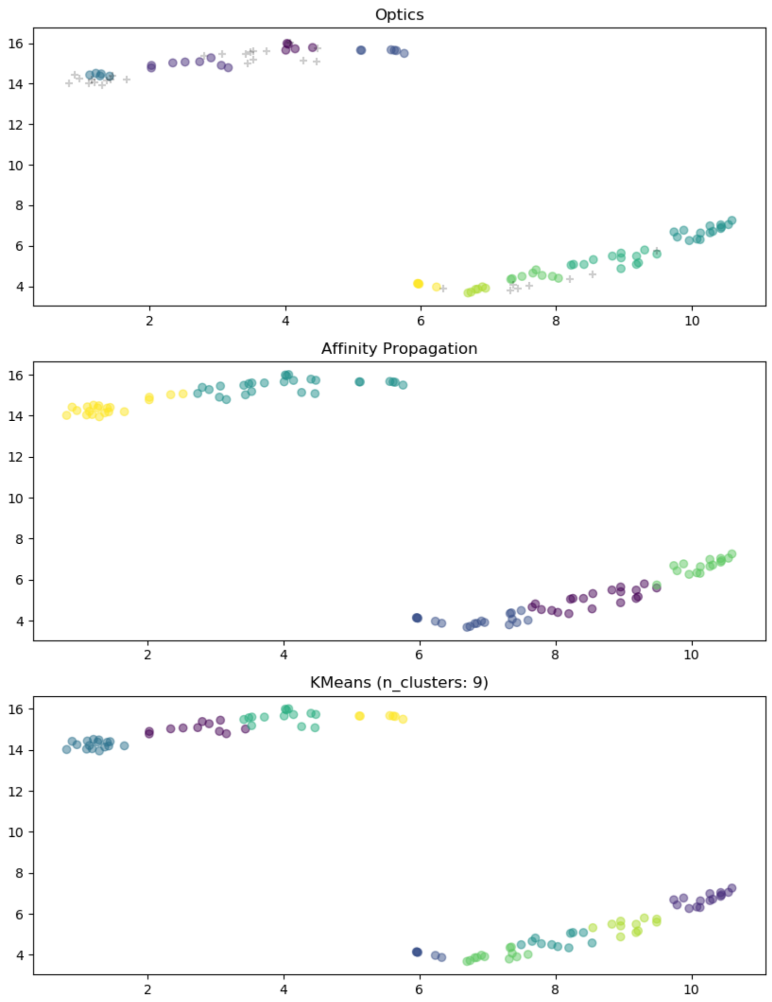

Figure B.14: Visualization for feature combination O.

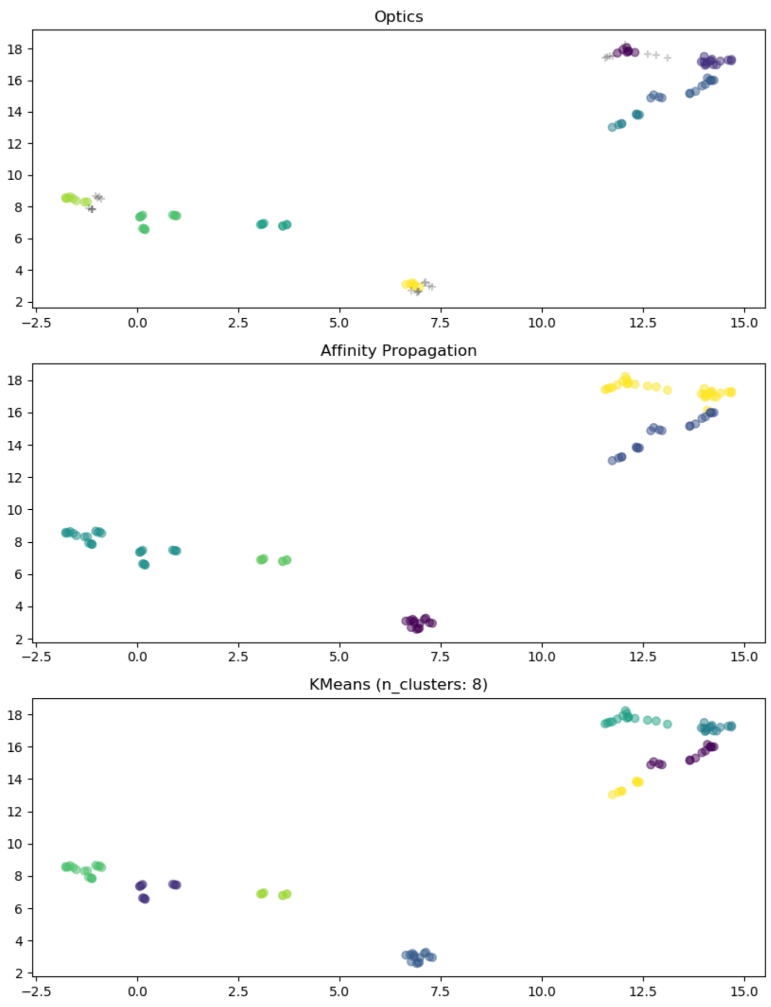

Figure B.15: Visualization for feature combination S.

Appendix C

PCA

Table C.1: Scores for the feature combination IOS.

IOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	0	N/A	N/A	N/A	N/A	N/A
k-means	6	0.98	0.46	0.62	0.89	0.63
OPTICS	6	1.94	0.13	0.52	0.78	0.63

Table C.2: Scores for the feature combination IO.

IO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	10	0.64	0.53	0.75	0.85	0.55
k-means	4	0.92	0.51	0.64	0.90	0.63
OPTICS	4	1.29	0.20	0.34	0.91	0.78

Table C.3: Scores for the feature combination IS.

IS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	10	0.80	0.46	0.79	0.84	0.53
k-means	7	0.77	0.54	0.73	0.86	0.57
OPTICS	7	1.94	0.14	0.70	0.77	0.53

Table C.4: Scores for the feature combination I.

I	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	0	N/A	N/A	N/A	N/A	N/A
k-means	3	0.39	0.90	0.35	0.94	0.79
OPTICS	3	0.44	0.97	0.35	0.94	0.80

Table C.5: Scores for the feature combination OS.

OS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	14	0.73	0.50	0.86	0.68	0.41
k-means	6	0.85	0.50	0.73	0.78	0.53
OPTICS	6	2.12	0.13	0.56	0.80	0.62

Table C.6: Scores for the feature combination O.

O	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	12	0.69	0.47	0.84	0.72	0.44
k-means	5	0.53	0.64	0.76	0.81	0.53
OPTICS	5	1.57	0.14	0.71	0.61	0.45

Table C.7: Scores for the feature combination S.

S	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	11	0.51	0.57	0.84	0.71	0.44
k-means	6	0.66	0.48	0.74	0.78	0.52
OPTICS	6	1.96	0.13	0.67	0.71	0.52

Table C.8: Scores for the feature combination INOS.

INOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	11	1.07	0.31	0.80	0.80	0.50
k-means	5	1.20	0.35	0.72	0.82	0.55
OPTICS	5	1.86	-0.03	0.49	0.78	0.65

Table C.9: Scores for the feature combination INO.

INO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	1.11	0.27	0.81	0.82	0.51
k-means	4	1.25	0.33	0.64	0.90	0.63
OPTICS	4	2.23	-0.06	0.10	0.83	0.86

Table C.10: Scores for the feature combination INS.

INS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	0.80	0.40	0.81	0.82	0.51
k-means	8	0.92	0.36	0.69	0.86	0.58
OPTICS	8	1.95	0.13	0.70	0.78	0.54

Table C.11: Scores for the feature combination IN.

IN	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	9	0.95	0.34	0.65	0.89	0.62
k-means	5	0.84	0.44	0.53	0.91	0.69
OPTICS	5	1.67	-0.06	0.65	0.91	0.63

Table C.12: Scores for the feature combination NOS.

NOS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	1.05	0.34	0.84	0.69	0.42
k-means	4	1.07	0.37	0.72	0.82	0.55
OPTICS	4	2.21	-0.08	0.37	0.69	0.66

Table C.13: Scores for the feature combination NO.

NO	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	14	0.89	0.32	0.84	0.71	0.43
k-means	4	1.23	0.27	0.77	0.76	0.49
OPTICS	4	1.74	-0.13	0.41	0.52	0.56

Table C.14: Scores for the feature combination NS.

NS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	13	0.84	0.41	0.82	0.74	0.46
k-means	6	0.94	0.32	0.74	0.76	0.51
OPTICS	6	1.86	0.02	0.75	0.85	0.55

Table C.15: Scores for the feature combination N.

N	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
Affinity	9	0.72	0.42	0.79	0.78	0.50
k-means	6	0.75	0.41	0.78	0.77	0.49
OPTICS	6	1.32	0.03	0.76	0.70	0.47

Table C.16: Scores for Affinity and all feature combinations.

Affinity	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	0	N/A	N/A	N/A	N/A	N/A
IO	10	0.64	0.53	0.75	0.85	0.55
IS	10	0.80	0.46	0.79	0.84	0.53
I	0	N/A	N/A	N/A	N/A	N/A
OS	14	0.73	0.50	0.86	0.68	0.41
O	12	0.69	0.47	0.84	0.72	0.44
S	11	0.51	0.57	0.84	0.71	0.44
INOS	11	1.07	0.31	0.80	0.80	0.50
INO	13	1.11	0.27	0.81	0.82	0.51
INS	13	0.80	0.40	0.81	0.82	0.51
IN	9	0.95	0.34	0.65	0.89	0.62
NOS	13	1.05	0.34	0.84	0.69	0.42
NO	14	0.89	0.32	0.84	0.71	0.43
NS	13	0.84	0.41	0.82	0.74	0.46
N	9	0.72	0.42	0.79	0.78	0.50

Table C.17: Scores for k-means and all feature combinations.

k-means	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	0.98	0.46	0.62	0.89	0.63
IO	4	0.92	0.51	0.64	0.90	0.63
IS	7	0.77	0.54	0.73	0.86	0.57
I	3	0.39	0.90	0.35	0.94	0.79
OS	6	0.85	0.50	0.73	0.78	0.53
O	5	0.53	0.64	0.76	0.81	0.53
S	6	0.66	0.48	0.74	0.78	0.52
INOS	5	1.20	0.35	0.72	0.82	0.55
INO	4	1.25	0.33	0.64	0.90	0.63
INS	8	0.92	0.36	0.69	0.86	0.58
IN	5	0.84	0.44	0.53	0.91	0.69
NOS	4	1.07	0.37	0.72	0.82	0.55
NO	4	1.23	0.27	0.77	0.76	0.49
NS	6	0.94	0.32	0.74	0.76	0.51
N	6	0.75	0.41	0.78	0.77	0.49

Table C.18: Scores for OPTICS and all feature combinations.

OPTICS	n	Davies Bouldin	Silhouette	MSS-External	MSS-Internal	Tot
IOS	6	1.94	0.13	0.52	0.78	0.63
IO	4	1.29	0.20	0.34	0.91	0.78
IS	7	1.94	0.14	0.70	0.77	0.53
I	3	0.44	0.97	0.35	0.94	0.80
OS	6	2.12	0.13	0.56	0.80	0.62
O	5	1.57	0.14	0.71	0.61	0.45
S	6	1.96	0.13	0.67	0.71	0.52
INOS	5	1.86	-0.03	0.49	0.78	0.65
INO	4	2.23	-0.06	0.10	0.83	0.86
INS	8	1.95	0.13	0.70	0.78	0.54
IN	5	1.67	-0.06	0.65	0.91	0.63
NOS	4	2.21	-0.08	0.37	0.69	0.66
NO	4	1.74	-0.13	0.41	0.52	0.56
NS	6	1.86	0.02	0.75	0.85	0.55
N	6	1.32	0.03	0.76	0.70	0.47

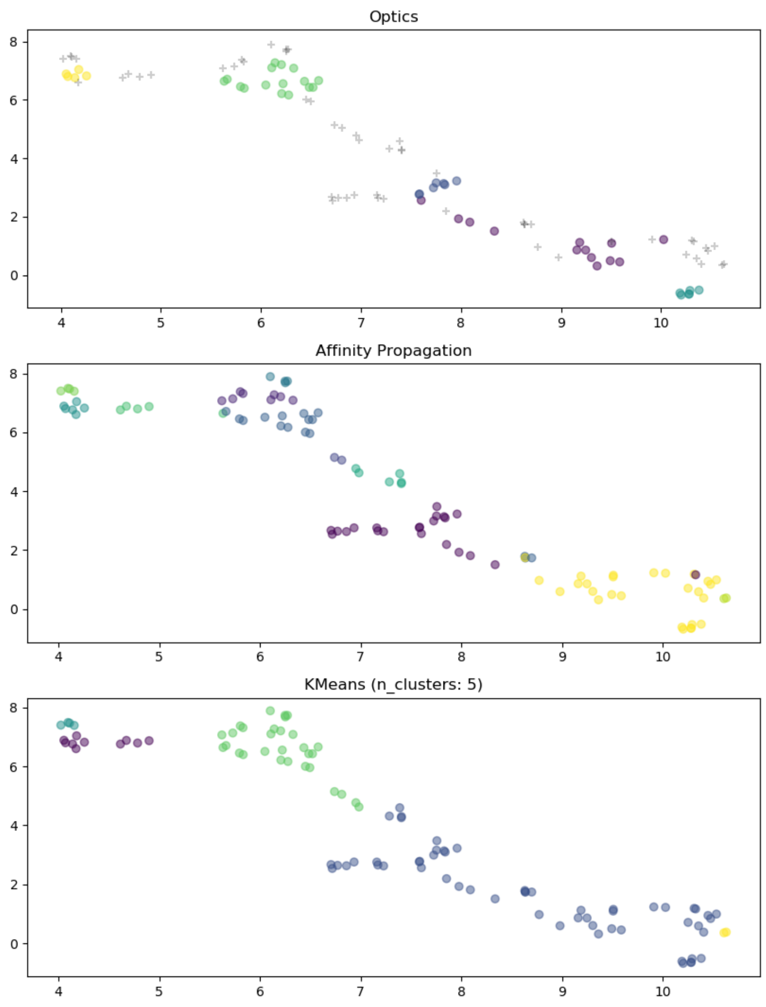

Figure C.1: Visualization for feature combination INOS.

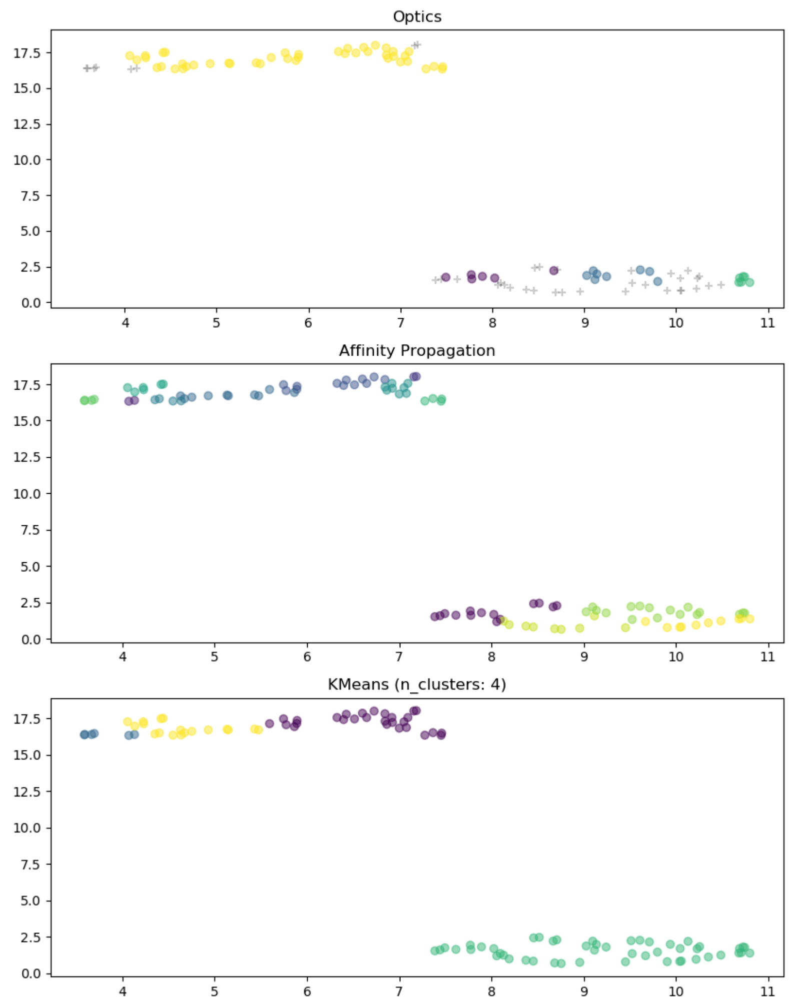

Figure C.2: Visualization for feature combination INO.

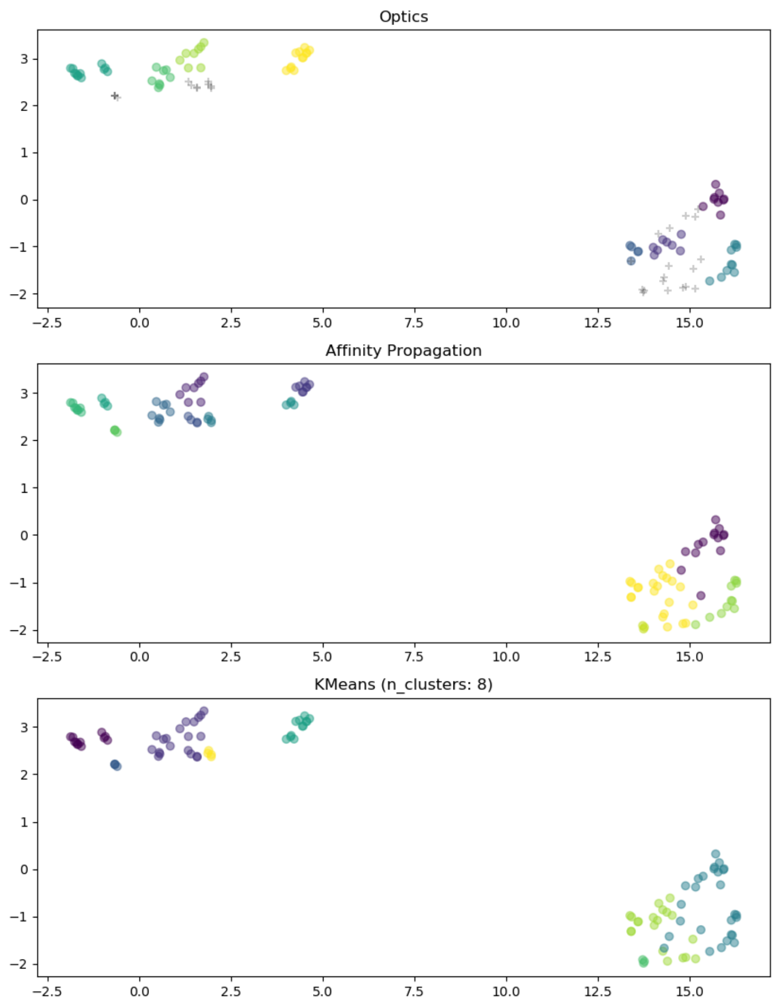

Figure C.3: Visualization for feature combination INS.

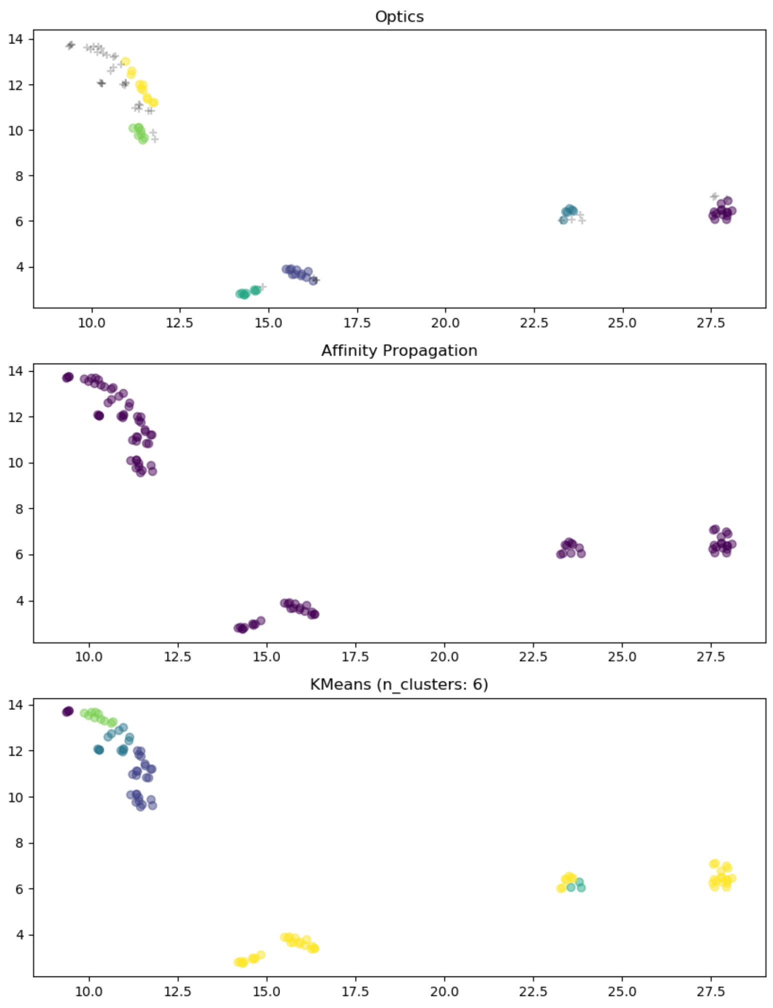

Figure C.4: Visualization for feature combination IOS.

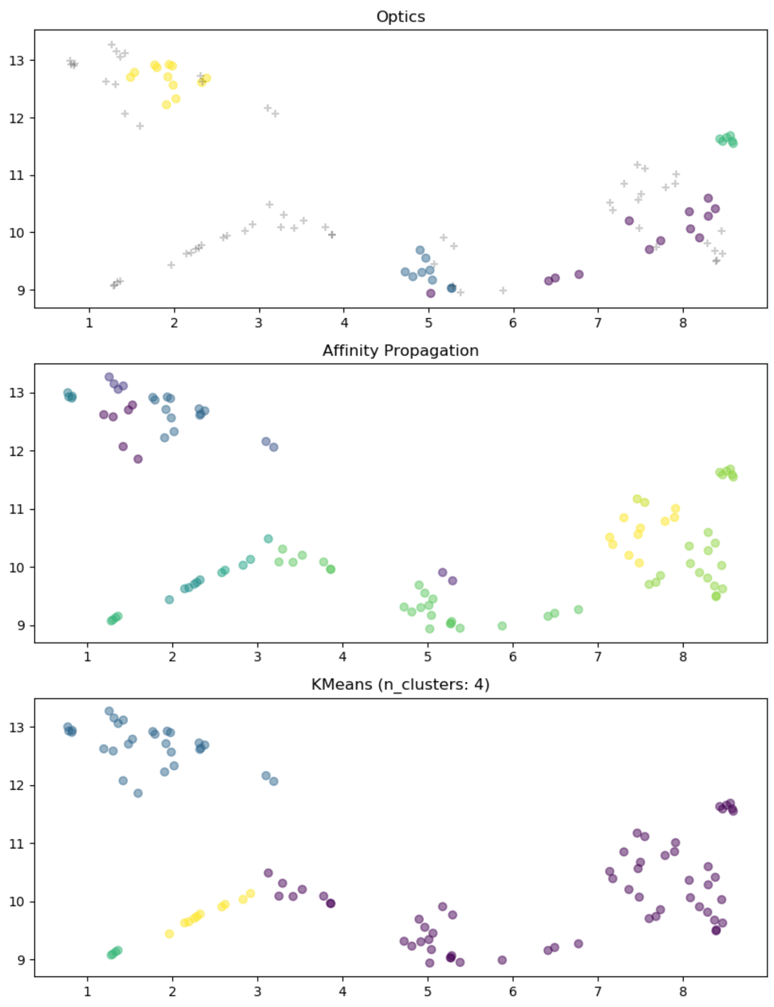

Figure C.5: Visualization for feature combination NOS.

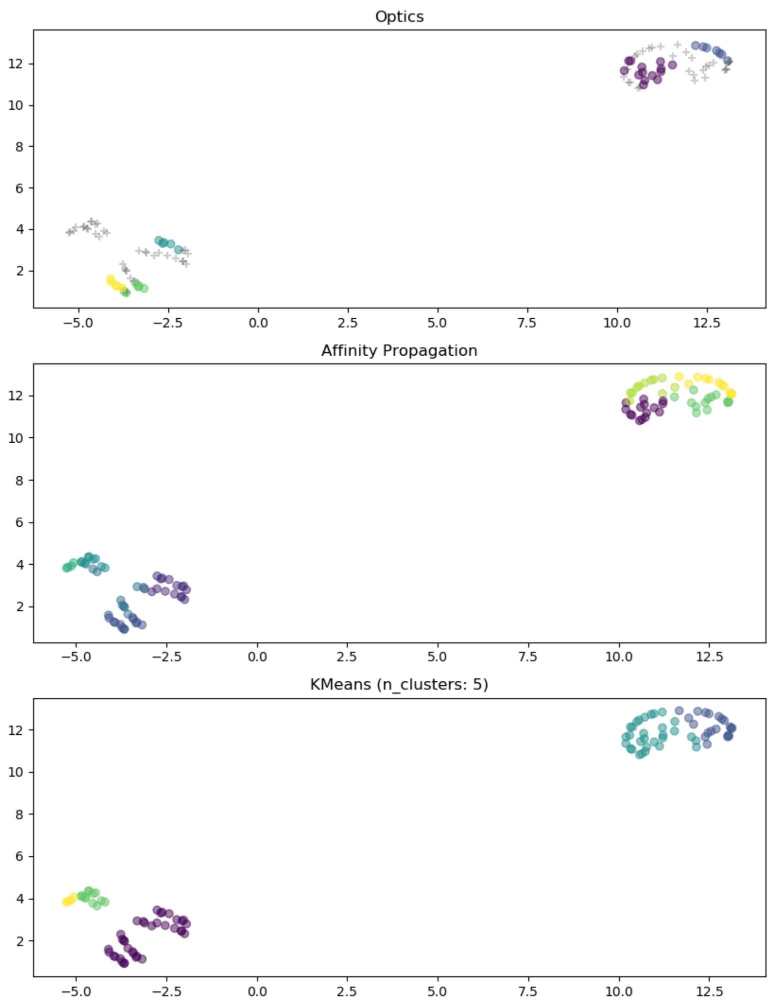

Figure C.6: Visualization for feature combination IN.

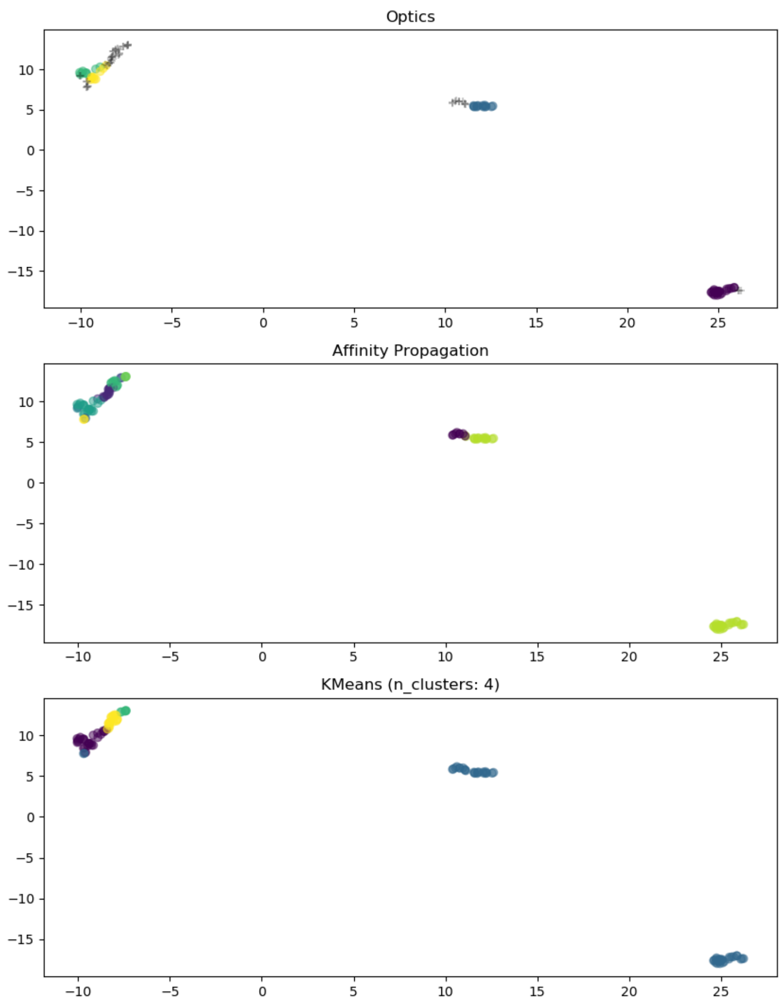

Figure C.7: Visualization for feature combination IO.

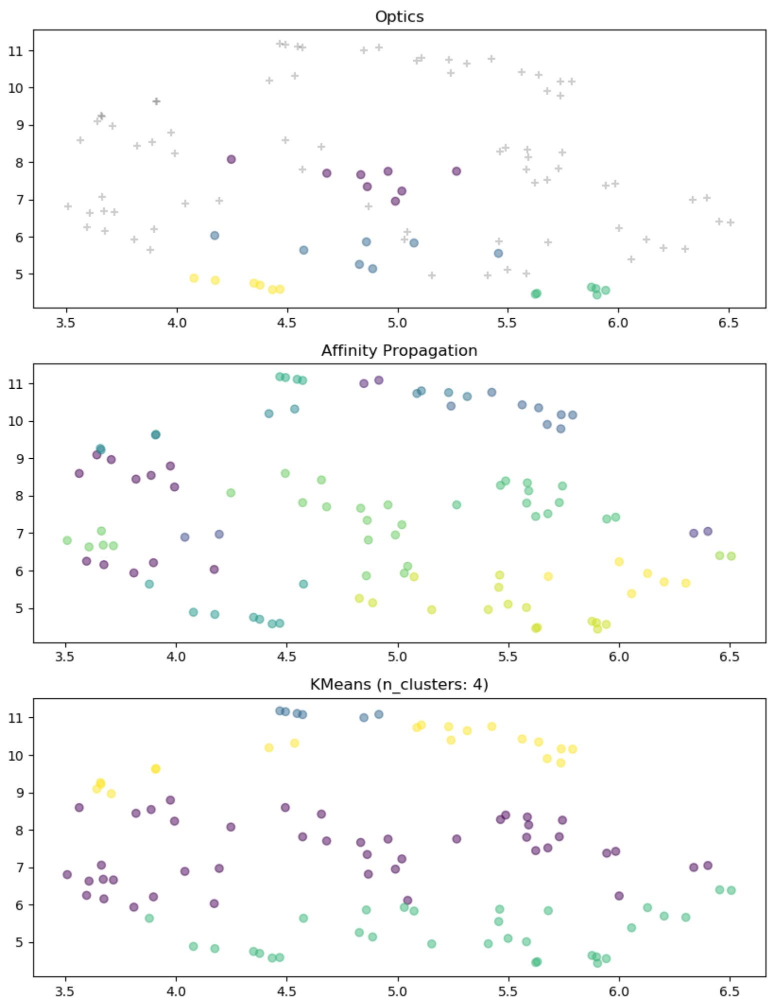

Figure C.8: Visualization for feature combination NO.

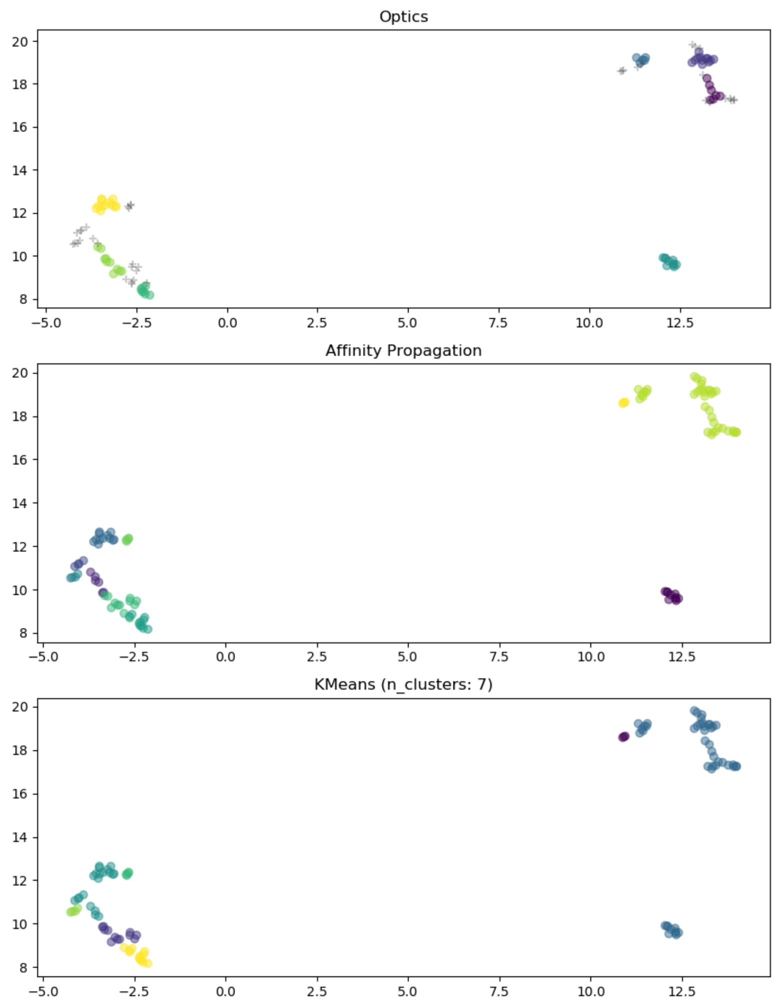

Figure C.9: Visualization for feature combination IS.

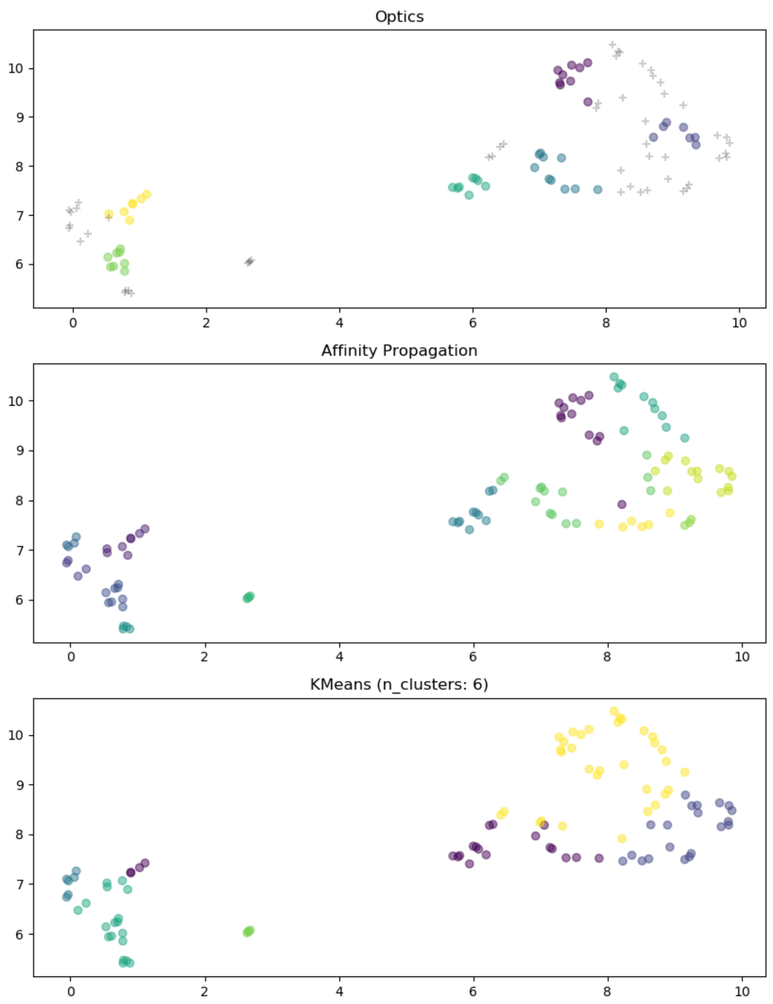

Figure C.10: Visualization for feature combination NS.

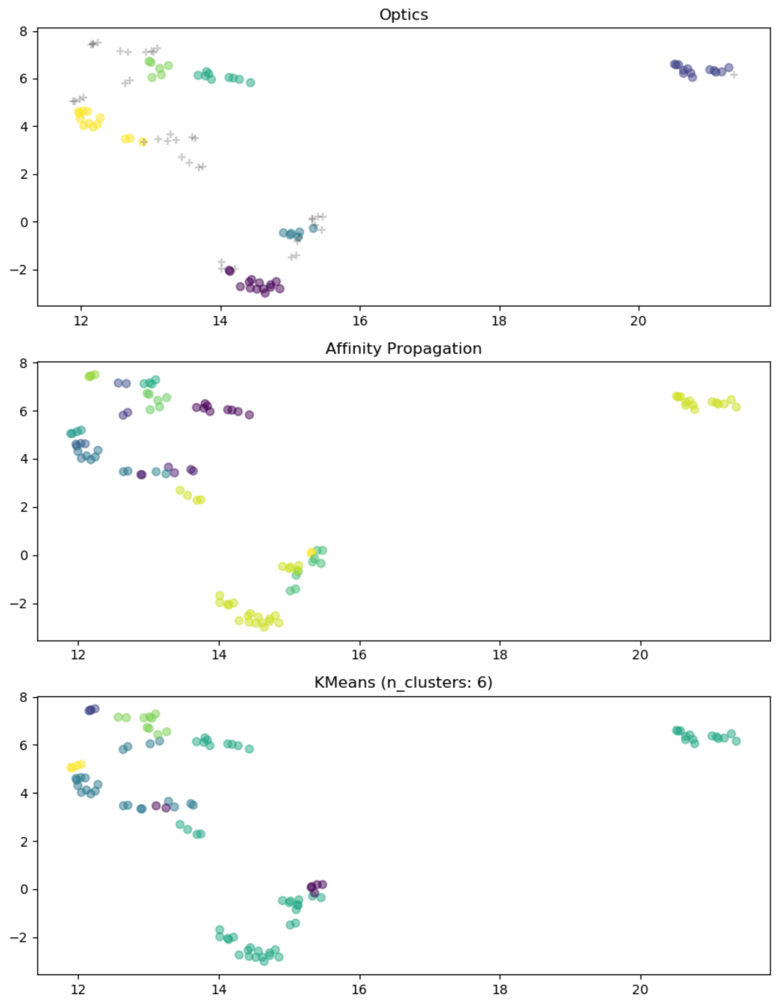

Figure C.11: Visualization for feature combination OS.

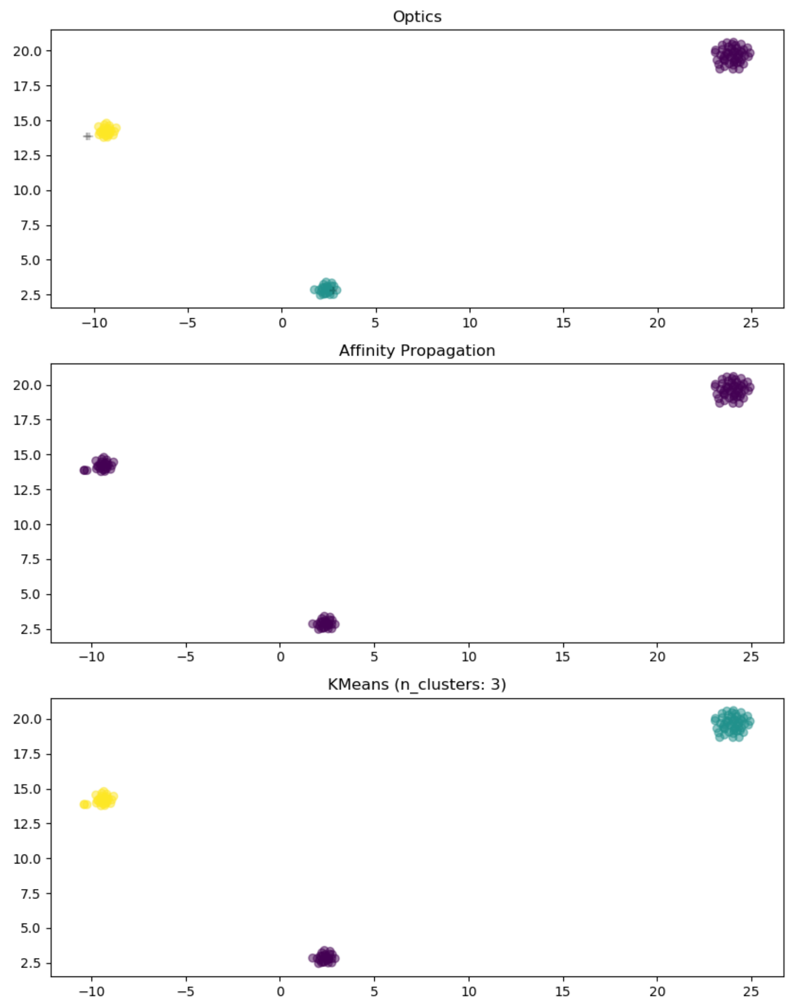

Figure C.12: Visualization for feature combination I.

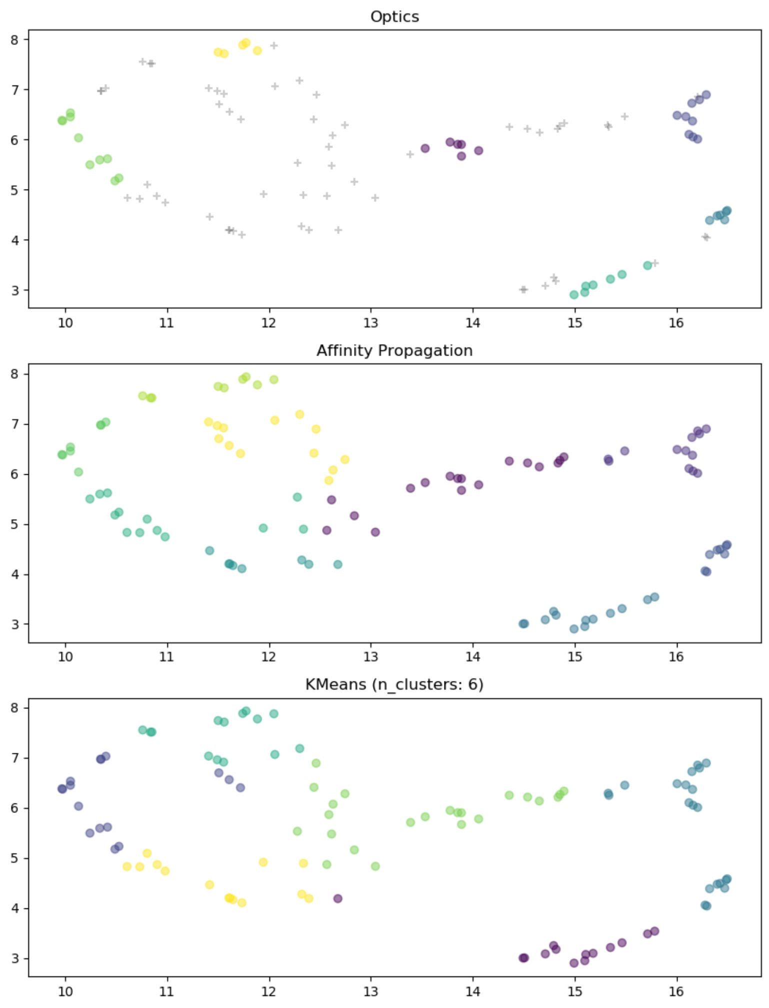

Figure C.13: Visualization for feature combination N.

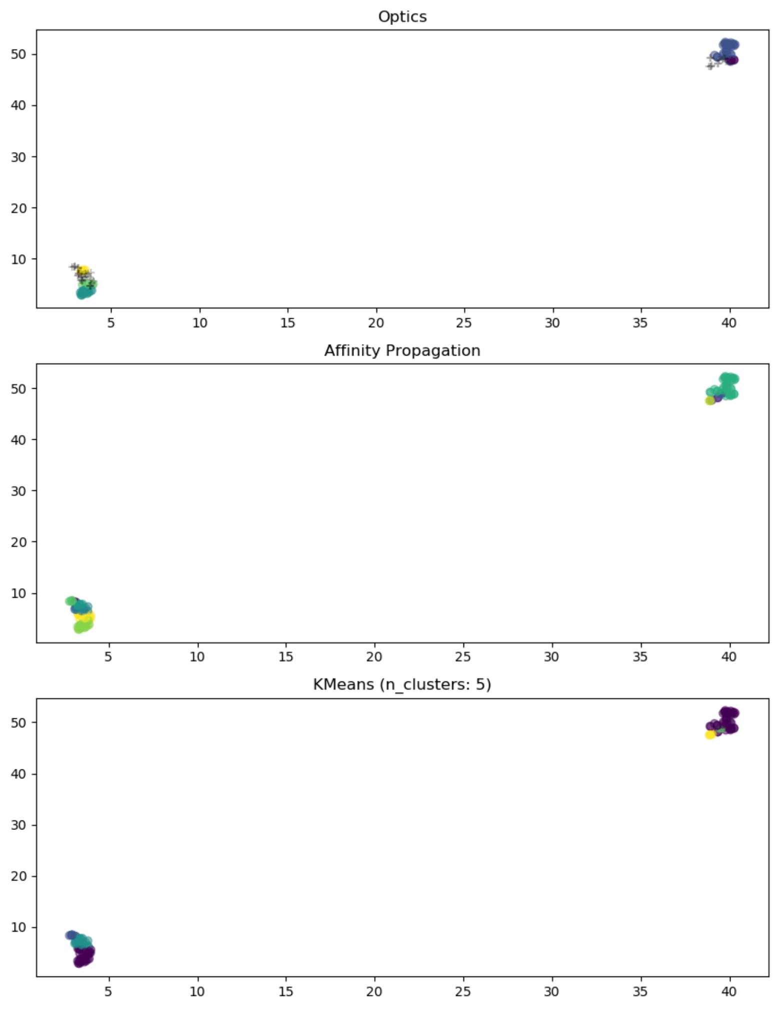

Figure C.14: Visualization for feature combination O.

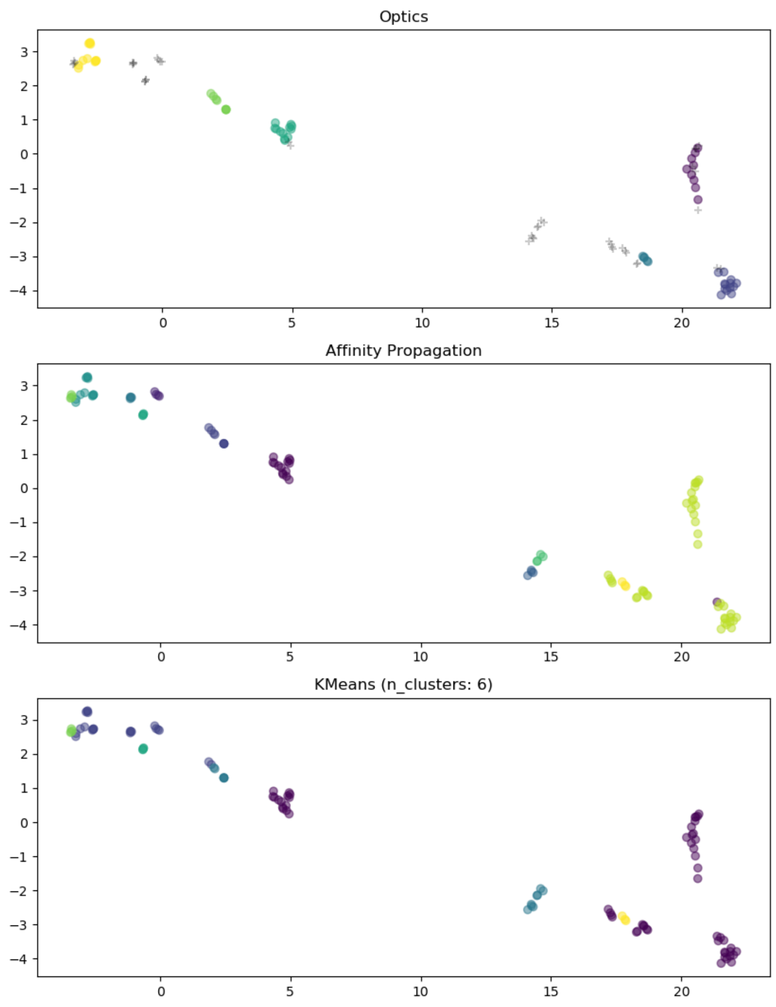

Figure C.15: Visualization for feature combination S.

Printed by Tryckeriet i E-huset, Lund 2021

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2021-812
http://www.eit.lth.se