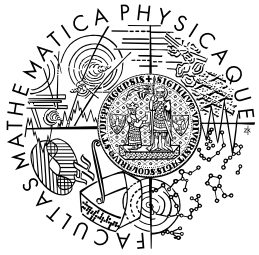

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Aakash Ravi

Machine learning based identification of separating features in molecular fragments

Department of Software Engineering

Supervisor of the bachelor thesis: **RNDr. David Hoksza, Ph.D**
Study programme: **General Computer Science**
Study branch: **Algorithms and Optimization**

Prague 2017

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: **Machine learning based identification of separating features in molecular fragments**

Author: **Aakash Ravi**

Department: Department of Software Engineering

Supervisor: **RNDr. David Hoksza, Ph.D**, **Department of Software Engineering**

Abstract: Chosen molecular representation is one of the key parameters of virtual screening campaigns where one is searching in-silico for active molecules with respect to given macromolecular target. Most campaigns employ a molecular representation in which a molecule is represented by the presence or absence of a predefined set of topological fragments. Often, this information is enriched by physiochemical features of these fragments: i.e. the representation distinguishes fragments with identical topology, but different features. Given molecular representation, however, most approaches always use the same static set of features irrespective of the specific target. The goal of this thesis is, given a set of known active and inactive molecules with respect to a target, to study the possibilities of parameterization of a fragment-based molecular representation with feature weights dependent on the given target. In this setting, we are given a very general molecular representation, with targets represented by sets of known active and inactive molecules. We subsequently propose a machine-learning approach that would identify which of the features are relevant for the given target. This will be done using a multi-stage pipeline that includes data preprocessing using statistical imputation and dimensionality reduction, application of subspace clustering in the molecular feature space, and finally analysis and scoring of the results. This information will then be fed into the molecular representation and used in further virtual screening campaigns.

Keywords: Cheminformatics, Virtual Screening, Subspace Clustering, Clustering, Machine Learning

Acknowledgements

The work and results in this paper would not have been possible without the tireless efforts of a few individuals and organizations. First and foremost, I would like to thank my two advisors David Hoksza and Petr Skoda, who provided both the inspiration and the guidance to pursue a challenging topic with disruptive applications, even when circumstances got difficult. Secondly, I would like to thank Charles University in Prague for providing me with the opportunity and a scholarship to pursue the research described in this paper, as well as MetaCentrum for providing the parallelized infrastructure to run the large-scale data processing jobs required for this type of research. Last but not least, I would like to thank the researchers in the Database Systems Group at Ludwig-Maximilians-Universität München for their advice and experience in the space of subspace clustering, as well as providing the ELKI data mining framework for the public; Arthur Zimek, in particular, was very helpful and I am grateful for his guidance.

Contents

1. Motivation: Drug Discovery and Virtual Screening

As the processing power of modern computers increases exponentially[2], the industries and domains that utilize these more powerful machines becomes ever more diverse. The field of chemistry, in particular, provides many rich applications of both traditional and modern tools from mathematics and computer science. These applications have become so manifest that a new domain known as *Chemoinformatics* [3] has risen both in academia and industry.

The field of cheminformatics brings with it a novel way to observe and conduct research regarding natural chemical phenomenon. In traditional chemical studies, especially quantum chemistry, molecules are observed in the context of electrons or nuclei, or in the context of the classical molecular model using atoms and bonds. [1] Certain branches of chemical informatics, however, bring in tools from mathematics and computer science to employ a different representation when describing molecules: these branches represent molecules as discrete points in a *chemical or molecular space* defined by certain descriptors or features. We can, for example, consider one set of such descriptors to be *topological fragments*-intramolecular fragments that represent different subgraphs of a 2-D graph representation of the molecule. We could then represent molecules as combinations of such fragments and therefore one specific representation of molecules in the chemical space could be a bit vector that indicates whether a particular topological fragment occurs in a molecule or not. Another representation of molecules in the chemical space could be via the values of specific *physio-chemical features* exhibited by the molecule. These features can range from simple molecular characteristics such as the number of atoms that make up the molecule to more complex intramolecular metrics such as the amount of aromatic atoms or bonds present. Such representations are interesting from a computational standpoint because they abstract a lot of the hidden intricacies of the traditional quantum chemical model. These branches of cheminformatics allow us to reason about chemical interactions with the abstraction of the chemical space with respect to various descriptors or features, rather than reasoning from a more fundamental level as quantum chemistry dictates.

One of the most powerful and intriguing applications of this new view of the structure and relationship between various molecules in the chemical space is known as *virtual screening*, a shift in the methodology used in pharmaceutical studies during the drug discovery process. In the pharmaceutical industry, the early stages of the discovery process rely heavily on finding potential reactants to specific macromolecular targets as *leads*; these leads are then taken into further stages of the drug discovery pipeline, eventually leading to clinical trials and a release of a new product or drug. [4] Before the advent of modern computing, the most sophisticated way of uncovering these leads was through a technique known as wet-lab high-throughput screening (HTS)[4], a relatively expensive and time-consuming method when considered in the context of our modern computing power. HTS uses advanced software and robotics to perform controlled experiments on macromolecular targets in large labs usually found only in industry.

Due to the state-of-the-art technology involved, developing and maintaining such a laboratory is a costly operation; therefore a search for more cost-efficient solutions became the main priority for practitioners of HTS[4]. As a result of this search and the aforementioned computational advances, the new trend of *in silico* or *virtual screening* established itself in the domain of drug discovery. Virtual screening uses known information about the behavior and chemical structure of targets and employs computer software to automate the process of searching for potentially active molecules with respect to this target. Virtual screening has been used in practice with numerous reported successes. [4] VS is usually used in conjunction with HTS: practitioners first use VS to lower the initial amount of potential candidate reactants to solely a high-likelihood subset, and then pass this subset onto the HTS stage, where this subset is further pruned to reveal the true active molecules. Therefore, even though at present VS is not used exclusively in the drug discovery process, it greatly reduces the amount of potential candidates so that HTS becomes much more economically viable.

Put succinctly, the VS process requires software that can learn the reactive behavior of a macromolecular target molecule, based on the structure of the target or the target's prior interactions with other molecules - labeled active if they react with the target, or inactive if they don't react with the target. As mentioned before, the chemical structure of molecules can either be viewed from a more fundamental subatomic perspective or from the more combinatorial and computational viewpoint of the chemical or molecular space. Consequently, one new approach to designing software that can shed light on the reactive behavior of a candidate is a method based on *descriptors in the chemical space*, and in particular, the *topological fragment* descriptors mentioned earlier. In this paper, we design and implement a novel machine learning pipeline that will understand the reactive behavior of a target macromolecule-based solely on in-depth analysis, using fragment descriptors, of prior active and inactive molecules with respect to the target.

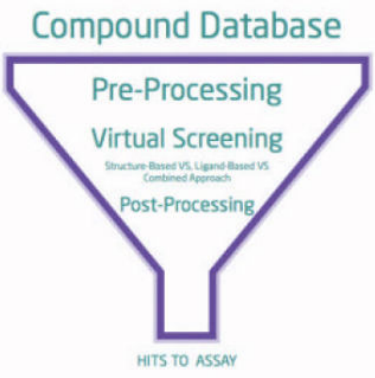

Figure 1.1: A visualization of the virtual screening process, wherein compounds in a chemical database are first preprocessed and then subsequently filtered using numerous possible techniques to determine viable candidates for HTS. (Source: Drug Discovery and Development Magazine [25])

2. Related Work and Our Approach

In this paper, we will introduce a new virtual screening technique that *combines* two separate chemical descriptors that in their own right can be used in virtual screening: topological fragment descriptors and physio-chemical feature descriptors. Topological fragment descriptors, as mentioned before, will help identify our molecules in the combinatorial chemical feature space, where each molecule can be represented by a bit vector where each coordinate is an indicator variable as to the presence or absence of a certain fragment of that molecule; for a visual representation of these intramolecular fragments, the reader is encouraged to look at Figure 2.1. The reader should also note that these topological fragments themselves can be considered as smaller molecules encompassed within a larger one. Feature descriptors, on the other hand, are physio-chemical properties of the molecule - e.g. the number of atoms, the number of aromatic bonds, etc. Various research endeavors have been conducted in the realm of topological fragments and their physio-chemical feature descriptor values in the context of virtual screening; these approaches can be categorized into the two broad categories of **Structure Based Virtual Screening** and **Ligand Based Virtual Screening**.

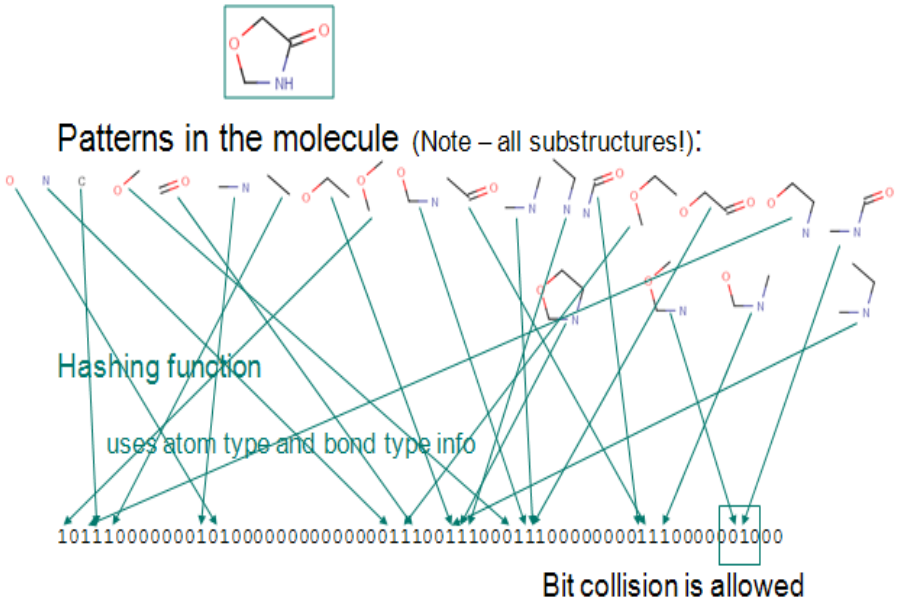

Figure 2.1: A single molecule is represented by a bit vector constructed by considering the different constituent intramolecular fragments within the molecule. Note that each of these *topological fragments* are also considered molecules themselves. [30]

Structure-based virtual screening methods use the structure of a given macromolecular target to rank and filter a chemical database for activity with respect to that target. An example of such a structure based approach is the use of *undesirable pharmacodynamics* of the candidate molecules in the database. Phar-

macodynamics is the study of complementary spatial relationships between the *bonding sites* of two molecules - in our case a given target and a candidate reactant. The filtering is done using 3D Pharmacophores and is based on undesirable fragments (structural alerts) present in some candidate molecules that affect the structural integrity of the bonding site of the candidate molecule with respect to the bonding site of the given target [1]. Our research focuses less on the 3D structural pharmacophore attributes of the topological fragments within a molecule, and more on the physiochemical features of these fragments.

Oftentimes, however, the structure of the target is simply not available. Ligand-based virtual screening, therefore, does not focus on the structure of a given macromolecular target, but rather on the utilization of pre-existing experimental data about prior active molecules with respect to this target to filter and rank the chemical database. Ligand-based methods, therefore, focus less on the target molecules themselves, and rather place more emphasis on the molecules that have reacted to these targets in experiments conducted beforehand. Ligand-based virtual screening can, therefore, be thought of as *similarity searching* of a molecular database; similarity searches of a molecular database are based on the intuitive hypothesis that molecules with a similar structure will have similar reactive behavior. The notion of similarity can be measured both in 2D and 3D, but the 2D metrics are very relevant to our research as they use the notion of a *molecular fingerprint*. These molecular fingerprints are a type of description of the molecule using the presence or absence of topological fragments (similar to our bit vector representation of the molecule), and lead to a very intuitive search of the database. We can, given a prior known active molecule and its fingerprint, for example, search the database for other molecules with similar fingerprints. Such a procedure has been shown to lead to a subset of the database with a high concentration of active molecules.[1] Our ligand-based approach, however, differs from existing similarity search-type algorithms, because although we use topological fragment descriptors, we use also the *physio-chemical feature descriptors* of these fragments. Once again, this combination of fragments and features proves to be a powerful training dataset in virtual screening applications.

2.1 A Dynamic Method

Of the two types of virtual screening methods mentioned in the prior section, our research falls under the category of *Ligand Based Virtual Screening* methods. Our work aims to, for any given macromolecular target and a list of prior active molecules and inactive molecules with respect to this target, **determine dynamically the key physiochemical feature descriptors and their respective value ranges that differentiate active molecules vs. inactive molecules with respect to the target**. Therefore, the user of our algorithms can gain a fundamental understanding of the structure of an active molecule because he/she can identify the key feature descriptors that contribute to activity, as well as their ideal value range within active molecules. The search for further actives and by extension the *ideal structure* of an active molecule, therefore, becomes much easier due to the ability to constrain the search to molecules that exhibit similar values in the key features. Therefore, after running our pipeline for any particular target, we obtain a machine learning model that contains the

aforementioned key features and their corresponding ideal intervals. A user can then use this model to filter their molecular database accordingly so that he/she is left with solely active molecules with respect to the target. Such a process is macromolecular target independent, meaning it can be applied to new targets without change; this leads to a powerful dynamic scheme across targets.

2.2 Machine Learning and Virtual Screening

We take a moment to observe some of the constituents of our ligand based method: the algorithm needs to understand the underlying structure of an active based solely on prior observed data - a task that can be considered challenging even for a human expert. Such understanding, however, is nowadays possible: along with newfound computing power we have also developed algorithms that may utilize this power to allow computers to analyze data, and moreover *learn automatically* different trends and patterns, both salient and latent, found within the data. These new algorithms fall under the general category of *Machine Learning* or *Artificial Intelligence* algorithms - algorithms that allow computer software to learn from existing data and perform a specific task on new data without being explicitly programmed. In our context of virtual screening, this task could be defined, for example, as using prior reaction information to classify new reactants into categories of *active* or *inactive*. For readers already familiar with machine learning, this problem can be formulated as a *discrete classification problem* and can be solved in numerous ways - for example using the statistical technique of logistic regression. Such classification techniques and other methods from machine learning and statistics have been applied by researchers and in industry with successful results.[5]

The tools we will borrow from the fields of machine learning and artificial intelligence, however, will be different from the traditional discrete classification algorithms that could be used to solely classify new candidate reactants into active or inactive categories. Rather, the tools we will employ enable us to **learn the much more profound ideal structure of an active molecule**. More concretely, our virtual screening pipeline will not only be able to filter the chemical database for actives, as do other algorithms, but will also be able to output a set of key physicochemical features and their corresponding ideal value ranges that are exhibited within most of the active molecules.

2.3 The Bayesian Approach

One already explored approach to the dynamic problem mentioned in the prior paragraph is the use of the Naive Bayes classifier, pioneered by Hoksza and Skoda. [27] Put briefly, the Bayesian approach utilizes Bayes' rule to determine the probability that a molecule, represented by a list of physio-chemical feature vectors for each of its fragments, belongs to the *active* class with respect to a macromolecular target. This probability is computed based on prior information given about the molecules which were active or inactive with respect to the target, and the feature values of their constituent topological fragments. For example, for a new candidate molecule m, one can compute the probability that each of the fragments f

of m belong to an active molecule based on the fragments of prior known active molecules; he or she can then take the mean of these probabilities to obtain the probability that m itself is active. By ordering the molecules according to the aforementioned activity probability, the Bayesian approach is quite successful in sorting a database of molecules according to activity for a dynamic target, and therefore we will use this method as one of our benchmarks when evaluating the approach developed in this paper. More will be said about the Bayesian approach in the sections pertaining to experimental evaluation.

2.4 Our Approach: Subspace Clustering

2.4.1 The Molecular Fragment Feature Space

As mentioned many times before, each molecule contains several topological fragments, and each of these fragments can be described by a vector containing the values of its physiochemical features. Therefore, as introduced in the Bayesian approach, a single molecule can be represented by several vectors in the $\mathbb{R}^d$ space (where d is the total amount of physiochemical features), with each vector representing one of its topological fragments.

Looking again to our ligand-based virtual screening pipeline, we are aiming to find the *key physiochemical features* and their *ideal* value ranges that correspond highly with activity for a specific target; we then use a new candidate molecule's fragments' similarity in these key physiochemical ranges to our existing ideal ranges to sort the molecular database according to activity. We present a formal definition of these notions below:

Key Feature. A feature k_i in the molecular physio-chemical feature space is a known as a key feature if for some *integer* α, and *real numbers* β_i and γ_i, it holds that every active molecule contains at least α fragments whose value for the key feature k_i falls in the interval $[\beta_i - \gamma_i, \beta_i + \gamma_i]$. The number α is known as the *key feature threshold*, and β_i and γ_i are known as the *key feature center* and *key feature range* respectively, for feature k_i. The interval $[\beta_i - \gamma_i, \beta_i + \gamma_i]$ is known as the *ideal interval* or the *ideal value range* of key feature k_i.

In order to find these key features and their ideal intervals, we first employ the following informal hypothesis:

Hypothesis For each key feature k_i and its corresponding ideal interval $[\beta_i - \gamma_i, \beta_i + \gamma_i]$, every active molecule contains at least one topological fragment vector f whose value for feature k_i, denoted by f_{k_i}, is in the ideal interval. Every fragment of every inactive molecule, on the other hand, has a value for feature k_i that is outside the ideal value range.

The above hypothesis assumes that α, the key feature threshold, is 1 in our problem setting. It also assumes that *none* of the fragment vectors of the inactive molecules exhibit feature values for the key features that fall in the ideal value ranges. The logic behind the second assumption is that since the inactive molecules don't exhibit the observed bonding behavior with respect to the target, all of their fragments take on values for the key features that display no obvious pattern, unlike some specific active fragments whose values lie within the ideal interval. All of these assumptions indeed are quite strong, but they will greatly simplify our problem setting.

2.4.2 Subspace Clustering in the Molecular Feature Space

The intuitive notions presented in the prior section can be formalized by the use of *subspace clusters*.

Before elaborating on the definition of a subspace cluster, we assume that the reader is generally aware of the notion of a cluster in the Euclidean space. There are various different, but related, definitions of clusters; indeed, these definitions actually aid in explaining the logic behind the many algorithms that aim to find them within some point set P in $\mathbb{R}^d$. For our purposes, we will use a *density based* definition.

Definition: Cluster. Let P be a set of points in $\mathbb{R}^d$. A cluster $C \subseteq P$ in $\mathbb{R}^d$ is a collection of points such that each point p in C has μ neighbors $p'_0 \dots p'_{\mu-1} \in C$ in a ball of radius ϵ, where ϵ is the *neighborhood size* and μ is the *density threshold*.

Definition: Subspace Cluster. Let P be a set of points in $\mathbb{R}^d$, and let the d dimensions be defined by the set $D = dim_1 \dots dim_d$. A subspace cluster $C \subseteq P$ is a collection of points such that for some $S \subset D$, each point $\mathbf{x} \in C$ contains μ neighbors at distance ϵ in each projection of D to the one-dimensional space defined by s, for all $s \in S$.

When considering the notion of a clustered set of points, therefore, a subspace cluster does not take into account the amount of neighbors in an ϵ-ball around a point in the *whole* space $\mathbb{R}^d$, but rather considers only the amount of points in some ϵ-ball in the dimensions $s \in S$ specific to the subspace cluster. Moreover, note that we specify clearly that $S \subset D$ in our definition of the subspace cluster, meaning that $S \neq D$; in this manner, we ensure that a subspace cluster should occur only in a *proper* subspace of $\mathbb{R}^d$.

Figure 2.2 provides an intuitive visual representation and succinct explanation of the role of subspace clusters in the molecular feature space.

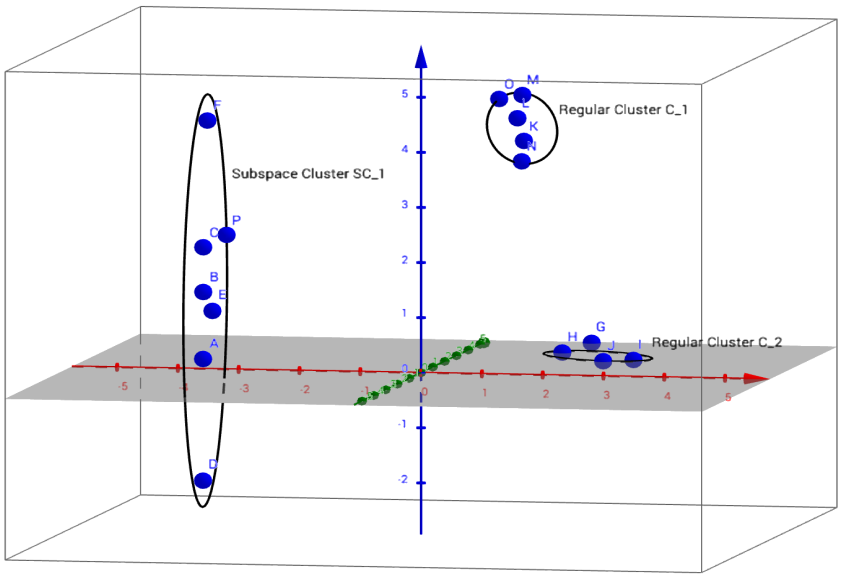

Figure 2.2: The difference between subspace clusters and regular clusters can be seen in the above image. Notice that the two regular clusters C_1 and C_2 are clustered along all the axes, whereas the *subspace* cluster SC_1 is clustered only along the red and green axes, with the values of the points in the blue axis fairly dispersed. One can imagine these 3 axes as *different physiochemical features* in the molecular feature space, and the points as vectors representing the values of the different features for *each topological fragment* of various molecules. Therefore, if SC_1 has a sufficiently high concentration of topological fragments found in active molecules, we can hypothesize that the value ranges of SC_1 in the features corresponding to the red and green axes are a strong indication of activity, whereas the feature corresponding to the blue axis has a marginal effect on activity. In other words, the red and green axes represent key features, while the blue axis does not.

2.4.3 Pure and Diverse Subspace Clusters

It doesn't suffice to just find *all* the subspace clusters in the molecular feature space; rather, we must find subspace clusters that are *pure* - that is, they contain *mostly* fragments from active molecules. The key features can then be identified based on the dimensionality of these detected subspace clusters. A typical example of an impure cluster is one consisting of topological fragments corresponding to *3-carbons*; 3-carbons can be found in virtually any molecule, both active and inactive, and therefore clusters containing many instances of this fragment from various molecules will naturally be impure.

In addition to being pure, the observed clusters should also have *fragments from various active molecules*. Indeed, if we uncover a subspace cluster that contains fragments from solely one active molecule, it may just be due to some intramolecular feature value pattern *within* the molecule itself. By contrast, a subspace cluster that contains fragments from *many* active molecules is *diverse* enough that it could be an indicator of a latent structure in a general active molecule.

The crux of the previous paragraph is that we are looking for subspace clusters

in the molecular feature space that satisfy a few necessary conditions in order for them to be deemed significant in our studies. We present these conditions in a concise and formal manner below, by adding a definition:

Definition. Significant Molecular Subspace Cluster. We call a subspace cluster C found in the molecular feature space over d dimensions a significant cluster - or a *non-degenerate* cluster - if it satisfies the following three conditions. Otherwise, we call it a *insignificant* cluster or also a *degenerate* cluster.

Condition 1: Diversity Let $f_1 \dots f_n$ be the fragments corresponding to the n fragment feature vectors that C contains. Then $f_1 \dots f_n$ belong to k unique active molecules. (C is also then known as a *diverse* molecular subspace cluster).

Note 1: Observe that in the definition of diversity we allow our diversity threshold k to remain a variable; this is done purposefully, as we want the user of the system to be able to control the level of diversity of the clusters himself or herself simply by adjusting k.

Condition 2: Purity Let A be the set of fragments that occur *only* in the active molecules, I be the set of fragments that occur *only* in the inactive molecules, and C_f be the set of fragments whose corresponding fragment feature vectors are contained in the cluster C. Then it holds that $\frac{|A \cap C_f|}{|A \cap C_f| + |I \cap C_f|} \geq p$. ($C$ is also then known as a *pure* molecular subspace cluster).

Note 2: Just as in the diversity threshold, observe that we parametrize the purity threshold p.

A question the reader may certainly have is as to why we even need the condition of purity to ensure we obtain significant clusters, when one could just find subspace clusters within solely the fragments of the active molecules, and exclude the fragments of the inactive molecules; these subspace clusters, after all, will surely be pure. The reason is that many fragments found in the active molecules can also be found in some of the inactive molecules - there is no molecular barrier stopping the occurrence of a topological fragment in both an active molecule and an inactive molecule. Therefore a subspace cluster that occurs within the active fragments could also occur within the inactive fragments, and such a subspace cluster does not lead to any deeper molecular insight about the ideal structure of an active and therefore must not be considered. We, therefore, want to find the subspace clusters within the *combined* active and inactive fragments, and then *prune* these found subspace clusters using the purity condition above to ensure that they truly occur only within the active fragments.

Our presentation of the virtual screening pipeline will be organized as follows: *Cleaning and combining large molecular datasets* will detail the procedures for cleaning noisy molecular data, as well as the data structures involved in representing and manipulating fragment feature vectors in the molecular feature space. These cleaning and data manipulation steps are necessary, as the raw input to our algorithm will be a list of prior known actives and prior known inactives of a specific macromolecular target, their respective topological fragments, and the feature vectors of these topological fragments. As with all large data sets, this collection of molecular information will need a thorough preprocessing procedure before machine learning algorithms can be applied on it.

The Subspace Clustering Approach chapter will explore in detail the various possible subspace clustering algorithms in virtual screening, as well as the chosen

algorithmic method within our research.

Experimental Results will evaluate our approach on various target macromolecules, as well as compare the results from our subspace clustering method to those obtained from the Bayesian method.

Finally, *Conclusions and Future Work* will explore the implications of our results, as well as possible improvements and further follow-up work in future research endeavors.

3. Cleaning and Combining Large Molecular Datasets

The raw molecular data needed for our algorithms is obtained from various online resources; the known active and inactive molecules with respect to each target were obtained from the PubChem Database[26], while the physiochemical feature descriptors for various fragments are obtained from the PaDEL Database.[10] As mentioned towards the end of the prior chapter, the raw input data to our machine learning pipeline requires preprocessing and denoising in order for it to be in a suitable form for machine learning algorithms; this section will explore in detail techniques that can be applied in the molecular feature space for not only cleaning data, but also for sequestering the valuable data from the redundant or even counterproductive data.

We first describe formally the chemical data that we are provided as inputs to our pipeline, in list format:

1. Two subsets A_t and I_t of $\mathcal{M}$. Subset A_t will be known as the *actives* or *ligands* with respect to a specific target t, and the subset I_t will be known as the *inactives* or *decoys* with respect to t. The set A_t contains a set of molecules that have shown prior *reactivity* with respect to t, and similarly the set I_t contains a set of molecules that have shown prior *inactivity* with respect to t.

2. For any molecule m in the set A_t and I_t, we are given a *topological fragment mapping*. This fragment mapping maps the molecule to its constituent toplogical fragments, which, once more, are just constituent molecules $f_1 \dots f_n$ that are contained in m.

3. For any of the topological fragments mentioned in point *3.*, we are given the physiochemical feature mapping that maps the fragment to its corresponding physiochemical feature values in the molecular feature space.

3.1 The Fragment-Feature Matrix

As a prequel to the discussion of our data cleaning and combining pipeline, we will first present the most important data structure that will be employed in our research in order to effectively represent the complex assortment of molecular data presented in the prior section: the **Fragment Feature Matrix**.

As its name suggests, the Fragment Feature Matrix, or *FFM* for brevity reasons, is a *matrix*. The rows of the FFM will be the *physiochemical feature vectors* for the various *fragments of the active or inactive molecules* w.r.t. our target molecule t. The entry FFM_{ij}, for example, will be the *j-th feature value* of the *i-th fragment*. Therefore, the matrix can be perceived as simply a collection of physiochemical feature vectors; this simple structure, however, proves to a powerful way to represent complex data about the inner fragment structures of the numerous molecules provided as input to our pipeline.

We will initially create two separate FFM structures for the fragments of the active molecules and the inactive molecules; this is due to the fact that we would like to perform our imputation procedure on the fragments of the active molecules and inactive molecules separately so that any chemical structure unique

to only the fragments of the active molecules is maintained after the procedure. After proper processing and imputation of the separate FFM structures, they will be combined to form one encompassing FFM to perform our future algorithmic procedures on. For a clearer depiction of this procedure, please see Figure 3.1.

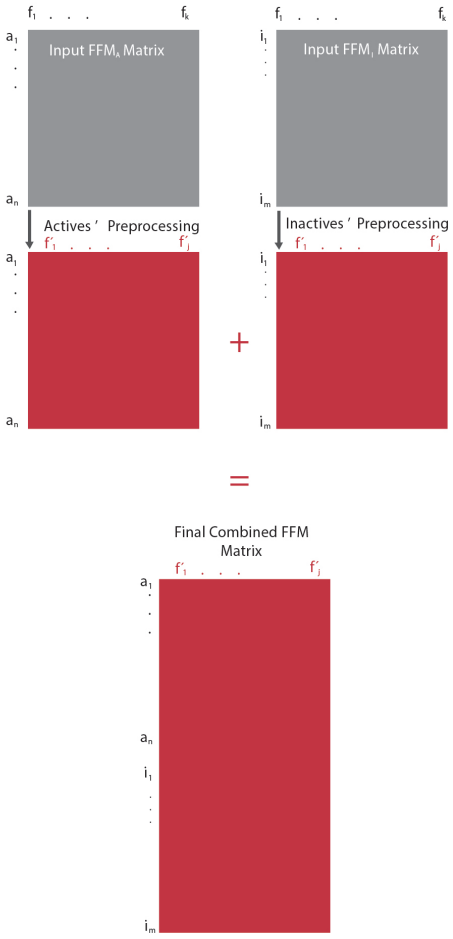

Figure 3.1: A visual depiction of the creation of the singular FFM containing fragment vectors of both of the active molecules and inactive molecules. We are initially given two separate matrices that contain either active fragment feature vectors $(a_1 \dots a_n)$ or analogously inactive fragment feature vectors $(i_1 \dots i_m)$ as rows; the two matrices are then preprocessed and imputed separately via the procedures introduced in the subsequent paragraphs. The resulting pre-processed matrices have columns that represent features $f'_1 \dots f'_j \subseteq f_1 \dots f_k$ that are deemed the *most significant features* during our processing of the matrices. Finally, the two matrices are combined or concatenated to form the final FFM for both the active and inactive fragments.

3.2 Constant Feature Removal

The first step of the data cleaning pipeline is a small but important step within the grand theme of molecular data purification. One of the primary problems

that we observe in our fragment feature matrices - albeit one that easy to solve - is the abundance of columns with the same values in every component. This signifies that some physiochemical features in our database attain the same values for every topological fragment. This is, for most features, never the case in the physical world, and therefore we assume with good reason that this is due to some error during the storage and prior processing of the data. In this situation we don't design any complex or clever algorithms: we simply *remove the columns* of the fragment feature matrices that correspond to the features that attain constant values across all fragments. To the best of our knowledge, no other intuitive and yet effective scheme exists to rectify such constant-valued features, and therefore our simple removal method seems to be the optimal way to deal with such a degenerate scenario.

3.3 Imputation of the Fragment Feature Matrix

Assuming the removal of all the constant-valued features from the fragment feature matrices, the key data purity problem whose effects we aim to mitigate is the fact that we occasionally encounter values for specific features that are not numbers, whereas all of the physiochemical feature values we use are represented by numerical values. For example, certain features of some fragments, according to the inputted molecular database, have values such as − or simply a blank. The reason for these corrupted feature values is presumably because no data was available on these features for particular fragments, and therefore non-numerical characters have been inputted to signal that these values are missing. These values obviously cannot appear in our final FFM because it would cause various errors later on in the pipeline when our algorithms try and operate on non-numerical values. Missing data is indeed a common occurrence in the world of data analysis, and there have been whole papers published on how to deal with this occurrence. In this paper, however, we will keep the discussion focused solely on the methods we choose to deal with the lack of data in our specific use-case in the molecular feature space, and refer the reader to other resources (e.g. [12]) for a more detailed discussion of the topic in general. We would like to focus on one method in particular that we chose to deal with the sporadic non-numerical feature values: data imputation.

In general, data imputation consists of inferring the value of a particular instance of missing data based on the value of other *present* and ideally *related* instances. Imputing missing observations based on **related** observation values gives us a reasonably satisfactory estimate for the missing instance's true value since we use observations that are similar in our inference procedure.

As the reader will see below, the process of imputation may also leave us with specific features that are **degenerate**; put roughly, these are features which exhibit so much missing data across fragments that we simply cannot infer about and impute them and have no choice but to label them as degenerate. Such degenerate features are subsequently removed from the fragment feature matrices altogether.

We will now separate our discussion of the fragment feature matrix structures into a dichotomous discussion regarding two separate feature matrices FFM_A and FFM_I - the FFM for the fragments of the active molecules and the fragments of

the inactive molecules, respectively.

The following imputation procedure is therefore split into two steps: we will first impute FFM_A, and using particular metadata pertaining to this imputation procedure - such as the set of degenerate features found in FFM_A - we will subsequently impute FFM_I.

3.3.1 Imputation Step 1: Imputing FFM_A

Our imputation strategy in the molecular feature space for the FFM matrix containing only the active molecules is as follows: let us assume that the missing value is, without loss of generality, that of feature d_i in fragment f of active molecule a_0. In order to impute this feature, we first take the set of fragments $F_{a_0} \dots F_{a_n}$ of each *active molecule* $a_0 \dots a_n$ and then compute the *median* of the feature d_i for each set of fragments separately. If the computation of the median $Md_{a_{j_i}}$ for some active molecule a_j fails (i.e. if some of the fragments in this molecule also attain a non-numerical value for feature d_i), then we simply move on to the next active molecule a_k and re-attempt the computation of the median $Md_{a_{k_i}}$. Finally, we take this set of medians $Md_{a_{0_i}} \dots Md_{a_{n_i}}$ that were able to be computed, and compute a final **global feature median** value Md_{final_i} over this set of active molecule medians for feature d_i. We then set the global feature median as the value of our original missing feature instance d_i in fragment f of the molecule a_0. One final degenerate case we must consider is when atleast one fragment in every active molecule has a non-numerical value for the feature d_i under consideration, in which case $Md_{a_{0_i}} \dots Md_{a_{n_i}}$ are all undefined and hence the global feature median cannot be computed. In this scenario, we consider this feature a **degenerate feature** and remove the column corresponding to this feature from FFM_A.

The rationale behind the - admittedly rather complex - imputation strategy is that we are looking for key features within fragments of active molecules. The values of these features should not vary very much between some specific fragments of the active molecules - they should all fall within the ideal interval. Therefore, if we find a missing instance of a feature value for a fragment of an active molecule, a reasonable strategy would be to find the median for this feature over all fragments of all active molecules, and set this median as the value of the missing instance. By utilizing the median, we are striving to maintain any key feature value trends present in the actives' fragments when we impute the missing feature value. Of course, this may lead to some data contamination as certain unrelated fragments that don't contain any trend amongst the key features may start exhibiting similar values for these features, simply because they have all been imputed with a common median taken across all fragments of the actives. However, this is a calculated risk we take in order to detect as many veracious key features as possible at the risk of also finding some key features that exhibit false trends due to an exorbitant amount of imputation. As a final note, we chose to use the median statistic - as opposed to the arithmetic mean - for the imputation in order to avoid the case where an outlier value for a particular feature skews our calculations. The pseudocode of the algorithm is presented below for further clarity:

Algorithm 1: Imputation of the fragment feature matrix for the actives

function ImputeFFM_A(A matrix FFM_A):
Result: Fully imputed Fragment-Feature Matrix FFM'_A (possibly with less columns than FFM_A as a result of degenerate feature removal)
GlobalMedianCache = *empty array with slots available for global medians for each feature*
DegenerateFeatureSet = *empty array that will hold the set of removed degenerate features*
CalculateMetadata(FFM_A, GlobalMedianCache, DegenerateFeatureSet)
for *Molecule* a_j *in the set of active molecules* **do**
    F_{a_j} = *fragments that* a_j *contains*
    for *fragment feature vector* $f_i \in F_{a_j}$ *(a row of* FFM_A*)* **do**
        foreach *Non-numerical feature* d_j *of* f_i **do**
            SetFragmentFeature(f_i, d_j, GlobalMedianCache.getFeatureValue(d_j))
        end
    end
end
return FFM_A

Algorithm 2: Calculation of the global median cache and the degenerate feature set metadata

```
1  function CalculateMetadata(FFM matrix FFM_A, GlobalMedianCache,
   DegenerateFeatureSet):
     Result: FFM matrix FFM'_A with no columns corresponding to
             degenerate features, a filled GlobalMedianCache array, and a
             DegenerateFeatureSet array containing the removed degenerate
             feature indices
2  for Every feature d_i do
3      featureMedianArray = [ ]
4      foreach Active molecule m do
5          F_m = rows of FFM_A that correspond to fragments that are
           contained in molecule m
6          featureMedian = ComputeMedianOfFeature(F_m, d_i)
7          //One of the fragments of m has a non numerical value for the
           feature, and therefore featureMedian also becomes NaN, and we
           continue on to the next molecule
8          if featureMedian == NaN then
9              continue
10         end
11         else
12             featureMedianArray.add(featureMedian)
13         end
14     end
15     //If featureMedian was non-numerical for every active molecule, then
       the featureMedianArray for d_i is empty; this implies that we have a
       degenerate feature, and this feature will be removed from the matrix
16     if featureAverageArray is empty then
17         FFM_A.remove(d_i)
18         DegenerateFeatureSet.add(d_i)
19     end
20     else
21         GlobalMedianCache.setFeatureValue(d_i,
           featureMedianArray.median())
22     end
23 end
```

3.3.2 Imputation Step 2: Imputing FFM_I using already imputed FFM'_A

Having already covered our imputation strategy for the fragment feature matrix for the actives, the imputation strategy for the inactive molecules' respective fragment feature matrix will be considerably simpler. In order to identify the key features, one needs to be able to compare and contrast the fragment feature vectors of the active molecules with those of the inactive molecules; it follows that this type of analysis is only possible if the fragment feature matrices of the active molecules and inactive molecules contain the *same features* in the

columns. If this weren't the case, then column i in the actives' FFM could correspond to a completely different physiochemical feature than column i in the inactives' FFM - making any comparisons between fragments of active molecules and inactive molecules downright impossible. Therefore, put concisely, a feature can be represented in the columns of the active molecules' FFM if and only if it is also represented in the same column of the inactive molecules' FFM. This is precisely the reason why we first complete our imputation of the active molecules' FFM before moving on to the inactive molecules' FFM. We may use the metadata from the actives' FFM - namely the global median cache and the degenerate features - to impute the inactive molecules' FFM while ensuring that we end up with the same exact physiochemical features represented in both matrices afterward.

Using the observations in the previous paragraph, we will see that imputing the inactive molecules' FFM is quite trivial. The first step is to remove any columns corresponding to the degenerate features that were found while imputing the active molecules' FFM - since these degenerate features don't appear in the imputed FFM'_A, they cannot appear in the imputed FFM'_I either, as observed in the prior paragraph. We then traverse through each row in the inactive FFM, and if we find a row that contains a non-numerical value for some feature d_i, we simply set this value to the corresponding feature value for feature d_i in the *global median cache* from the actives' imputation. One might argue that using the same global median feature values, computed solely from the actives, to also impute the inactives may lead to an artificial similarity between the values of the active and inactive fragments, and therefore a failure to identify some key features. While this is true, we argue that this procedure is still usable in our scenario: ideally, the number of non-numerical feature values present across all the inactive fragments is relatively low, and therefore performing a simple imputation step that saves us a lot of time need not necessarily do much harm. Indeed, consider that usually the number of inactive molecules - and by extension inactive fragments - with respect to a target is vastly greater than the number of active molecules. Therefore, performing a global median operation for the inactive molecules as we did with the active molecules in the previous stage could take a considerably larger amount of time.

Algorithm 3: Imputation of the fragment feature matrix for the inactives

```
function ImputeFFM_I(Fragment feature matrix FFM_I,
GlobalMedianCache, DegenerateFeatureSet):
    Result: Fully imputed Fragment Feature Matrix FFM'_I (possibly with
            less columns than FFM_I, again due to removal of columns
            corresponding to degenerate features)
foreach Feature d in DegenerateFeatureSet do
    RemoveDescriptorColumn(FFM_I, d)
end
for Molecule i_j in the set of inactive molecules do
    F_{i_j} = fragments that i_j contains
    for fragment feature vector f_i ∈ F_{i_j} (a row of FFM_I) do
        foreach Non-numerical feature d_j of f_i do
            SetFragmentFeature(f_i, d_j,
            GlobalMedianCache.getFeatureValue(d_j))
        end
    end
end
return FFM_I
```

3.4 Dimensionality reduction in the molecular feature space

3.4.1 General dimensionality reduction

It is at this point that we combine the two imputed fragment feature matrices FFM'_A and FFM'_I into one *combined* fragment feature matrix FFM_C, as mentioned earlier.

As with all problems involving so-called *Big Data*, it is of paramount importance to discern which subsets of the data are important in enabling a deeper insight, and which subsets are redundant, or even decoys, during our analysis procedures. Such isolation of the most important subset of the data will prevent the multifarious troubles associated processing extremely large data sets. Indeed, without an appropriate trimming of the input dataset, any algorithm that we apply will naturally require a lot of computational effort; moreover, most of this effort will be wasted on processing unimportant data. A perhaps more insidious trouble is that many algorithms will return sub-optimal results if they are forced to train themselves on too much noisy data, and we as the users of the algorithm will assume that these sub-optimal results are due to the choice of the algorithm rather than an improper distilling stage for the dataset. An archetypal example of the subtle but harmful effects of unclean datasets is the so-called *curse of dimensionality*[13], which occurs because of the distorted meaning of distance when working in very high-dimensional Euclidean or metric spaces. The curse may indeed pertain to our input dataset: we have approximately 1500 physiochemical features for each fragment of each active or inactive molecule with respect to the target. Without proper preprocessing, therefore, we will work in a 1500-dimensional Euclidean space. We, therefore, present our method for dealing

with these dimensionality-related problems in the molecular feature space.

In data mining, most techniques for dimensionality reduction rely on removing specific features that are heavily correlated with other features in the dataset, thereby minimizing the amount of redundant information present in the data. In the machine learning community, the generally used and widely accepted way to deal with large dimensionality is to use a technique known as *principal component analysis*[14]. Put briefly, PCA projects a set of vectors from a dimension $\mathbb{R}^d$ to dimension $\mathbb{R}^k$, where $k < d$. This is done by expressing the vectors as *linear combinations* with respect to a basis formed by taking the eigenvectors of the covariance matrix of the dataset (for a much more rigorous description of the procedure we refer the reader to [14] or the vast amount of literature on the topic). In our setting, however, *this method may lead to undesirable consequences*. To understand why, consider that the vectors - fragment feature vectors in our setting - are originally expressed in terms of the standard orthonormal basis $\{e_1 \dots e_d\}$, where each dimension e_i is an abstraction corresponding to a specific *feature* in the chemical space. By projecting the vectors onto a different basis that has no natural physical meaning - unlike the basis $\{e_1 \dots e_d\}$, which corresponds directly to physiochemical features - we ameliorate our problem of dealing with very high dimensional data, but lose any direct chemical meaning of the coordinates of the fragment feature vectors. We, therefore, use a different approach for dimensionality reduction in our setting; this approach uses an apparatus similar to the covariance matrix - namely the *correlation matrix*.

3.4.2 Correlation neighborhoods

Let $C \in \mathbb{R}^{d \times d}$, where d is the original number of physiochemical features (before any dimensionality reduction), denote the *correlation matrix* of the dataset. In our problem setting, the correlation matrix is a decidedly better tool than the covariance matrix to measure the level of correlation between different features. This is due to the fact that different chemical features span different ranges in terms of their values; indeed, our features vary in magnitude from the order of 10^-3 to 10^3. As a consequence of such disparate value ranges, the covariance matrix would be a very inaccurate metric to determine the level of correlation between two features since there is no standardization of the covariance values - unlike the correlation matrix, which is standardized across value ranges.

Using the correlation matrix C, one can view the set of features as *vertices in a graph*, with edges between the features ft_i and $ft_j \iff C_{i,j} >= \epsilon$, where ϵ denotes a *correlation threshold* (For the original idea regarding this abstraction, we refer the reader to [31], here we will present it again in full and discuss its applications). The correlation threshold, ϵ, can be arbitrary - but of course, it would make sense to set it to a number $\geq .50$, which semantically indicates that two features have an edge between them if and only if they are highly positively correlated. Strictly speaking, a vertex should also have a loop edge (an edge going from itself to itself) since all features are perfectly correlated with themselves; for the purposes of our algorithm, however, we will not consider loop edges.

Let us call this newly created graph $G = (V, E)$ the *correlation graph*; the correlation graph can be considered a collection of *connected components*, with some singular vertices that correspond to features that don't exhibit high corre-

lation with any other features. A visual depiction of such a correlation graph is shown in Figure 3.2

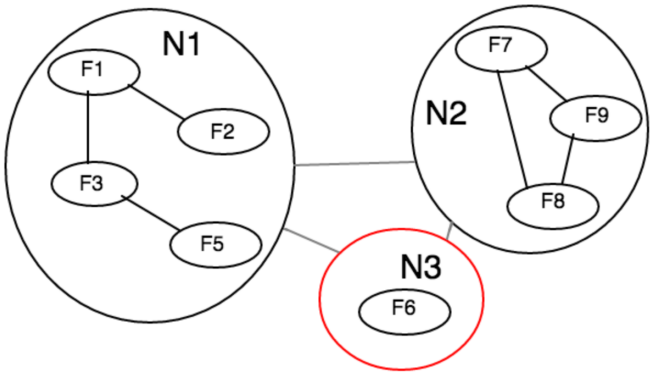

Figure 3.2: A visual representation of the correlation graph. Observe that there are 2 *correlation neighborhoods* containing features which are strongly correlated with each other - N1 and N2 - and one correlation neighborhood N3 containing a solitary feature. The gray edges in between the neighborhoods represent the low correlation coefficients between two feature vertices in two different neighborhoods (e.g. F1 and F7), whereas the black edges in between feature vertices in the same neighborhoods (e.g. F1 and F2) represent large correlation coefficients above our correlation threshold ϵ. The feature corresponding to the vertex contained in N3 - F6 - exhibits a correlation coefficient below the threshold with respect to all other features, except itself (we don't consider loop edges in the correlation graph). Therefore, the neighborhood N3 is colored red, as F6 is fairly independent of all the other features, and will, therefore, be added to the output feature set as it is not redundant. On the other hand, for each of the other neighborhoods, we will choose a *representative vertex* whose corresponding feature will be included in the output feature set.

Let us assume we already have some initial members in our final list of non-redundant features in the form of the features corresponding to the singular vertices in the correlation graph. Indeed, the features corresponding to these vertices don't show high correlation with any other features; we, therefore, must take them into our final set of features, as they must contain valuable information not captured anywhere else in the feature set.

We then proceed to take this initial set of non-redundant features and add a *subset* of the remaining features from the correlation neighborhoods. This combination of completely uncorrelated features with a subset of the correlated features minimizes the redundancy while simultaneously preserving as much information present in the data as possible. Since we only choose a subset of the correlated features, we minimize the redundancy; at the same time, we preserve all the uncorrelated features and hence ascertain that we are not losing any valuable insight that is not replicated elsewhere in the feature set. Formally, we take the *union* of the completely uncorrelated features with a set of several *representatives* from each correlation neighborhood in the correlation graph. The algorithm, presented below, should shine a clearer light on our procedure for mining the non-redundant subset of features from the full set of features:

Algorithm 4: Finding the set of non-redundant features using neighborhood representatives

```
function IdentifyNonRedundantFeatures(Correlation matrix of the dataset C, ϵ): Result: Array of non-redundant features f_1 ... f_n in the dataset
NonRedundantFeatures = {Initial array of all the features (both redundant and non-redundant)}
G(V, E) = GetCorrelationGraph(C,ϵ)
v_i = max{v ∈ G, δ(v)} // Choose vertex with largest degree
// While there still exist some neighborhoods - i.e. we don't only have singular vertices
while δ(v_i) ≠ 0) do
    N(v_i) = set of neighbors of v_i
    for v_j ∈ N(v_i) do
        V = V \ v_j // Remove all vertices in the neighborhood of v_i
        NonRedundantFeatures.remove(v_j.feature) // Remove the feature corresponding to the vertex v_j from the set of non-redundant features
    end
    v_i = max{v ∈ G, δ(v)} // Initiate the next iteration of the loop
end
return NonRedundantFeatures
function GetCorrelationGraph(Correlation matrix C,ϵ):
Vertices = {empty set of vertices}
Edges = {empty set of edges }
for feature f_i in C do
    CorrelatedFeatures = {set of features f_j such that C_ij ≥ ϵ AND j ≠ i}
end
Vertices.add(v_i) // add new vertex corresponding to this feature
if CorrelatedFeatures is not empty then
    for f_j in CorrelatedFeatures do
        Edges.add((v_i, v_j)) // Add edge between correlated features in a neighborhood
    end
end
return {Vertices, Edges}
```

3.5 Normalization

We have one final step in our data purification pipeline: *normalization*. Statisticians employ normalization to enable comparison of data that occupy different value ranges. Such a procedure is incredibly relevant in the molecular feature space: different physiochemical feature values may be 2 or 3 orders of magnitudes larger or smaller relative to each other. This has hitherto not been a problem, as we have only created the fragment feature matrix; we will, however, start to experience serious problems when utilizing the current feature values in machine learning algorithms without proper normalization. It would be futile to attempt to explain to the reader why this is the case at this point in time, as a complete explanation would require us to reveal details about the actual algorithms we will be applying on the fragment feature matrix FFM_C - a discussion which we will save for the near future. For now, we ask the reader to accept that, as with many other algorithmic schemes in the fields of data mining and machine learning, normalization is a necessary final step in our data cleaning procedure. We outline formally the normalization method performed on FFM_C:

Algorithm 5: Normalization of the Fragment Feature Matrix

```
1  function NormalizeFeatures(Fragment Feature Matrix FFM_C):
   Result: Fully normalized Fragment Feature Matrix FFM'_C
2  foreach Column c in FFM'_C do
3  |  columnMaxValue = maximum(c)
4  |  columnMinValue = minimum(c)
5  |  // Note that columnRange can never be 0, as this would mean that
   |     this feature is a constant valued feature, whereas we already removed
   |     all constant valued features from our feature matrices in earlier steps
6  |  columnRange = columnMaxValue - columnMinValue
7  |  foreach Value v in c do
8  |  |  v = (v - columnMinValue) / columnRange
9  |  end
10 end
```

As the reader can easily deduce from the algorithm, we constrain the values of FFM'_C to the interval $[0, 1]$ by dividing each feature value of each fragment by the *range of the feature* over all fragments. This enables us to make values from different columns comparable since the values no longer differ in orders of magnitude, but are rather all constrained to the same interval. This will greatly empower us when we aim to glean insight about the key features in the molecular feature space.

4. The Subspace Clustering Approach

4.1 Introduction

There exist numerous algorithms covering the problem of finding subspace clusters that each have different advantages and disadvantages when compared with one another. We, therefore, present a brief overview of these various algorithms and discuss which of them would be most appropriate to use in the molecular feature space.

4.1.1 The density of a neighborhood of a point

We already implicitly presented the idea of density earlier when defining a cluster and a subspace cluster, by discussing this idea in the context of a specific fragment feature vector having μ many points in some *dense* ϵ neighborhood of it in the molecular feature space. Briefly restating what we already discussed in a more general context, we say that the neighborhood of a point $\mathbf{x}$ in some Euclidean space is **dense** if, for some parameters ϵ and μ, there exist μ many other points in the ϵ-ball around $\mathbf{x}$. This notion is very important when constructing and understanding clustering algorithms, namely it is key to understand that these sets of dense collections of points - clusters - are separated by regions of low-density collections of points in the Euclidean space. This is a key idea utilized by numerous cluster detection algorithms, as will be seen very soon.

4.2 Basic Clustering Algorithms

Before we move on to algorithms that detect subspace clusters, we first present two algorithms that detect general clusters - that is, clusters defined over all dimensions. The rationale behind this order of analysis is that the subspace clustering algorithms that we will discuss later heavily utilize the fundamental ideas that the more general clustering algorithms are built upon. Consequently, by first analyzing general clustering algorithms, we will have a foundational understanding of the tools of the trade before commencing the study of the more advanced subspace clustering algorithms. We first present the most quintessential density-based clustering algorithm: **DBSCAN**. DBSCAN accepts two parameters, ϵ and μ (where μ is also referred to as *minpts*), and partitions the point vectors in our dataset into 3 groups, core objects, density-reachable objects, and noise objects. [15]

Very briefly stated, the algorithm passes through the dataset and performs ϵ sized neighborhood queries for each point; if a point p has n many other points $o_1 \dots o_n$ in an ϵ-sized neighborhood around it, where $n \geq \mu$, it is deemed to be a core object and the points $o_1 \dots o_n$ are added to this new core object's cluster. Moreover, $o_1 \dots o_n$ are said to be *density reachable* from the core object p; further neighborhood queries are subsequently executed on these density-reachable points to augment the new cluster with further *indirectly* density-reachable points -

known as the *cluster expansion* phase. Points that do not have at least μ other points in their ϵ neighborhood *and* are not density-reachable - neither directly nor indirectly - via another core object are known as *noise* objects. A visualization of some of the above notions can be seen in Figure 4.1. We see that DBSCAN nicely captures the notion of density-based clusters as it asserts that a point p has a *dense* neighborhood around it before it is added to the list of core objects that form clusters. DBSCAN is also surprisingly efficient: the main bottleneck in speed is encountered when performing the ϵ-sized range queries over the points of the database - although even this stage can be optimized with an appropriate spatial indexing structure.

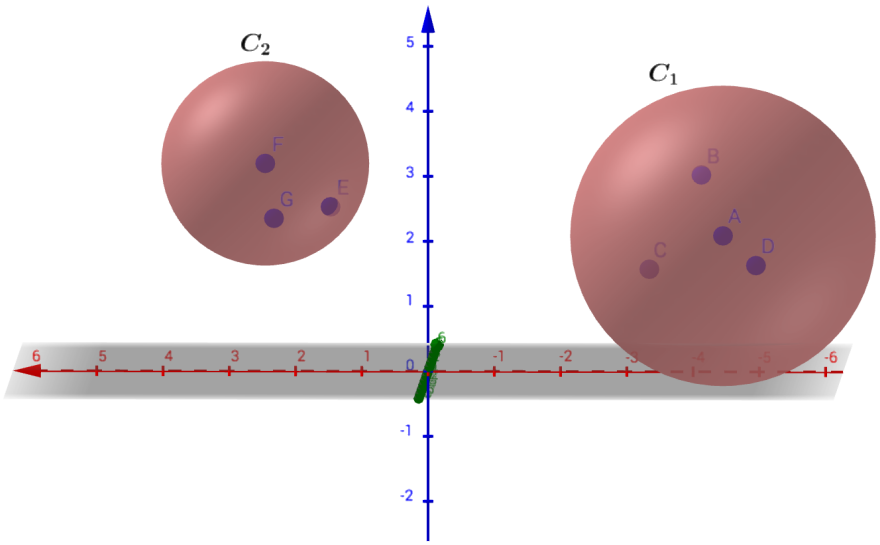

Figure 4.1: The above image gives a visual representation of an ϵ-sized range query done with two different center points: A and F, with $\epsilon = 2$. One can quickly see that one ϵ-sized ball, labelled C_1, is quite dense, while the other is quite sparse. If we assume that μ is set to 3, then C_1 would be considered a cluster while the other ball would be discarded as noise. We can quickly see the effect that ϵ and μ have on the notions of density and by extension what *balls* constitute an actual cluster in the d-dimensional Euclidean space.

DBSCAN, however, exhibits some problems when attempting to find clusters of different densities, for which no single value of ϵ or μ would be adequate. In order to cope with such scenarios, improvements to DBSCAN have been presented that remove the need to specify a strict ϵ. The **OPTICS** algorithm, for example, accepts a parameter ϵ' known as the *generating distance*, as well a *minpts* parameter μ. OPTICS is more flexible than DBSCAN because it requires only a *generating* distance parameter ϵ', rather than a *strict* distance parameter ϵ. This allows us to set ϵ' as the *maximum radius* that we expect to encounter for some cluster in our dataset if we were to hypothetically run DBSCAN. [16]

More concretely, OPTICS can be viewed as an extension of DBSCAN in the sense that, similar to DBSCAN, it makes ϵ sized range queries iteratively through the dataset. OPTICS differs from DBSCAN, however, due to its utilization of a key data structure known as the *reachability distance plot*. The reachability

distance plot is essentially a linear ordering of the points in the database; this ordering starts with some origin point o_0, and then iteratively adds points $o_1 \dots o_n$ to the ordering based on the *minimum reachability distance* of a new point o_i from any existing point o_j, $j < i$, in the reachability plot during a particular iteration. The reachability distance (for which a formal definition can be found in [16]) is very informally the *minimum* $\epsilon'' \leq \epsilon'$ such that o_i is *directly* density-reachable from some o_j already present in the reachability plot. Therefore, during a particular iteration, the algorithm adds a new point o_i to the ordering if and only if out of all the candidate points, o_i minimizes the reachability distance from some existing point o_j. The algorithm then proceeds by generating further candidate points to add to the reachability plot in the next iteration via an ϵ'-sized range query on the newly added point o_i. The profound consequence of such a procedure is that one may visually represent the clusters in the dataset by looking for *low reachability distance valleys* in the reachability plot - Figure 4.2 offers a clear visualization of how this can be done.

Since OPTICS adds points to the reachability plot based on the minimum reachability distance $\epsilon'' \leq \epsilon'$, we are able to detect both *wide* and *narrow* clusters corresponding to different values of ϵ''. Given the reachability plot returned by OPTICS, it is, of course, imperative that we extract the actual clusters from the data structure. This is done by correlating clusters with the regions of low reachability distance, or valleys, of the reachability plot, and marking all the points in a specific valley as a single cluster. Several such algorithms exist to extract clusters from the reachability plot, and for an in-depth discussion on these methods we refer the reader to [16]. A final observation regarding OPTICS is that the notion of a reachability distance can be modified to fit any *metric* distance function: this idea will be incredibly useful when we analyze extensions of OPTICS to subspace clustering.

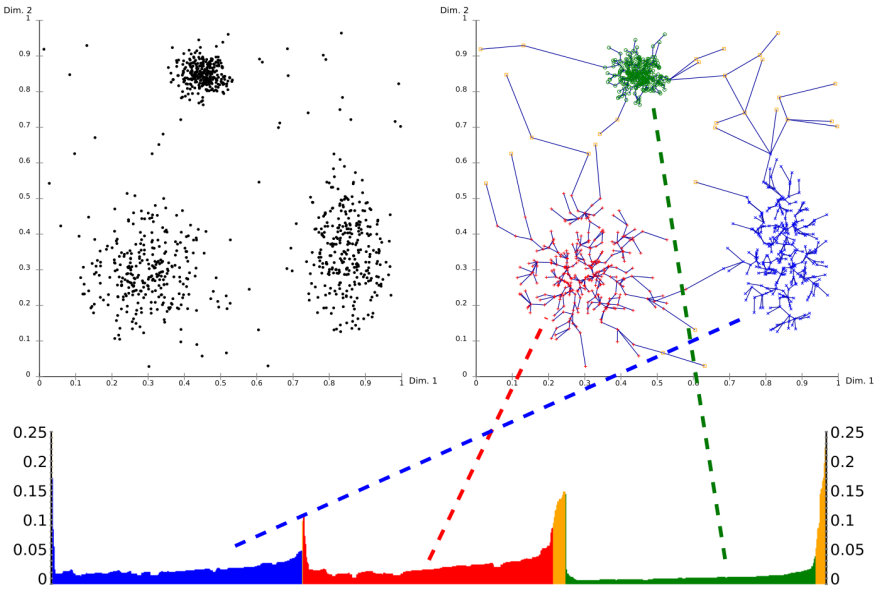

Figure 4.2: A visual depiction of the reachability plot utilized by OPTICS elucidates the different clustering distributions within the dataset. As shown by the image, the *valleys* in the reachability plot - or the regions of *low* or *decreasing reachability distance* - correspond to the different clusters in the dataset. On the other hand, the regions of increasing or high reachability distance correspond to the *low density zones* that are encountered in between clusters.

Although OPTICS provides a fairly robust solution to the crippling problem of finding an appropriate value for ϵ in DBSCAN, neither DBSCAN nor OPTICS specify a way to directly detect *subspace* clusters, which is what we require in the molecular feature space.[1] DBSCAN and OPTICS, however, provide an important foundation for which more advanced subspace clustering algorithms are built. DBSCAN captures precisely the definition of *density* mentioned earlier, and OPTICS introduces the notion of a *minimum reachability distance* between two points; both ideas will remain very important as we move on to our analysis of subspace cluster detection algorithms.

4.3 Subspace clustering algorithms

We now move on to a short discussion of subspace clustering algorithms, and identify two algorithms whose properties indicate that they may be appropriate for application in the molecular feature space. We will focus our discussion only on these two algorithms, but leave the reader with an abundance of algorithms for subspace cluster detection to review at his or her leisure [17][18][19][20][21].

Important Note: Both of the following algorithms that we will present are only suitable for finding *axis-parallel* subspace clusters.[20] One might postulate, therefore, that these algorithms will return incomplete results as they fail to

[1] We say *directly*, since one can always create an indirect scheme for detecting subspace clusters using DBSCAN or OPTICS via projecting the vectors into several permutations of subspaces, and running DBSCAN or OPTICS on these projected vectors to obtain subspace clusters. But these procedures are not particularly clever, nor are they computationally efficient.

detect certain subspace clusters that are not completely axis-parallel. While this may lead to erroneous results in other fields, we claim that the detection of solely axis-parallel subspace clusters suffices in our application in the molecular feature space. Indeed, we require only clusters of fragment feature vectors with similar values across certain key features to be detected; axis-parallel clusters dutifully capture this notion by ensuring that on some dimensions (corresponding to the key features), the values of the vectors contained in the cluster fall within the ideal interval. We illustrate the prior statement regarding the nature of axis-parallel clusters in Figure 4.3.

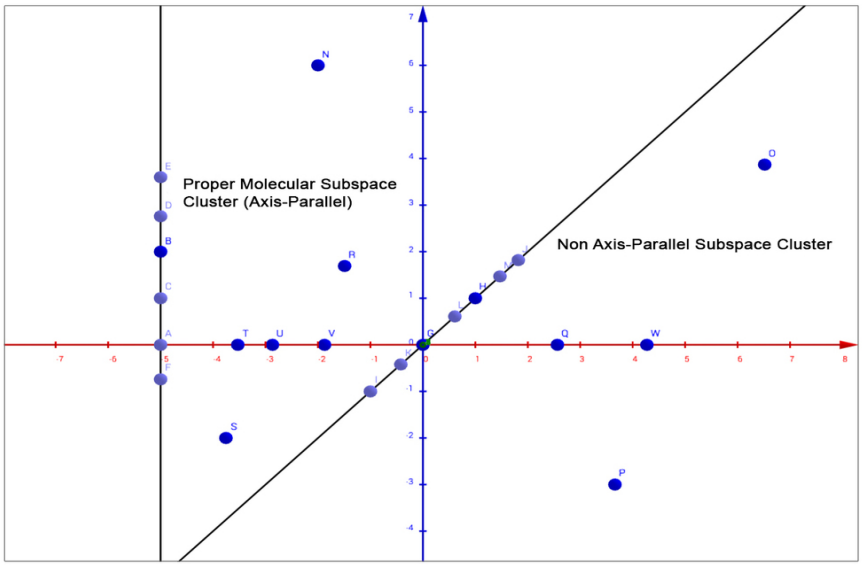

Figure 4.3: The figure above shows two potential subspace clusters projected into the $\mathbb{R}^2$ Euclidean space; one - let us call it S_{AP} - is axis-parallel, while the other - S_{NAP} - is not. All the vectors contained in S_{AP} have x-axis coordinates that fall within some ideal interval but y-axis coordinates that are fairly randomly distributed, leading to a subspace cluster that is *axis-parallel* to the y-axis. S_{NAP}, on the other hand, contains vectors whose coordinates fall within some larger interval in both the x-axis and the y-axis, but are more dispersed than the vectors in the first cluster in both axes. If we imagine the two axes as representing different physiochemical features, then we could veraciously claim that there is a clear trend within S_{AP} with respect to the feature corresponding to the x-axis. On the other hand, such a claim can not be made for S_{NAP} for neither the x nor the y axis since it is not axis-parallel in either axis. Therefore, S_{NAP} doesn't exhibit rigid trends in any particular dimension, and is of minimal use in our applications. We conclude, therefore, that for our purposes in the molecular feature space, searching for axis-parallel subspace clusters suffices.

We will, therefore, focus on the analysis of axis-parallel subspace clustering algorithms in the molecular feature space. There do exist, however, algorithms for finding subspace clusters in a more general setting; as a starting point, we refer the reader to [18].

4.3.1 HiSC

HiSC - which stands for Hierarchical Subspace Clustering - is quite a robust algorithm that determines subspace clusters using a number of properties that are quite relevant in the molecular feature space. The main key principle of HiSC that not only differentiates it from its many predecessor algorithms in the space of subspace clustering, but also makes it very usable in the molecular feature space is its ability to determine a *hierarchy* of subspace clusters.[19] This ability is important because of the somewhat contrived but nevertheless illustrative example that is presented in the following visual in Figure 4.4. The visual illustrates the case where there are two axis-parallel lines that form subspace clusters in the 1-D space. These two axis-parallel lines, however, are in fact just subsets of a higher dimensional subspace cluster - the axis parallel hyperplane in 2-D.

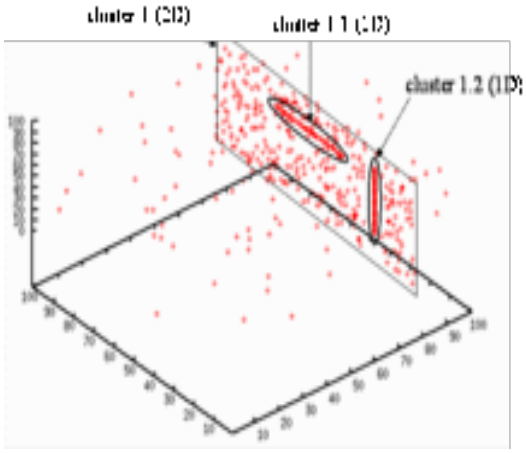

Figure 4.4: **Hierarchies of subspace clusters.** We have two axis-parallel lines that form subspace clusters in the 1-D space of the ambient 3-D space - one clustered along the x-axis and parallel to the y-axis, and one clustered along the y-axis and parallel to the x-axis. These two axis parallel subspace clusters, however, are in fact just subsets of a higher dimensional cluster - the axis parallel hyperplane, or a 2-D subspace cluster. [33]

HiSC can accurately detect such nested hierarchies of subspace clusters, making it quite useful - especially in our application domain. Indeed, in the molecular feature space defined over the set D of physiochemical features, we could have two separate subspace clusters C_1 and C_2 of fragments that are defined in the subspaces $S_1, S_2 \subset D$, respectively. However C_1 and C_2 could also be subsets of an *encompassing* subspace cluster C in the subspace $T \subset D$, where $S_1 \cup S_2 \subseteq T$. In this case, many interesting analyses could be done. For example, recalling the *purity* and *diversity* conditions mentioned in prior sections, it could very well happen that C itself may not be a significant cluster because it is not pure enough, but we may proceed down the hierarchy and find that C_1 and C_2 actually satisfy the purity conditions. Therefore, this could lead to an interesting situation where C_1 and C_2 could be significant (assuming they both also satisfy Condition 1 - Diversity) even though their *parent* cluster in the hierarchy, C, is not. As a further step, by observing the point in the hierarchical ladder when adding more subspaces causes purity to diminish below a certain threshold, we

can gain enormous insight into a **maximal set of key features** that correspond to pure clusters. In conclusion, it is highly valuable for both present and future research to find and store the entire subspace clustering hierarchy - which is why we deem HiSC as a standout candidate for use in the molecular feature space.

We now proceed to a short analysis of the HiSC algorithm itself - HiSC is in fact very similar to the OPTICS algorithm we saw earlier. Indeed, HiSC analogously walks through the dataset and creates a reachability plot based on the minimum reachability distance between already reached vectors and not yet reached vectors. The key differentiator between the two algorithms is that HiSC **redefines the reachability distance function to suit subspace clustering**. We mentioned earlier that the distance function employed in OPTICS could be arbitrarily chosen, with the caveat that it obeys the standard properties of a metric distance function. HiSC indeed chooses a valid distance function obeying the metric properties, while simultaneously capturing the notion of a subspace cluster in the Euclidean space. We start our analysis of this new distance function with the following definition:

Subspace preference vector. In order for HiSC to detect the hierarchies of subspace clusters, it assigns to each point p what is called a subspace preference vector w_p, which is a bit vector of size d. To determine the vector, we first perform a *k-nearest neighbor search* on p, where k is an input parameter, obtaining points $o_1 \dots o_k$. We then take the *average variance of the distance from p* of the neighborhood of points $o_1 \dots o_k$ in a local neighborhood of p in the attribute A_i and denote this value as $Var_{A_i}(p)$ using the following formula, where $\pi_{A_i}(p)$ is the projection of point p to the subspace A_i:

$$Var_{A_i}(p) = \frac{\sum_{o_i \in o_1 \dots o_k} (\pi_{A_i}(o_i) - \pi_{A_i}(p))^2}{|o_1 \dots o_k|} \tag{4.1}$$

HiSC then defines the subspace preference vector w_p for p as follows, for some threshold α, also given as an input parameter:

$$w_{p_i} = \begin{cases} 1 & Var_{A_i}(p) \leq \alpha \\ 0 & Var_{A_i}(p) > \alpha \end{cases}, i = 1 \dots d \tag{4.2}$$

We intuitively set the coordinate of the preference vector corresponding to a specific attribute A_i to 0 if there is a high degree of average variance in the distance of the k-nearest neighbors to the point p in that attribute, and set the coordinate to 1 otherwise. The *basic* reachability distance function, or subspace distance function, between two points p and q is then defined as the **number of zero values** in the *logical and* $w_p \wedge w_q$ of the two subspace preference vectors of the points p and q (this is actually a slight simplification: for the full details regarding the subspace distance, we refer the reader to the original paper [19]). This customized distance function intuitively captures the difference in subspace dimensionality between the two points - the more attributes A_i that both p and q exhibit low variance in, the lower the reachability distance.

At the end of this procedure, we can, similarly to OPTICS, create the reachability plot using the modified reachability distance and extract the subspace clusters and their corresponding subspaces from the reachability plot - giving us our much sought after key features.

4.3.2 DiSH

We now turn to the final subspace clustering algorithm that we will be discussing, and the one that we will be actively using: DiSH. [20] DiSH builds on HiSC by fixing a few severe flaws. One of the key pitfalls of HiSC is that it assumes that if a point p belongs to a subspace cluster C in some projection of the full ambient space to a specific subspace, then the points of C must be visible in the neighborhood of p in the *entire* ambient feature space. This pitfall can be easily seen by examining one of the initial phases of HiSC, namely the phase in which it constructs the preference vectors: we compute the k-nearest neighbors over the *whole feature space*, but then subsequently compute the average variance of the k-nearest neighbors on only an *attribute (or feature) basis*. If the points of a subspace cluster C that p belongs to are not visible from the k-nearest neighbors search in the *entire* feature space, then the per-attribute variance will be misleading and p's subspace preference vector will be computed incorrectly and will beget inaccurate results.

DiSH remedies such scenarios by performing *attribute wise* neighborhood queries instead of feature space wide neighborhood queries - which is the main reason why it is much more attractive in our usage in the molecular feature space. This difference between DiSH and HiSC can be seen in the redefinition of the subspace preference vector of a point.

DiSH redefines the preference vector of a point p in the following way: for each attribute A_i, it performs an ϵ sized neighborhood query around the point p *in the attribute* A_i, where ϵ is an input parameter. If the number of points in this single-dimensional neighborhood around p is greater than or equal to μ, another input parameter, we add this attribute to a list of *candidate attributes* that could be a part of the dimensions of a subspace cluster containing p.

After determining all the initial candidate attributes for a point p by a series of neighborhood queries, the candidates need to be consolidated in an optimal way so as to obtain the attributes that most likely constitute the dimensions of a subspace cluster that point p belongs to. To understand the manner in which these optimal subspace attributes are chosen, we refer the reader to [20], where two schemes are proposed: one using frequent itemset mining (another popular algorithm in the field of data mining) and the other using a simpler and more computationally efficient heuristic. For our purposes in this paper, it suffices to assume that we are given the optimal set of attributes of a subspace cluster containing a point p and denote this set S_p. The subsequent construction of the subspace preference vector w_p is then quite simple: for a point p and the set of attributes S_p, we define the subspace preference vector of p, w_p, as:

$$w_{p_i} = \begin{cases} 1 & A_i \in S_p \\ 0 & A_i \notin S_p \end{cases}, i = 1 \dots d \tag{4.3}$$

The subspace distance of two points p and q is then defined, similarly to HiSC, as the *number of 0s* in the vector $w_q \wedge w_p$ (this is again a slight simplification: for the full details regarding the subspace distance, we refer the reader to the original paper [20]). Therefore, the only thing that changed from HiSC's definition of a subspace distance between two points p and q is the way the original subspace preference vectors w_p and w_q are computed. The familiar walk through

the dataset can then be computed, exactly as it was with OPTICS and HiSC, using our new distance function, and we may subsequently obtain a reachability plot just as before. The clusters can then, once again, be retrieved from this reachability plot.

It's worthwhile to pause for a moment and consider the beauty of the distance function abstraction. By simply modifying how we formed the preference vectors - and by extension the formulation of the distance function - we completely change the type of clusters obtained as we perform our walk through the dataset. We have thus far seen three different distance functions (one for OPTICS, one for HiSC, and yet another one for DiSH), that all aim to capture different logical notions with respect to the type of cluster we are trying to identify. This is a very useful abstraction for any further research work that aims to be done in this area: one simply needs to define a valid metric distance function that captures the properties of the clusters he or she wants to identify, and then simply compute a walk through the dataset using this distance function.

In the molecular feature space, we aim to capture *significant* clusters with specific properties - namely diversity and purity. A point of further research, therefore, could be the design of metric distance functions with properties that optimize for the retrieval of significant clusters. In this paper, however, we will focus on utilizing the clustering algorithms that have already been engineered for more general cases without modification.

4.4 Filtering of the found subspace clusters

From the subspace clustering algorithms presented previously, we get significant subspace clusters as well as *noisy* subspace clusters (clusters that do not satisfy our previously defined **Significant Cluster** conditions). It is, therefore, imperative to devise a scheme that sequesters the significant clusters from the noisy ones, based on our definition of significance. We will now showcase, for each condition of significance, an algorithm that filters those clusters which do not satisfy the respective condition; after all the subsequent filtering algorithms have been applied, therefore, we will be left with solely significant clusters.

Filtering Stage 1: Diversity The algorithm for filtering out clusters that are not diverse (i.e. clusters that contain fragments from only a few active molecules), is fairly straightforward. We simply define a threshold α and then iterate through the list of clusters; for each cluster c_i, we compute the *number of active molecules* which contain *any* fragments that are contained in that cluster and denote this number as β_i. If $\beta_i < \alpha$ for some i, we remove cluster c_i; otherwise, we conclude that c_i is diverse enough to pass to the next filtering stage. The threshold α will be set as a *percentage* of the number of active molecules - and therefore $0 \leq \alpha \leq numActiveMolecules$.

Filtering Stage 2: Purity The algorithm for filtering clusters based on purity is similar to the algorithm for filtering clusters based on diversity. We accept a threshold parameter $\alpha \in [0, 1]$, and then iterate through the set of all found clusters and remove any clusters that don't satisfy $\frac{|A \cap C_f|}{|A \cap C_f| + |I \cap C_f|} \geq \alpha$, where A is the set of all fragments found only in active molecules, I is the set of all fragments found in at least one inactive molecule, and C_f is the set of all

fragments within the cluster.

The filtering is made slightly more complex, however, by the fact that there could exist substantially more inactive molecules than active molecules in the dataset. Subsequently, there would also exist many more inactive fragment vectors than active fragment vectors in our feature space, since every molecule contains several fragments. This is indeed the case in most real life datasets and therefore requires a slight adaptation of our prior condition for removing a cluster. For example, if we set the threshold α too high and there exist many more inactive fragments than active fragments, then we may have the case where a cluster is actually significant but gets filtered out simply because there are many more inactive fragments in the dataset and therefore the ratio $\frac{|A \cap C_f|}{|A \cap C_f| + |I \cap C_f|}$ is naturally lower. We, therefore, protect our algorithm from such effects by redefining α as α':

$$\alpha' = \begin{cases} \alpha & |I| - |A| \leq 10 \\ \frac{\alpha}{(1+\log_{10}(|I|-|A|)} & |I| - |A| > 10 \end{cases} \tag{4.4}$$

The formula above can be understood as follows: if the number of inactive fragments is *less than* the number of active fragments or greater than the number of active fragments *by less than 10*, then we simply keep the current purity threshold. Since there are more active fragments than inactive fragments in the dataset, or the amount of inactive fragments is only marginally greater, we don't need any adjustment to the purity threshold. On the other hand, if the number of inactive fragments is greater than the number of active fragments by 10 or more, we set the value of the purity threshold correspondingly lower - via the logarithm function - to account for the disparity in the amounts of the two types of fragments. To see how this adjustment is done, one can observe that the denominator $1 + \log_{10}(|I| - |A|)$ in the latter case will get correspondingly larger as the difference between the number of inactive fragments and active fragments increases, leading to a smaller and smaller value for the purity threshold α. This step will be done completely automatically without the user's knowledge, as we do not expect the user to know the distribution of the active and inactive fragments in his/her data set and set α accordingly. Accounting for some innate, admittedly rather esoteric, properties of the dataset is, therefore, a task delegated to our system and abstracted from the user.

In conclusion, we have designed two schemes that filter the significant clusters from the degenerate clusters quite effectively, while also keeping in mind the potential problems that may arise due to the intrinsic properties of the dataset with respect to the number of active and inactive fragments.

4.5 The Key Feature Model

4.5.1 Choosing the Best Cluster with Validation

In our problem setting, we aim to create a **machine learning model** learned based on an input **training set** of active and inactive molecules with respect to some macromolecular target.

We wish to create, for a specific target molecule, a model that captures its key features and their values. Such a model should aid us in *scoring* a new candidate

molecule based on its activity or inactivity with respect to the aforementioned target. Such a scoring mechanism is important because, in virtual screening, one is given a list of candidate molecules, or leads, and is tasked to *rank* the candidates so that the active molecules, in general, are ranked higher than the inactive molecules. A pharmaceutical researcher would then extract only the top ranked candidates, ideally mostly active molecules, for further testing. A scoring function for new candidates is, therefore, imperative for us to obtain such a ranking, and therefore in this section we present an algorithm for scoring new candidate molecules for a specific target.

We first consider each obtained cluster in a specific collection of subspace clusters returned by DiSH as a **key feature model**. These key feature models represent the varying combinations of key feature value ranges which could lead to activity with respect to the current target. To gauge a new molecule's active nature, therefore, it suffices to measure its *similarity to existing key feature models, or clusters*.

Concretely, we may first represent a subspace cluster over l dimensions in a d dimensional ambient space as an l-dimensional centroid. This l-dimensional centroid is calculated as the arithmetic mean of the vectors contained in the cluster projected to the cluster's subspace. This centroid vector, therefore, represents the key features corresponding to the l dimensions in which it is defined, as well as the specific key feature values corresponding to its individual coordinates. An ideal value range for each key feature can, therefore, be established by considering some small interval around the corresponding coordinate of the centroid.

To determine the accuracy of a specific cluster, we utilize a validation set - a **subset of the input training set**: we will take 10% of our input active and inactive molecules as our validation set. Using this validation set and a single cluster, we first **rank the validation set of molecules according to activity** using the algorithm below. Note that the input candidate set C, in this case, is simply the validation set of molecules, $Cluster$ is a single subspace cluster found by DiSH, and $scoringMethod$ is a parametrizable input variable used within the

algorithm:

Algorithm 6: Candidate Molecule Ranking Algorithm

```
1  function RankCandidatesBasedOnActivity(Candidate Set C = [c_1, ..., c_m],
   Cluster, scoringMethod):
   Result: Ranked Candidate Set C' (Where C' is some permutation of C)
2  sortedActivityList = []
3  foreach Candidate c_k do
4      candidateDistanceArray = []
5      // The ranking will be based on the distance of each fragment of c_k
6      // from the cluster centroid
7      foreach Fragment fr_jk of candidate c_k do
8          // Compute the projected subspace Euclidean distance
9          fragmentCentroidDistance =
           computeSubspaceDist(fr_jk,Cluster.centroid))
10         candidateDistanceArray.append(fragmentCentroidDistance)
11     end
12     // A parametrizable scoring option dictates how the individual scores
       for the candidates are computed
13     if scoringMethod == mean then
14         candidateScore = mean(candidateDistanceArray)
15     end
16     else
17         candidateScore = min(candidateDistanceArray)
18     end
19     // We keep track of all the scores of candidates with respect to the
       input cluster
20     sortedActivityList.append((candidateMoleculeName:c_k,score:
       candidateScore))
21 end
22 // We sort the scores to obtain a ranking of candidates according to
   activity
23 sortBasedOnScore(sortedActivityList)
24 return sortedActivityList
```

Now that we have a ranking of the candidate validation molecules, we may then **score this ranking** using the **AUC score (Area Under the ROC Curve)** - a standard metric for scoring binary classification algorithms - for a ranking of molecules in a chemical database. For more information regarding the AUC score and its applications in virtual screening, we refer the reader to [32]. For our purposes, it suffices to understand that the AUC score AUC is a number in the interval $[0, 1]$, where a score closer to 1 indicates that a model, very accurately, places active candidates at the top and the inactive candidates at the bottom of the ranking; a score closer to 0, on the other hand, indicates the opposite and reveals the inaccuracy of a model.

Subsequently, we perform one more filtering of the subspace clusters, in addition to the filtering of clusters using the purity and diversity conditions. This filtering stage will be done using the validation set, the ranking algorithm shown above, and the AUC score mentioned in the prior paragraph to leave us only with

the *best key feature model* from a given set of input clusters.

The full scheme for determining the optimal model is given below - note that we use the method for determining the AUC score of a ranked list of molecules, as described in [32], as a black box in the form of the function $getAUCScore(candidateRanking)$:

```
1  function obtainBestClusterModel(Candidate Validation Set
   C = [c_1, ..., c_m], SetOfClusters):
   Result: Best Cluster Model Cl ∈ SetOfClusters
2  bestAUC = 0
3  bestClusterModel = NULL
4  foreach Cluster Cl_i ∈ SetOfClusters do
5      for scoringMethod in [mean,min] do
6          candidateRanking = RankCandidatesBasedOnActivity(C, Cl_i,
           scoringMethod)
7          currentAUC = getAUCScore(candidateRanking)
8          if currentAUC > bestAUC then
9              bestClusterModel = Cl_i
10         end
11     end
12 end
13 return bestClusterModel
```

4.5.2 Parameter Tuning via Validation

As is typical in various machine learning and artificial intelligence methods, it is of paramount importance to choose the appropriate parameter values for our pipeline - i.e. parameters for DiSH - that return the most accurate model for our particular problem of finding the key features in the molecular feature space.

Upon experimenting with various methods and heuristics to determine optimal values for the various parameters based solely on the properties of the input training dataset, we were unable to find any method that was robust across targets and datasets. We, therefore, chose a brute force search method, wherein we traverse the parameter space and choose the parameter combination that returns the best model; this method is also known as **Grid searching** in the domain of machine learning. [24]

The *best parameter combination* is determined by yet another *validation set*. This validation set is different from the one introduced in the previously. Namely, given a full set of input active and inactive molecules, we split the input into 3 sets - 80% of the input will be used as input to DiSH to find subspace clusters, 10% will be used as the first validation set to find the best cluster amongst a set of clusters returned by DiSH as presented in the algorithm shown above, and the last 10% will be used as the second validation set, as will be presented in this section, to determine the best cluster **across parameter combinations**.

To reify the discussion in the prior paragraph, we present the **full training and optimal model choosing algorithm** below, adding an outer loop for grid searching the parameter space. Note that $PipelineInput$ is the initial input to our pipeline that was elaborated on in the chapter regarding cleaning and combining

large molecular datasets:

Algorithm 7: Obtaining the Key Feature Model

```
1  function GetKeyFeatureModel(PipelineInput):
   Result: Get the best key feature model as determined by DiSH and our
           candidate ranking algorithms
2  // Split the input training actives and inactives into a training set and two
   validation sets, as mentioned in the above paragraph
3  trainingMolecules, ValidationMoleculesOne, ValidationMoleculesTwo
   = splitDataset(PipelineInput)
4  // Create the fully preprocessed molecular feature matrix FFM'_C using
   the training molecules and the data purification algorithms mentioned in
   prior sections
5  FFM'_C = cleanDataAndGetFFM(trainingMolecules)
6  // Keep track of all the best cluster models obtained from various
   parameter combinations, so we may choose the best one afterwards
7  bestClusterModels = []
8  foreach μRatio in [.2, .6, .9] do
9      foreach epsilon in [5e-9,.5e-7,5e-6] do
10         foreach scoringMethod in [mean,min] do
11             foreach purityThreshold in [.3,.5,.7] do
12                 // μ is taken as a ratio multiplied by the number of training
                   fragments
13                 μ = μRatio * length(FFM'_C)
14                 ClusterModel = DiSH(FFM'_C, μ, ϵ)
15                 // We keep the diversity threshold fixed, as we found
                   empirically that this worked best
16                 diversityThreshold = .5 * length(FFM'_C)
17                 // Filter the model based on purity and diversity to obtain
                   only the significant clusters
18                 SignificantClusterModel = filterModel(ClusterModel,
                   diversityThreshold, purityThreshold)
19                 // Perform the second filtering stage based on the first
                   validation set to obtain the optimal cluster from the set of
                   clusters
20                 bestCluster =
                   obtainBestClusterModel(ValidationMoleculesOne,
                   SignificantClusterModel)
21                 // Add this cluster to the list of best cluster models
                   obtained for various parameter combinations
22                 bestClusterModels.add(bestCluster)
23             end
24         end
25     end
26 end
27 // Now we use the second validation set to determine the best model
   across various parameter combinations
28 return obtainBestClusterModel(ValidationMoleculesTwo,
   bestClusterModels)
```

The choice of looping values for μRatio, ϵ, and $purityThreshold$ are meant to cover a range of possible values that could lead to optimal cluster detection. Setting μRatio and $purityThreshold$ is quite straightforward: since they are both constrained to the interval $[0, 1]$, it suffices to solely choose a range of values within that interval. The choices for ϵ were slightly more challenging, but we chose a robust range of values that would handle a variety of cases, based on the advice of the creators of DiSH. [20]

The algorithm keeps track of all the obtained AUC scores for various runs of DiSH, and then returns the model that achieved the highest AUC score at the end. Any subsequent candidate **testing molecules** will then be ranked using this model, as we deem it the best suited to make future predictions.

Although this approach for choosing the best parameter combination and consequently the best model might seem slightly unconventional, we argue that conceptually it is no different than learning the parameter combination that returns the best model using the intrinsic properties of the input dataset. Indeed, instead of learning the parameter combination implicitly using some clever scheme that utilizes the distribution of the feature values of the input training molecules, we rather learn these parameters *explicitly* by our brute force grid search approach and a validation set. The only valid argument against this approach is from the practical side: searching 51 different combinations of parameters versus just 1 is definitely more computationally demanding. But as we will see in the subsequent **Experimental Results** section, this computational inefficiency is not as bad as it first seems.

5. Experimental Results

5.1 Implementation

5.2 External Tools Used

Before embarking on a description of our experimental results, we would first like to briefly mention some external tools that were used in our research, without which the large-scale experiments conducted on enormous chemical datasets could not have been completed.

We would first like to thank MetaCentrum for providing distributed computing infrastructure on which we ran our parallelized experiments.[29] We would also like to thank the researchers developing and maintaining the ELKI data mining framework, at the time of writing researchers from Heidelberg University and the University of Southern Denmark, for providing us an implementation of the DiSH algorithm.[28] Both parties played a vital role in our experiments: the former party provided the raw computing power required for big data analysis, while the latter party provided the developed and tested software system which forms the core of our machine learning pipeline.

5.3 Evaluation of DiSH with Artificial Data

Due to the nature of our machine learning procedure as a multi-stage approach, it is imperative that we test each phase of the pipeline separately - otherwise known as *unit testing* in the domain of software engineering - and then subsequently test the modules together in ensemble as one integrated system - also known as *integration* or *functionality testing*. Although such an approach proves efficacious in debugging and improving the quality of software in nearly all development environments, the realm of machine learning provides an especially interesting case study of performing unit and integration tests. This is due to the fact that the accuracy and durability of any stage s_i in a machine learning pipeline depends heavily on the accuracy and robustness of all the previous stages $s_0 \dots s_{i-1}$. Indeed, if we find that our models return low AUC scores on new testing candidate molecules, it is often difficult to say exactly in which phase of the pipeline the algorithm goes awry since the output of every stage is dependent on the outputs of previous stages. Unit testing, therefore, provides an efficient way of isolating the source of errors in many situations. More than any other stage in the pipeline, the stage that requires the most testing is the stage involving the **utilization of the DiSH algorithm in the molecular feature space**. We will present a short discussion of a testing scheme to ensure that DiSH is a reasonable choice to use in the molecular feature space, before moving on to our experimental results description.

5.3.1 Methodology

Subspace clustering provides the cornerstone for any insight we wish to obtain about the key features with respect to a target molecule. It is, therefore, necessary to verify that when applied to the molecular feature space, DiSH behaves with reasonable accuracy in detecting any underlying fragment vector clusters. Before testing DiSH with *real* molecular data, therefore, we first unit test its capabilities by generating *artificial* data, feeding this data as input to DiSH, and observing the results. The advantage of using artificial data is that we, the creators of the data, have prior knowledge regarding the underlying distribution of vectors in the artificial dataset. Moreover, we have some expectations about the real molecular data that allows us to model our artificial data after the real data. Consequently, it becomes substantially easier to evaluate the effectiveness of DiSH because we can verify whether it detected the specific trends - in our case clusters - that we ourselves implanted within the artificial dataset, and then extrapolate that these trends will be detected also in the real data.

To reify this abstract discussion, we will now present details regarding the aforementioned artificial data. In our application, the artificial data will closely resemble real data; indeed, our artificial database will simply be, once more, a database of d-dimensional vectors labeled by affiliation to an active or inactive molecule; these vectors correspond to the fragment feature vectors. The key difference between our artificially generated database and our real molecular database is in the distribution of the vectors: during our generation of these data points, we ensure that the vectors form n clusters, where n is any natural number. On the other hand, in real data, we are not given any guarantees that clusters actually exist - drawing a clear distinction between our artificial data and real data. We generate these n clusters according to several parameters: *ValueRange, InterClusterDistance, Density, NumClusterPoints, NumClusteredDimensions, and TotalNumberDimensions.*

We will briefly describe the parameters one by one:

ValueRange Determines the range of values that the cluster centroids can take; if the $ValueRange$ is a number VR, then we generate cluster centroids with coordinates in the interval $[0, VR]$.

InterClusterDistance Determines the degree of *separability* of the generated clusters: if the $InterClusterDistance$ is given by a number ICD, then the cluster generation engine ensures that any two centroid coordinates differ by *at least* ICD in *any one dimension.*

Density Determines the overall *spread* of the points in the cluster; one can also imagine this parameter as a modified version of the *radius* of a cluster. Omitting certain details, a *higher* input value for density will lead to clusters that are more dispersed, whereas a *lower* input value for density will lead to clusters that are more compact. The reader should already note here that the combination of $InterClusterDistance$ and $Density$ will play a large part in determining the overall shape of our clustering distribution.

NumClusterPoints Quite self-explanatory - simply denotes the number of unique points in each generated cluster. The sole interesting connotation of this parameter is that *each artificial cluster will have the same number of points.* Although this may not be realistic, this keeps the cluster generation slightly simpler.

NumClusteredDimensions and TotalNumberDimensions Since we are generating subspace clusters, we are required to specify how many dimensions are clustered, and how many are left unclustered. As a proxy for those two parameters, we take as input the number of clustered dimensions and the *total number of dimensions*, subsequently calculating the number of unclustered dimensions as follows: $NumUnclusteredDimensions = TotalNumberDimensions$ - $NumClusteredDimensions$. In this manner, we obtain exact values pertaining to the dimensionality of the generated subspace clusters. The reader should note that for the purpose of simplicity, **we keep these parameters constant across artificially generated datasets**.

Although the above discussion of the parameters inputted to the cluster generation module may seem slightly confusing to the reader, we stress that it is simply a formal way to depict and describe what we intuitively imagine about clusters (and via the dimensionality parameters, subspace clusters). Indeed, we imagine clusters as collections of points in a *close proximity* of some centroids in the Euclidean space, and moreover, we imagine any two clusters to be differentiated by the *distance between their centroids*. The first idea is captured by the *Density* parameter, and the second by the $InterClusterDistance$ parameter. For several combinations of values of the parameters above, we generate a clustering dataset with five clusters; the principle, therefore, is that we generate different datasets with different characteristics via varying combinations of parameters, and subsequently observe how DiSH performs in response to the dynamic input.

The concrete consequence of these parameter combinations in the subdomain of subspace clustering is that we may generate several clustering datasets, and then run DiSH on each of these datasets one by one. After running DiSH multiple times, we may then calculate multiple *scores* based on DiSH's performance in detecting the various clusters present in the datasets. **This score is not to be confused with the AUC Score mentioned in the previous section:** this score is one that we will devise for the sole purpose of measuring DISH's performance on our generated testing data.

The scoring of DiSH for some general artificially generated dataset D containing clusters $C_1 \dots C_n$ will be done as follows. Assume that for this particular dataset D, DiSH manages to detect clusters $T_1 \dots T_m$, where m need not equal n. Our goal is to generate a number which will *unambiguously capture the level of accuracy* in which DiSH managed to find the embedded artificial clusters $C_1 \dots C_n$.

We create this unambiguous score with a technique known as *matching*; as its name suggests, the technique revolves around trying to match each detected cluster T_i with some generated cluster C_j. We refine the idea slightly with some more formalisms below.

Definition. *Cubic Intersection* Two arbitrary subspace clusters C_1 and C_2 form a cubic intersection if and only if:

Condition 1 They are defined in the same dimensions.

Condition 2 Let Condition 1 hold; then for any arbitrary dimension i for which C_1 and C_2 are both defined, let Max_{C1_i} denote the maximum value of coordinate p_i for some point $p \in C_1$, and let Min_{C1_i} denote the minimum value of coordinate q_i for some point $q \in C_1$. If we let Max_{C2_i} and Min_{C2_i} be the analogues for C_2, then we must have that either $Min_{C1_i} \leq Min_{C2_i} \leq Max_{C1_i}$ *or* $Min_{C1_i} \leq Max_{C2_i} \leq Max_{C1_i}$.

Returning back to our evaluation of DiSH, we start by first taking each cluster T_i detected by DiSH, and filter the set of generated clusters $C_1 \dots C_n$ to include only clusters *which form a cubic intersection* with T_i. Then from this filtered list of clusters, we pick the cluster C_j whose centroid is closest to the centroid of T_i, based on the Euclidean distance defined in the subspace of T_i with C_j, and then *match* T_i with C_j. We then remove both T_i and C_j from consideration, and continue looking for further matches. Note that a cluster T_i need not have a match; for example, if no cluster C_j satisfies the conditions to form a cubic intersection with T_i, then T_i will necessarily have no match. Therefore, quite intuitively, the accuracy of the detected clusters $T_1 \dots T_m$ outputted by DiSH can be measured by the amount of matches we manage to create with the generated clusters.

We follow this intuition and enhance it with a formal score: we define the score s_D for dataset D as:

$$s_D = \frac{NumberOfMatches}{|C_1 \dots C_n|} \tag{5.1}$$

It is trivial to see that $0 \leq s_D \leq 1$; and moreover, it is intuitively clear that the closer s_D is to 1, the better DiSH has performed on the dataset D. For example, if s_D is 0, it implies that the detected clusters returned by DiSH could not be matched with any of the generated clusters, and subsequently, we can infer that DiSH did not perform very well for the dataset D. As in any situation involving complex data analysis, it is imperative that we circumvent any problems caused by a particularly abnormal dataset. We would, therefore, like to apply this scoring procedure for various artificially generated clustering datasets $D_1 \dots D_k$, created as mentioned before by various parameter combinations, and then analyze the scores of the datasets in toto. Such a holistic procedure over many datasets prevents us from being misled by any particularly high or low score for some dataset D_j. Therefore, an integrated analysis will give us a realistic indication of DiSH's *overall* performance over many types of clustering data.

In order to obtain various datasets and their corresponding matching scores, we generate clustering datasets by varying just *4 Parameters*, while keeping the rest of the parameters constant. We will then subsequently run DiSH on these varied datasets and obtain a multitude of scores for various types of clustering datum.

Varying of the 2 Input Parameters to Artificial Cluster Generation: $InterClusterDistance$ and $Density$. We vary the Inter-Cluster Distance and Density parameters because of the relative magnitude of the ramifications that varying these parameters have on the distribution of the clusters. For example, changing the $Density$ parameter from small to large will have huge effects on how compact our generated clusters are, and by extension change DiSH's behavior when trying to detect these clusters. A similar inference can be made about changes to the $InterClusterDistance$ parameter. Changes to other parameters, meanwhile, don't intuitively portend any huge changes to DiSH's behavior: for example, changing the $ValueRange$ parameter won't imply a large change to DiSH' performance since our datum is normalized according to the range anyway.

Varying of the 2 Input Parameters to DiSH itself: ϵ and μ (also called $NumPoints$, following the convention of clustering terminology). We would also

like to observe how the score changes when DiSH is faced with the same input dataset, but different input parameter combinations of ϵ and μ. This will enable us to understand how sensitive DiSH is to changes in the input parameters with a lot more clarity and therefore prepare us for choosing appropriate parameters when we move on to analyses with real molecular datasets.

In conclusion, we generate our various diversified clustering datasets by changing the *InterClusterDistance* and *Density* parameters, while keeping the rest of the parameters constant. We then run DiSH on these datasets using various combinations of values for ϵ and μ, thereby obtaining a set of scores for a wide variety of possible scenarios.

5.3.2 Results

Now that we have understood the peculiarities of our testing platform, we proceed on to a more concrete discussion regarding the results of evaluating DiSH using our artificial data as input. A full, in-depth discussion of the experimental results is available at the end of this paper in the **Attachments** section; in this section we provide a succinct summary of these results.

We commence by presenting the following histogram showing an overview of the performance of DiSH:

Figure 5.1: A histogram showcasing the distribution of scores accross clusterings generated via different combinations of parameters. In total there were 256 different combinations of parameters utilized in the parameter space grid search. Each bucket in the histogram corresponds to a particular range of scores - e.g. the **bin 0 represents scores between 0 and 0.2**, the **bin 0.2 represents scores between 0.2 and 0.4**, and so on until we reach **bin 1, which represents only perfect scores of 1**.

The statistical metadata corresponding to the distribution of the scores are

as follows: **Mean Score:** .5, **Variance of the Scores:** .19, **Maximum Score:** 1.00, and **Minimum Score:** 0.00. One can quickly comment that the contour of the bar chart forms some type of inverse normal distribution; indeed, this immediately gives us an indication that DiSH performs very well for some datasets, but simultaneously very poorly for others. We can moreover glean, through our statistical metadata, that DiSH generally counterbalances all the perfect scores of 1 with the worst scores of 0; moreover, there are relatively few clusterings achieving scores somewhere in between the two extremes. This information, along with *heat maps* in the *Attachments* section, are highly valuable - they indicate that DiSH is highly susceptible to the implicit input parameters of the dataset: so much so, that a change in even one of our 4 chosen parameters $InterClusterDistance$, $Density$, ϵ, and μ seems to be able to take DiSH's resulting score from 0 to 1, or vice versa. We have neglected to analyze one possibly important piece of statistical metadata - the variance. A variance of .19 implies a standard deviation of about 0.44: these values are also quite telling and consistent with our bar chart. We can see that the standard deviation is nearly half the range, which means that many of the scores exhibit not even a modicum of conformity with the mean. Indeed, this once again indicates a distribution resembling the inverse normal distribution and sends us strong signals that the value of input parameters is playing a large part in DiSH's performance. Empirically, DiSH seems to be a bipolar algorithm, switching between amazing and dismal performance, when applied to artificial datasets resembling those found in the molecular feature space. The outcome of our quantitative analysis may seem rather unsurprising to the reader - of course, the input parameters will change the behavior of DiSH and the resulting score. What is more surprising is the fact that the change seems to be so stark - we don't obtain just a small variance in the score but rather scores that are magnitudes better or worse due to a change in input parameters.

Having observed the varying performance of DiSH based on the choice of parameters, we strove to determine particular combinations of parameter values that seem to produce very high scores or very low scores, so that we may better prepare ourselves for understanding the variance in DiSH's performance if it appears with real molecular data. This analysis was done, again, using *heat maps* of different testing runs of DiSH using our artificial data generated by different parameter combinations, and is presented in the **Attachments** section at the end of this paper; here, we simply present the results of our analysis on a parameter by parameter basis below:

InterClusterDistance: The highest scores occur when the ICD is close to half of the $ValueRange$ of the dataset - therefore not too big nor too small with respect to the $ValueRange$ interval.

Density: The highest scores occur when the $Density$ parameter is small compared to the overall value range - ideally less than 5% of $ValueRange$.

Epsilon: ϵ exhibits no general trends with respect to the score: both high and low values of *epsilon* exhibit high scores - therefore it's effect is simply too dependent on other parameters to make a general conclusion.

μ (or $NumPoints$): Lower values of μ, with respect to the number of points per cluster in our generated dataset, result in higher scores - ideally μ is no greater than 30% of $NumClusterPoints$.

Final Conclusions About DiSH. We have done an in-depth analysis of

DiSH's performance using many artificially generated clustering distributions inspired by the various implicit parameters in high dimensional subspace clustering problems. Using a combination of histograms and heat maps, we have found that even small variations in some of the implicit parameters can result in large changes of the matching score, meaning that in a lot of cases we either achieve very accurate scores or very inaccurate scores. We have also proceeded one step further, and understood in which value ranges of certain highly important implicit parameters DiSH performs optimally. Concluding that an intermediate value for $InterClusterDistance$, a low value for $Density$, - both with respect to the $ValueRange$ - and a low value for μ with respect to $NumClusterPoints$ attain optimal scores, we now understand DiSH well enough to test our pipeline on real molecular data.

5.4 Evaluation of DiSH With Real Data

5.4.1 The Evaluation Method

We evaluate DiSH on several **classes of targets**. Each class contains targets, whose respective actives and inactives will be used as input to our pipeline, that achieved an AUC score in a particular range when identical data were fed as inputs to other benchmark virtual screening algorithms. These algorithms include Atom Pairs (AP), ECFP2, FCFP2, MACCS, and Topological Torsion (TT) fingerprints. In this manner, we can evaluate DiSH's performance against several other benchmark algorithms in the realm of virtual screening.

Concretely, in our evaluation dataset, we have *4 different classes* - **.80-.85**, **.85-.90**, **.90-.95**, and **.98-.10**. Each class contains targets that exhibited AUC Scores in the range corresponding to the class name in prior experiments with the aforementioned other virtual screening algorithms.

We showcase the results of this testing using the visualization method of a *heat map*. The rows of the heat map represent the *different targets of each class*, and the columns represent the different *training/test splits* of the input actives and inactives. In our evaluation datasets, we use 10 training/test splits of the active and inactive molecules $Split_1 \dots Split_{10}$. If we subsequently have 5 target macromolecules $t_1 \dots t_5$, then the cell entry $Heatmap_{1,5}$ in the heat map will correspond to the AUC score of the model obtained from the function $ChooseBestModel(FFM'_{C_1}, Split_5.testMols)$ - the $ChooseBestModel$ function was presented in the prior section - where FFM_{C_1} corresponds to the fully preprocessed Fragment Feature Matrix of the *training molecules*, $Split_5.trainingMols$, with respect to the target t_1.

Moreover, the scores across various splits can then be averaged, leading to a final *aggregate* score for the target; this minimizes the chance that any outlier AUC scores obtained from particularly good or bad splits skew our view of the pipeline, and gives us a more holistic review of our performance.

5.4.2 Results

We present the heat map of results below, in Figure 5.2

Class .80-.85

	Split 1	Split 2	Split 3	Split 4	Split 5	Split 6	Split 7	Split 8	Split 9	Split 10	Average
5HT2B	.49	.41	.62	.45	.55	.60	.47	.48	.58	.59	.52
5HT2C	.48	.55	.44	.59	.57	.58	.56	.48	.59	.54	.54
ADA2A	.55	.63	.65	.55	.47	.49	.53	.63	.55	.58	.56
CDK2	.50	.49	.57	.61	.56	.50	.54	.53	.56	.50	.54
HDAC01	.62	.65	.67	.46	.53	.63	.62	.54	.57	.61	.59
PXR_Agonist	.45	.56	.66	.61	.54	.60	.57	.58	.59	.57	.57

Class .85-.90

ACM1_Agonist	.54	.59	.66	.64	.63	.57	.54	.58	.60	.54	.59
ADA2B_Antagonist	.47	.60	.54	.56	.54	.52	.49	.48	.58	.49	.53
ADA2C_Antagonist	.53	.58	.66	.65	.62	.61	.57	.56	.55	.60	.59
CHK1	.53	.52	.64	.57	.63	.62	.66	.61	.60	.58	.60

Class .90-.95

5HT1F_Agonist	.54	.55	.58	.64	.63	.57	.64	.67	.65	.54	.60
DRD1_Antagonist	.52	.61	.53	.63	.56	.61	.54	.64	.52	.51	.57
DRD2_Agonist	.50	.65	.66	.70	.53	.63	.62	.51	.52	.69	.60
LSHR_Antagonist	.65	.66	.53	.54	.49	.68	.69	.56	.63	.59	.60
OPRM_Agonist	.60	.57	.57	.59	.63	.62	.64	.67	.52	.51	.59

Class .98-1.0

DHFR	.68	.70	.67	.51	.55	.68	.53	.67	.59	.65	.62
MTR1A_Agonist	.52	.64	.58	.56	.54	.50	.61	.65	.67	.54	.58
MTR1B_Agonist	.55	.63	.56	.57	.58	.71	.52	.53	.65	.66	.60
P38	.61	.54	.67	.53	.58	.63	.52	.59	.63	.68	.60
V2R_Antagonist	.63	.52	.65	.71	.62	.64	.51	.55	.54	.68	.61

Figure 5.2: A heat map showcasing the experimental results of DiSH with real macromolecular targets.

We can immediately observe that, in general, higher scores are obtained as we go from the *.80-.85* class to the *.98-1.0* class - which is to be expected, as the classes *.90-.95* and *.98-1.0* have exhibited higher scores with other benchmark algorithms in previous experiments as well. Overall, DiSH seems to perform slightly worse than other benchmark algorithms when tested with the above targets: this can be seen when comparing the average scores, depicted in the final column, with the name of the class that the target belongs to.

Although the results of applying DiSH and our pipeline in the domain of virtual screening were seemingly unimpressive when compared to other benchmark algorithms, there are still some interesting analyses left to be done as to determine the cause of this discrepancy and suggest some positive future steps that can be enacted to rectify the difference in score.

5.4.3 Evaluation using Bayesian Centroids

In order to uncover the source of the systematic difference in score between our pipeline and other benchmark algorithms with respect to our testing targets,

we investigate each step of the pipeline separately and try and find errors in individual steps that affect the overall results; once again, the technique of *unit testing* will be used to reveal possible pitfalls in our approach.

Concretely, there are three separate steps in our pipeline that could lead to the discrepancy in the score: the *preprocessing* step, the *cluster finding* step, and the *candidate scoring and ranking step*; each step, therefore, needs to be tested to ensure that it is doing its job properly.

1. The Preprocessing Step The testing of the preprocessing step is made quite simple by utilizing the results of another research endeavor - namely the utilization of the *Bayesian Method* in virtual screening, mentioned in the earlier chapter *Related Work And Our Approach*. The researchers who conducted experiments using the Bayesian method used the *same preprocessed data* as we did in our research, to quite successful results, as shown in their publication. [27] This is conclusive proof that our preprocessing methods cannot be the cause of the lower scores, due to the fact that another method used the same preprocessed data as us to achieve quite successful results. This implies that the insight needed for virtual screening is very much there in the preprocessed data, but the subsequent *cluster finding* and *candidate scoring* steps are not able to find it as easily as we'd hoped.

2. The Cluster Finding Step As the reader will recall from earlier on in this chapter, we already tested the cluster finding step by *evaluating DiSH with artificial data*. Recalling the results of those tests, the reader may remember that DiSH was highly susceptible to the input parameters both implicitly present in the dataset, such as the $InterClusterDistance$ and $Density$, and explicitly chosen by us, such as ϵ and μ. Indeed, it was so susceptible that a change in one parameter seemed to cause DiSH to go from finding all of the clusters we implanted to finding none. Furthermore, we found through empirical observation that certain value ranges of the implicit and explicit input parameters were conducive to DiSH finding more or fewer clusters. Therefore, the lower AUC scores with the real molecular data could be due to the values of certain implicit parameters - such as too high of an $InterClusterDistance$ between clusters, or too high a $Density$ within clusters; these values would subsequently lead to DiSH not finding the molecular fragment clusters properly and consequently lead to a lower AUC Score. The question arises as to using other subspace clustering algorithms to circumvent DiSH's pitfalls. We hypothesized that DiSH, due to its theoretical properties discussed earlier, was the most appropriate algorithm in the molecular feature space, and were limited due to time constraints in testing other algorithms. A brief discussion of the usage of other algorithms is available in the next chapter *Conclusion and Future Work*.

3. The Candidate Scoring and Ranking Step Assuming that we obtain proper molecular fragment clusters that correspond to real *modes of activity*, it is necessary to test whether we are able to properly utilize these clusters to score and rank new candidate molecules. After all, if this is done in an improper manner, our AUC score would be low no matter how accurate the previous stages of the pipeline are. In order to test this step of the pipeline, we utilize once again the results of the Bayesian method. For each feature, the Bayesian method splits up the value range into several intervals or *buckets*; for each of these intervals, the Bayesian method calculates a conditional probability: the probability that a cer-

tain fragment vector is active conditioned on the fact that the vector has a value for that respective feature in that respective bucket.[27] One may then choose the interval for each feature that has the highest probability, and consider this the *feature ideal interval* for that particular feature. Choosing 3 of the intervals in separate features with the highest such probability, one can create a *key feature interval model* with the Bayesian method; moreover, by taking the midpoints of these intervals, we can create a *subspace cluster centroid* corresponding to the top 3 key features, as determined by the Bayesian method.

Since we have a cluster centroid, we may subsequently test the Bayesian method in the same manner as we did for DiSH: we pass the *Bayesian centroid* as input to our candidate ranking and scoring algorithm, $RankCandidatesBasedOnActivity(Split.testMols, BayesianCentroid, min)$. We may then use this ranking to obtain our AUC score, just as we did for DiSH.[1] The results of this analysis, for just one training/test split, are shown below, in Figure 5.3.

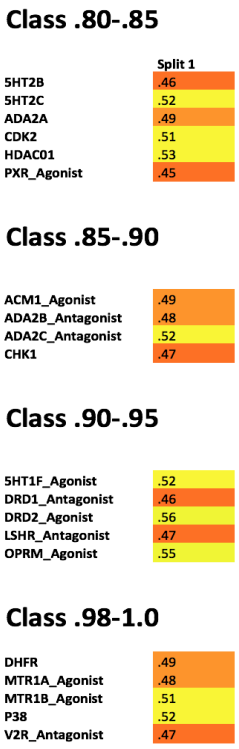

Figure 5.3: A heat map showcasing the experimental results of evaluating the Bayesian centroid with one split.

Surprisingly, the Bayesian centroid model generally performs worse than the cluster model of DiSH when evaluated with our scoring method, even though the Bayesian algorithm itself achieved very high AUC scores in other evaluations by other researchers using a different scoring method.[2] This is a very significant

[1] We set the scoring method to *min* here just for illustrative purposes; when testing the Bayesian centroid, we found practically no difference between using *min* or *mean* as the scoring method.

[2] We state this with the caveat that the other scoring method uses all the dimensions, and not just 3 as we have done here. We, however, performed informal scoring of the Bayesian

indication that the candidate scoring and ranking step may be one of the key reasons behind our pipeline's relatively lower AUC scores.

To further test this hypothesis, we even evaluated the similarity between DiSH's optimal cluster centroid and the Bayesian centroid; this similarity can be evaluated by simply calculating the Euclidean distance between the two centroids, after projecting the centroids *onto their common dimensions* - i.e. the dimensions that they are *both* defined in. We first present the number of common dimensions between the DiSH centroid and Bayesian centroid when trained over various targets and splits in Figure 5.4.

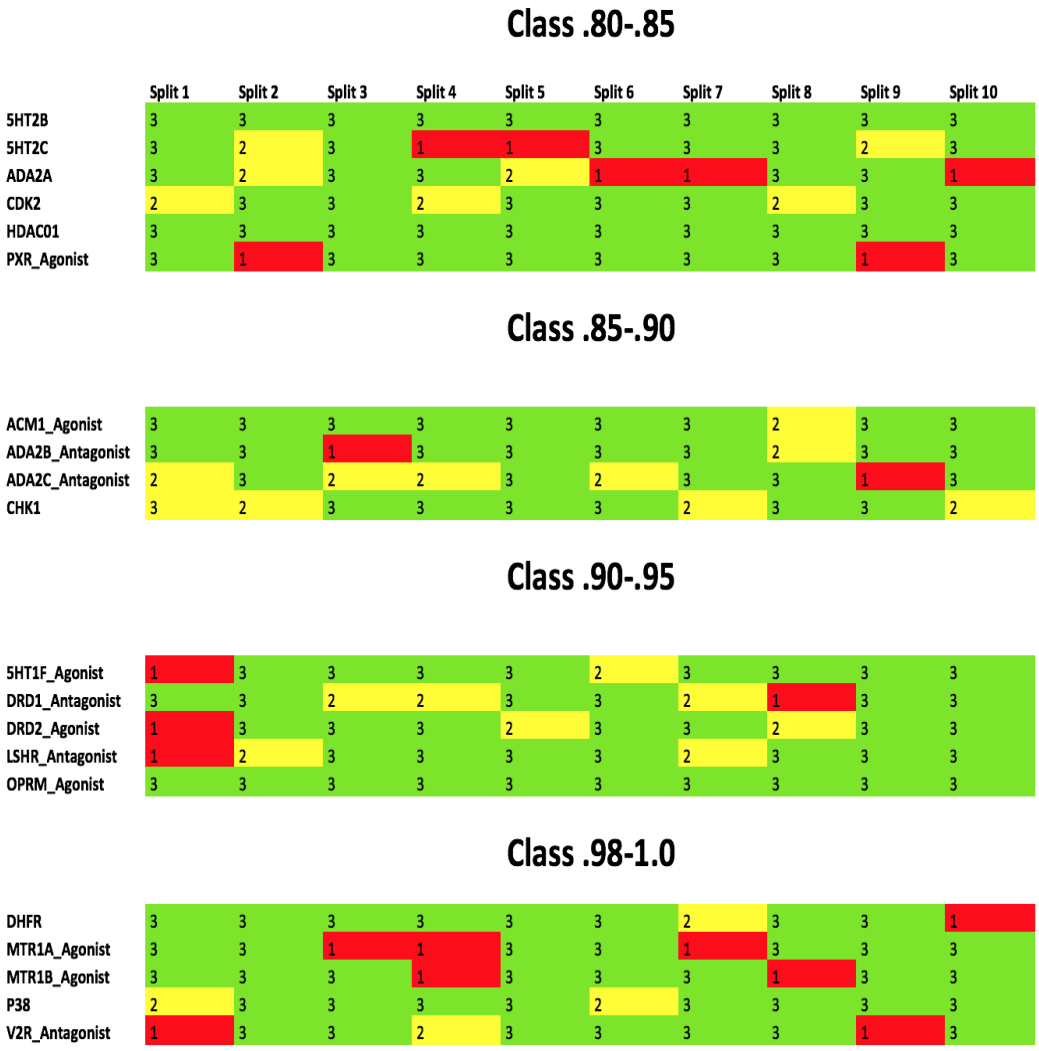

Figure 5.4: A heat map showcasing the common dimensionality between the DiSH centroid and the Bayesian centroid.

Recalling that the Bayesian centroid is only defined over 3 dimensions, we notice that, for the majority of targets and splits, the number of common dimensions is indeed 3 - or the maximum possible. Therefore from a pure dimensionality standpoint, the DiSH centroid and the Bayesian centroid seem to exhibit quite a

centroids with even more dimensions, and found that scores still do not improve beyond the showcased results in the heat map in Figure 5.3.

bit of similarity. Now we present the results of the Euclidean distance calculations between the two centroids in Figure 5.5.

Class .80-.85

	Split 1	Split 2	Split 3	Split 4	Split 5	Split 6	Split 7	Split 8	Split 9	Split 10
5HT2B	.21	.32	.12	.07	.17	.30	.23	.08	.15	.25
5HT2C	.14	.17	.08	.06	.08	.13	.23	.09	.04	.19
ADA2A	.35	.23	.38	.28	.25	.27	.36	.19	.26	.35
CDK2	.31	.34	.39	.32	.28	.33	.26	.23	.30	.21
HDAC01	.40	.41	.47	.41	.47	.34	.45	.39	.46	.47
PXR_Agonist	.38	.35	.33	.38	.29	.41	.42	.21	.19	.15

Class .85-.90

ACM1_Agonist	.21	.20	.24	.22	.31	.24	.20	.18	.32	.26
ADA2B_Antagonist	.33	.34	.66	.32	.31	.24	.45	.23	.21	.38
ADA2C_Antagonist	.27	.20	.25	.21	.19	.20	.25	.31	.20	.19
CHK1	.18	.45	.21	.32	.21	.23	.34	.43	.29	.38

Class .90-.95

5HT1F_Agonist	.00	.22	.14	.16	.10	.00	.08	.18	.14	.14
DRD1_Antagonist	.35	.35	.44	.39	.32	.45	.31	.35	.38	.37
DRD2_Agonist	.40	.16	.39	.38	.21	.14	.15	.32	.15	.14
LSHR_Antagonist	.34	.32	.24	.28	.37	.40	.35	.31	.34	.35
OPRM_Agonist	.13	.15	.14	.11	.12	.15	.10	.14	.13	.20

Class .98-1.0

DHFR	.36	.30	.28	.13	.35	.54	.10	.32	.48	.21
MTR1A_Agonist	.22	.16	.38	.21	.24	.23	.36	.32	.21	.19
MTR1B_Agonist	.17	.07	.10	.23	.25	.26	.22	.21	.20	.17
P38	.00	.00	.31	.23	.10	.21	.15	.13	.11	.17
V2R_Antagonist	.72	.32	.14	.45	.34	.28	.29	.35	.29	.13

Figure 5.5: A heat map showcasing the Euclidean distance between the DiSH centroid and the Bayesian centroid over their common dimensions. We didn't color all the entries in the heat map due to the fact that it is hard to compare Euclidean distance metrics defined over different numbers of dimensions; we did, however, color a special subset of scores green if they satisfy the condition that the score is below .20 *and* the number of common dimensions is the maximum 3.

The above heat map isn't colored as it was previously since it is too difficult to compare Euclidean distance metrics defined over different numbers of dimensions. A special subset of the entries, however, those that contain a score that is below .20 *and* the number of common dimensions corresponding to that respective entry is the maximum 3, were colored green. The green color is an indication that these entries indicate a very visible similarity between the Bayesian centroid and the DiSH centroid. Indeed, keeping in mind that the ranges for all the features are normalized to the interval $[0, 1]$, a Euclidean distance defined over 3 dimensions that are below .20 is remarkably low. Even more striking is that quite a few entries exhibit this property and are colored green, indicating to us that the Bayesian

centroid and the DiSH centroid are quite closely related in several cases.

5.4.4 Positive Results and the Future of Subspace Clustering in Virtual Screening

All in all, this line of logic leads to the same question: if the Bayesian centroids also perform poorly when combined with our scoring method, *and* these centroids are in several cases almost the same as the DiSH centroids, is there something fundamentally wrong with the way we rank and score molecules? There is no way to know for sure, unless we can engineer a ranking method that performs better; we can, however, definitely hypothesize that this step is the most likely reason as to why our pipeline performs worse than other benchmark algorithms. This leaves a tantalizing question for future researchers: what method can we construct to use the DiSH centroids to rank the molecular database so that the AUC score is improved? Whatever the answer is, the DiSH centroids' similarities to the Bayesian centroids indicate that there is indeed a future ahead for DiSH and subspace clustering in the space of virtual screening - as long as we understand how to use them properly.

5.5 Time Complexity

As with any computational procedure, it is necessary to analyze the time complexity, both theoretically and practically, of our algorithmic pipeline. According to the analysis done by the authors of the algorithm, DiSH scales slightly superlinearly with respect to the number of vectors in the dataset as well as with respect to the dimensionality of the aforementioned vectors.[20] Additionally, our data preprocessing procedures have a time complexity of $O(m^2n)$, where m is the *total number of constituent fragments* of the active and inactive molecules, and n is the dimensionality of the dataset - or the number of physiochemical features. Finally, the procedure for ranking the validation set of molecules and choosing the best cluster model has time complexity $O(n(m + nlogn))$ - where m is the total number of fragments of the validation set of molecules, and n is the total number of molecules in the input training set.

From a practical standpoint, one run of the algorithmic pipeline - including the grid testing of the parameter space for the best model - with a set of real active and inactive molecules as input data takes on average 16 hours on an x86 64-bit computer with 8 GB of RAM.

6. Conclusion and Future Work

Having concluded with the experimental results section, we reflect on the results of our novel machine learning approach in the field of virtual screening, and envision some future work that can be done to further advance the state of the art.

As we concluded towards the end of our *Experimental Results* section previously, although the AUC Scores of the pipeline are not too impressive, it need not mean that subspace clustering is not applicable in the molecular feature space. Indeed, the close proximity of the cluster centroids discovered by DiSH and the *Bayesian centroids* returned by the Naive Bayes method in the key subspaces indicates that the underlying model is captured, but is not able to be found - possibly due to the **improper candidate scoring method**. A possible point of future research could subsequently be in re-engineering the method in which we score a candidate molecule, based on further research on the DiSH cluster centroids and the fragment feature vectors of the candidate.

Looking past possible short-term improvements to subspace clustering in virtual screening, several long-term, more ambitious, research endeavors could also be undertaken to improve the accuracy of this machine learning technique in cheminformatics. A logical next step is to test the other modern subspace clustering algorithms mentioned in the *Subspace Clustering Approach* chapter; indeed, the centroid of the DiSH key feature model, although sharing many similarities with the Bayesian key feature model in 3 dimensions, also contains several other *noisy* features, or dimensions, that could be the cause of the lower AUC Scores. We did not test other subspace clustering algorithms in our research solely due to time constraints, but the pipeline detailed in this paper can be expanded to be utilized with any subspace clustering algorithm and is not exclusive to DiSH. The ELKI data mining framework itself - mentioned in the beginning of the *Experimental Results* chapter - has implementations of several subspace clustering algorithms that can be used as the basis for further research in the area.

An even more creative approach is also possible: the reader may recall from the prior *Subspace Clustering Approach* chapter the *clustering distance function* abstraction used in the construction of the *reachability plot* in certain clustering algorithms, such as DiSH. As we mentioned earlier, this distance function is an incredibly powerful abstraction that is the foundation for the construction of the reachability plot; as a result, it can have a profound effect on the type of clusters we identify.

The thought arises, then, as to the possibility of *customizing* this distance function for virtual screening purposes. Concretely, taking the example of DiSH's *subspace distance* between two fragment vectors p and q, one need not only consider the *number of 0s* in the logical and of the subspace preference vectors $w_q \wedge w_p$ as the subspace distance, but rather may also consider the *intrinsic properties of the fragment vectors* - such as *whether they belong to active or inactive molecules*. To reify this discussion, assume that there also exists a third point r. We have that both q and r are candidate points to be added to the existing reachability plot containing p, $SubspaceDist(p, q) \leq SubspaceDist(p, r)$ (where $SubspaceDist(P1, P2)$ is the subspace distance used in the DiSH algorithm), *and*

both p and r are fragments that are contained in several active molecules whilst q is contained solely in inactive molecules. The traditional DiSH algorithm, everything else held constant, would prefer to add q rather than r to the existing reachability plot due to its closer proximity to p; a DiSH algorithm optimized for virtual screening would, however, add r rather than q due to its active nature. A more *activity-centric* reachability plot would subsequently lead to the detection of more relevant activity-centric clusters, a better key feature model, a more accurate candidate ranking, and finally higher AUC scores. Such re-engineering is just one example of several simple changes that could be made to existing subspace clustering algorithms to make them more relevant in the molecular feature space.

The usage of machine learning and subspace clustering in a remarkable array of problems in various fields will only increase in the future as more research time is dedicated these topics and the computational power of modern computers continues to follow Moore's law. Virtual screening and the identification of key features and their ideal intervals will continue to be prominent research topics in the domain of cheminformatics. In this paper, we've built the foundation for subspace clustering to be amongst the key methods used to solve these complex and multifarious problems, and we envision a bright future ahead for it in cheminformatics.

Bibliography

[1] Edited by Alexandre Varnek and Alex Tropsha. *Chemoinformatics Approaches to Virtual Screening.* Royal Society of Chemistry, 2008. ISBN 978-1-84755-887-9.

[2] Gordon Moore. *Gordon Moore's Law.* Intel Online Website. 2016.

[3] Jun Xu and Arnold Hagler. *Chemoinformatics and Drug Discovery.* Discovery Partners International, Inc., 2002. ISSN 1420-3049.

[4] Tiejun Cheng, Qingliang Li, Zhigang Zhou, Yanli Wang, and Stephen H. Bryant. *Structure-Based Virtual Screening for Drug Discovery: a Problem-Centric Review.* AAPS J. 2012 Mar; 14(1): 133–141. DOI: 10.1208/s12248-012-9322-0.

[5] Alfonso T. García-Sosa, Mare Oja, Csaba Hetény, and Uko Maran *DrugLogit: Logistic Discrimination between Drugs and Nondrugs Including Disease-Specificity by Assigning Probabilities Based on Molecular Properties.* J. Chem. Inf. Model., 2012, 52 (8), pp 2165–2180, American Chemical Society. DOI: 10.1021/ci200587h.

[6] David Hoksza, Daniel Svozil, and Martin Svicho. *Activity Driven Exploration of Chemical Space with Morphing.* IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2015. ISBN 978-1-4673-6799-8.

[7] Wermuth CG et al. *Glossary of terms used in medicinal chemistry (IUPAC Recommendations 1998).* Pure and Applied Chemistry, 1998. 70. 5. 1129–1143. 10.1351/pac199870051129.

[8] Kier LB. *Molecular Orbital Calculation of Preferred Conformations of Acetylcholine, Muscarine, and Muscarone.*Mol Pharmacol 1967;3(5):487-494.

[9] Everything Explained. *Pharmacophore.* http://everything.explained.today/Pharmacophore. 2016.

[10] Yap CW. *PaDEL-Descriptor: An open source software to calculate molecular descriptors and fingerprints.* J Comput Chem. 2011 May;32(7):1466-74. doi: 10.1002/jcc.21707. Epub 2010 Dec 17.

[11] Aakash Ravi *Molecular Feature Engineering- Online Open Source Code on GitHub.* https://github.com/aakashrav/MolecularFeatureEngineering. 2016.

[12] Marina Soley-Bori. *Dealing with missing data: Key assumptions and methods for applied analysis.* Boston University School of Public Health, Department of Health Policy and Management, 2013.

[13] Richard Ernest Bellman; Rand Corporation. *Dynamic programming.* Princeton University Press, 1957. ISBN 978-0-691-07951-6., Republished: Richard Ernest Bellman. Dynamic Programming. Courier Dover Publications, 2003. ISBN 978-0-486-42809-3.

[14] Karl Pearson. *On Lines and Planes of Closest Fit to Systems of Points in Space.* Philosophical Magazine 2 (11): 559–572. doi:10.1080/14786440109462720.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. ISBN 1-57735-004-9. CiteSeerX: 10.1.1.71.1980

[16] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander. *OPTICS: Ordering Points To Identify the Clustering Structure.* ACM SIGMOD International Conference on Management of Data. ACM Press. pp. 49–60.

[17] Elke Achtert, Christian Bohm, and Peer Kröger. *DeLiClu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking.* Proc. 10th Pacific-Asian Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD'06), Singapore, 2006.

[18] Elke Achtert, Christian Böhm, Peer Kröger, Arthur Zimek. *Mining Hierarchies of Correlation Clusters.* Proc. 18th Int. Conf. on Scientific and Statistical Database Management (SSDBM'06), Vienna, Austria, 2006.

[19] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, Arthur Zimek. *Finding Hierarchies of Subspace Clusters.* Proc. 10th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany, 2006.

[20] Elke Achtert, Christian Bohm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, Arthur Zimek. *Detection and Visualization of Subspace Cluster Hierarchies.* Proc. 12th Int. Conf. on Database Systems for Advanced Applications (DASFAA '07), Bangkok, Thailand, 2007.

[21] Karin Kailing, Hans-Peter Kriegel, Peer Kröger. *Density-Connected Subspace Clustering for High-Dimensional Data.* Proc. 4th SIAM Int. Conf. on Data Mining, pp. 246-257, Lake Buena Vista, FL, 2004.

[22] Rokach, Lior, and Oded Maimon. *"Clustering methods." Data mining and knowledge discovery handbook.* Springer US, 2005. 321-352.

[23] Trupti M. Kodinariya and Dr. Prashant R. Makwana. *Review On Determining Number of Cluster in K-Means Clustering.* International Journal of Advance Research in Computer Science and Management Studies Volume 1, Issue 6, November 2013.

[24] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[25] Ivan Solt, Anna Tomin, Krisztian Niesz, ChemAxon Ltd. *New Approaches to Virtual Screening.* Drug Discovery and Development Magazine, December 2013.

[26] Kim S, Thiessen PA, Bolton EE, Chen J, Fu G, Gindulyte A, Han L, He J, He S, Shoemaker BA, Wang J, Yu B, Zhang J, Bryant SH. PubChem Substance and Compound databases. Nucleic Acids Res. 2016 Jan 4; 44(D1):D1202-13. Epub 2015 Sep 22 [PubMed PMID: 26400175] doi: 10.1093/nar/gkv951

[27] David Hoksza, Petr Škoda. *Using Bayesian Modeling on Molecular Fragments Features for Virtual Screening.* IEEE International Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2016), Chiang Mai, Thailand, IEEE, 2016.

[28] *ELKI: Environment for Developing KDD-Applications Supported by Index-Structures; Online Open Source Code.* https://elki-project.github.io.

[29] MetaCentrum. *MetaCentrum - Virtual Organization, Online Web Page.* https://metavo.metacentrum.cz/en/. 2016.

[30] Daniel Szisz. *Online Image.* Chemical Hashed Fingerprint, Chemaxon. http://www.chemaxon.com/jchem/doc/user. 2014.

[31] Ondrej Micka. *Covariance Neighborhoods.* 2016.

[32] Alexey Grigorev. *ROC Analysis — MLWiki.* http://mlwiki.org/index.php. 2016.

[33] Database Systems Group, LMU Munich. *HiSC (Hierarchical Subspace Clustering).*

List of Figures

Attachments

Heatmap Analysis

In order to better visualize DiSH's performance in our artificial datasets, we use various *heat map* visualizations, as shown below:

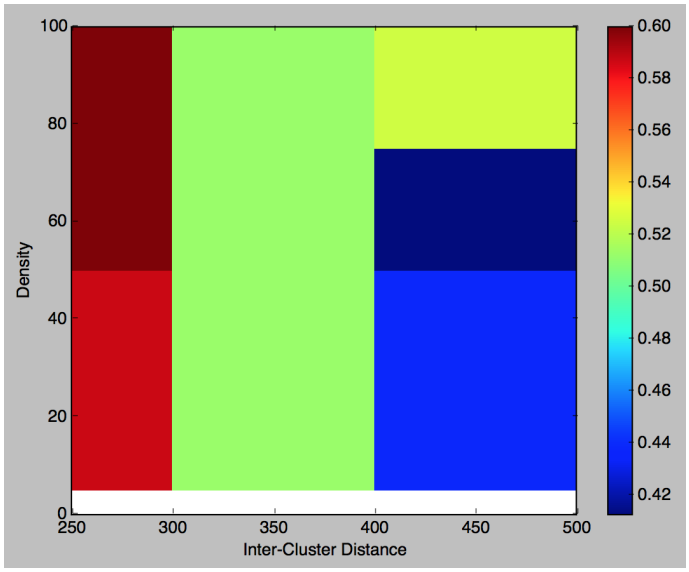

Figure 6.1: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *InterClusterDistance* and *Density* fixed, and varying the other parameters.

The heat map above is an intuitive way to understand how DiSH reacts to different parameter combinations; in the example above, we see different values for the $InterClusterDistance$ parameter along the x-axis and different values for the $Density$ parameter along the y-axis. The specific coloring on the grid at point (x, y) represents the **average score** of datasets generated with $InterClusterDistance$ fixed at x, $Density$ fixed at y, while other parameters are varied.[1]

Quick Note. Observe that the $InterClusterDistance$ parameter is always set to be at least twice as big as the $Density$ parameter. This is done purposefully, as it is reasonable to expect that any datasets that have verifiable clusters satisfy the property that the $InterClusterDistance$ between every 2 cluster centroids is greater than $2 * Density$.

[1] This averaging of the scores is the reason why the scale does not go from the $[0, 1]$ interval, but rather a more constrained interval.

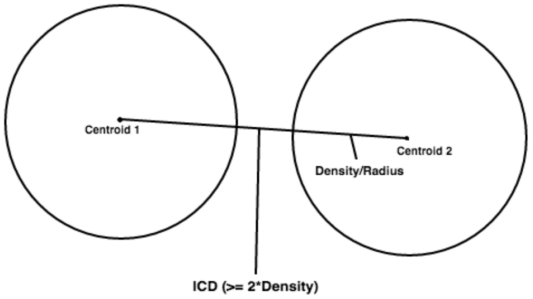

Figure 6.2: A simple visualization of why, intuitively, we expect that $InterClusterDistance \geq 2 * Density$.

Similarly, other parameters' value ranges used in the generation of the clustering datasets are set with some intuitive reasoning in mind, and we won't bore the reader with details regarding all the arguments as to why these ranges were chosen over others.

In order to expedite the reader's progress, we present below an abridged version of a full-blown analysis of the various heat maps. Afterward, we briefly discuss the ramifications of our analysis not only when choosing appropriate input parameters for DiSH when analyzing real molecular data, but also when debugging the aforementioned analysis to see why it is not performing as we expect it to.

Most Important Trends: Heatmap 1

$InterClusterDistance$ and $Density$. (Figure 6.1)

It seems that, in general, the accuracy of DiSH increases as the ICD parameter attains lower values. Moreover, when combined simultaneously with higher values for $Density$, we achieve the highest scores. Indeed, this trend seems slightly counterintuitive: we expect that as we set *higher values* for ICD and *lower values* for $Density$, we would achieve better scores as the clusters have a larger space between them in the feature space and are therefore easier to detect.

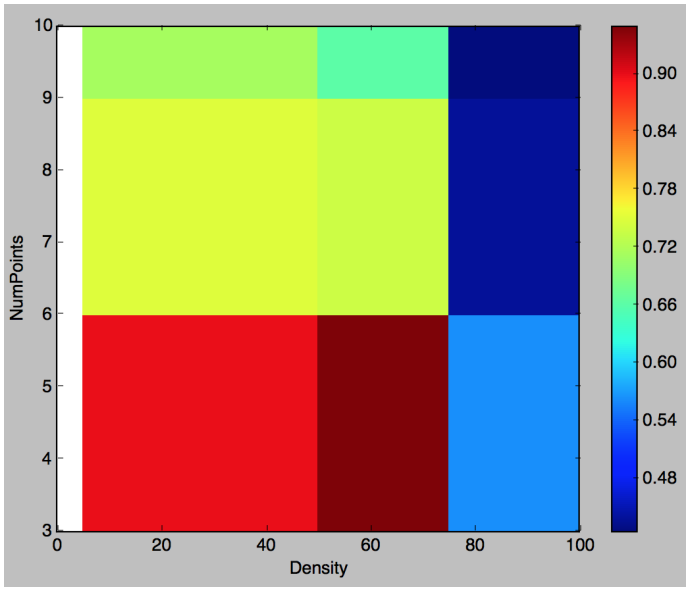

Figure 6.3: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *Density* and *NumPoints* - or μ - fixed, and varying the other parameters.

Most Important Trends: Heatmap 2

Density and *NumPoints*. (Figure 6.3)

Unlike Heatmap 1, the trends visible in Heatmap 2 are largely more intuitive: we see that the score depends *inversely* on both the *Density* and *NumPoints* parameters. It is easy to see why this is the case: as *NumPoints*, an input parameter to DiSH itself, decreases, the criterion for expanding a cluster becomes less strict. Combined simultaneously with a low value for *Density* - which enables the points in the cluster to be packed more compactly and therefore more easily detected by a range query - we achieve a perfect combination of parameters that enables high rates of detection by DiSH.

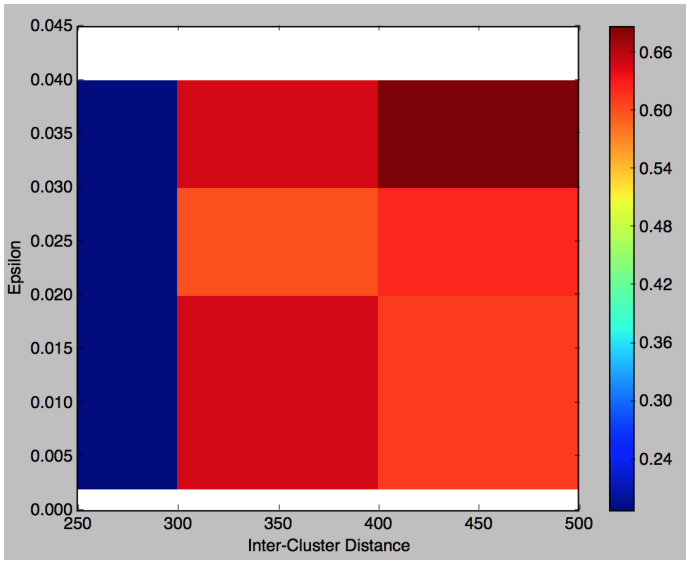

Figure 6.4: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *InterClusterDistance* and *Epsilon* fixed, and varying the other parameters.

Most Important Trends: Heatmap 3
InterClusterDistance and *Epsilon*. (Figure 6.4)

The trends in this heat map are slightly more enigmatic compared to the previous heat maps. It seems that the only visible trend is that a higher value for *InterClusterDistance* produces less accurate results. This trend is, however, contradictory with the findings from our analysis of Heatmap 1, where a lower value for the *InterClusterDistance* parameter resulted in *more* accurate scores.

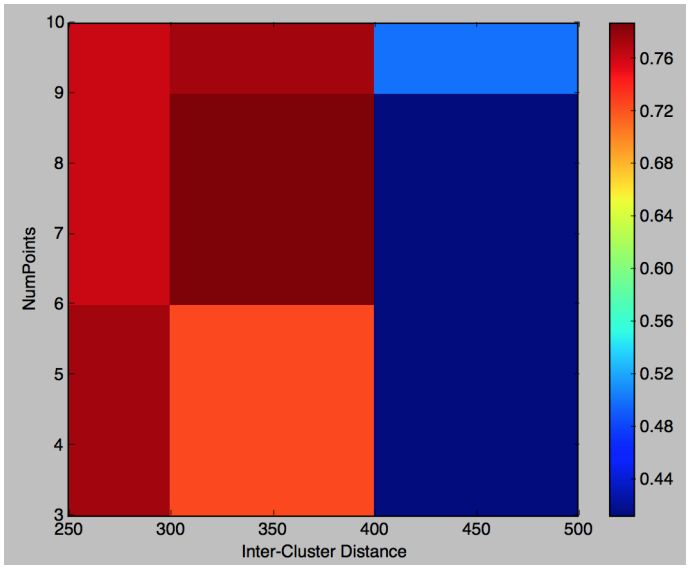

Figure 6.5: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *InterClusterDistance* and μ fixed, and varying the other parameters.

Most Important Trends: Heatmap 4
ICD and $NumPoints$. (Figure 6.5)

The analysis of Heatmap 4 is intriguing: as the *InterClusterDistance* parameter is set lower, we achieve higher accuracies. This is in direct contradiction to Heatmap 3, where a higher *InterClusterDistance* enabled better results, but consistent with Heatmap 1, where the opposite occurred. We will resolve this discrepancy in our empirical analysis in the consolidated heat map analysis a few paragraphs below. The *NumPoints* parameter, on the other hand, is quite consistent with what we saw in Heatmap 2 - the lower the value for *NumPoints*, the higher the scores we achieve.

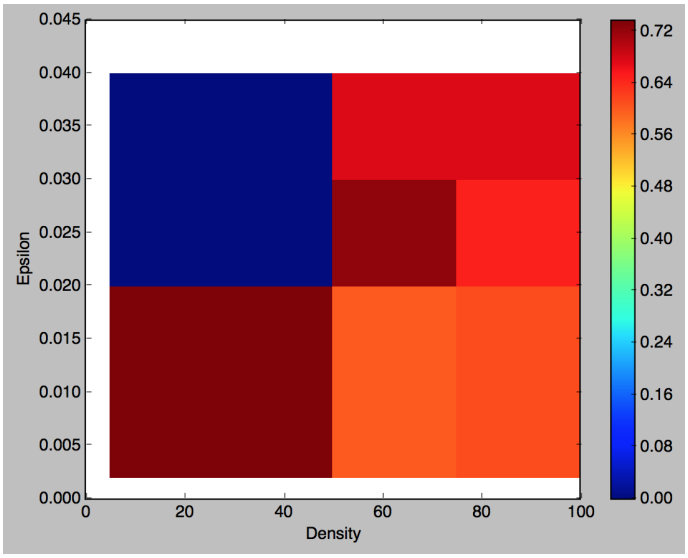

Figure 6.6: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *Density* and *Epsilon* fixed, and varying the other parameters.

Most Important Trends: Heatmap 5

Density and *Epsilon*. (Figure 6.6)

Heatmap 5 is quite consistent with Heatmap 2 in the sense that a lower value for *Density* seems to be conducive to higher scores. Moreover, unlike Heatmap 3, we observe a discernable trend for *Epsilon*: lower values for *Epsilon* seem to result in higher scores, in general.

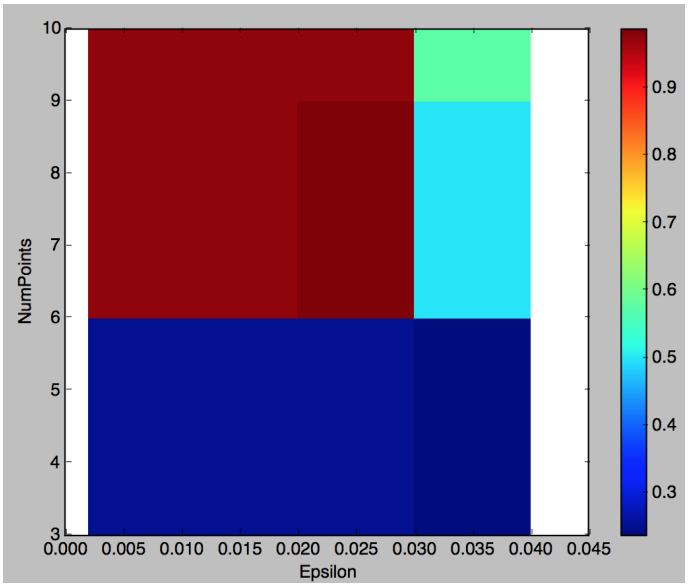

Figure 6.7: A heat map showcasing DiSH's average score on artificial data observed when keeping different value combinations of parameters *NumPoints* and *Epsilon* fixed, and varying the other parameters.

Most Important Trends: Heatmap 6

ϵ and $NumPoints$ (or μ). (6.7)

Unfortunately, Heatmap 6 doesn't bring with it any notable advances to our understanding of the heat maps. Indeed, Heatmap 6 displays absolutely no trend from the perspective of ϵ; in the case of $NumPoints$, Heatmap 6 seems to directly contradict the observations made in Heatmap 4 that a lower value of $NumPoints$ leads to more accurate cluster detection. We, therefore, can't glean any substantial information from observing Heatmap 6 in isolation; we need to consolidate our analyses of different heat maps together in order to arrive at meaningful conclusions.

Consolidated Heatmap Analysis

We have gained a respectable amount of insight from observing each heat map in isolation, but the analysis is not complete: indeed, different heat maps seem to connote contradictory information or, in the case of some, no information at all. In order to reach justifiable conclusions on a macro level, we need to perform the vital step of consolidating the conclusions from the individual heat maps. To this end, it makes sense to proceed parameter by parameter, describing which ranges of parameter values are the most conducive to a fruitful run of DiSH based on the different heat maps:

Consolidated Analysis of InterClusterDistance *Exhibited in Heatmaps 1,3, and 4*

Although our individual analyses of the heat maps resulted in contradictory observations as to how the value of ICD affects the results of DiSH, we may resolve the conflicts by observing the *magnitude* of the scores achieved by DiSH for different values for $InterClusterDistance$ across the heat maps. We will showcase the meaning of the prior statement by example. Although Heatmaps 1 and 4 both seem to indicate that a lower $InterClusterDistance$ results in more accurate results, we can make a few reasonable observations. Heatmap 4, although in general returning higher accuracies for low values for $InterClusterDistance$, seems

to perform optimally for *intermediate-sized* values of $InterClusterDistance$: indeed, we achieve darker shades of red - indicating higher scores - at points where the $InterClusterDistance$ value is not located in either extremes. This sentiment seems to be echoed in Heatmap 3, where, with the sole exception of one patch of very dark red, the higher scores seem to occur when $InterClusterDistance$ takes on values that are not too big nor too small. Heatmap 1, unfortunately, doesn't exhibit any such trends; however, the average scores achieved by lower values of $InterClusterDistance$ in Heatmap 1 are in fact *lower in magnitude* than the scores attained by intermediate values in Heatmaps 3 and 4 (approximately .6 vs. .64 and .8, respectively). We should, therefore, value trends apparent in Heatmap 1 less than we do with trends in Heatmaps 3 and 4.

Conclusion: The highest scores occur when the $InterClusterDistance$ is not too big nor too small with respect to the overall value range.

Consolidated Analysis of Density *Exhibited in Heatmaps 1,2, and 5*

Similarly as with the $InterClusterDistance$ parameter, the heat maps for the $Density$ parameters seem to convey contradictory beliefs: Heatmap 1 indicates that higher values of $Density$ result in higher scores, while the inverse is true for heat maps 2 and 5. However, observing the *scores* of the various heat maps, as we did with the $InterClusterDistance$, a clear winner emerges from the dichotomous trends; the scores exhibited in Heatmap 1 for high values of $Density$ are around .6, whereas the scores exhibited in heat maps 2 and 5 for lower values of $Density$ are .9 and .7, approximately.

Conclusion: The highest scores occur when the $Density$ parameter is small compared to the overall value range.

Consolidated Analysis of ϵ *Exhibited in Heatmaps 3,5, and 6*

Even when considering in toto heat maps 3,5, and 6, it is hard to argue about a reasonable value for ϵ, simply because no apparent trends were visible in heat maps 6 and 3. We, therefore, don't have enough evidence to make conjectures about values of ϵ that are conducive to high scores, even when considering the partial trends visible in heat map 5.

Conclusion: ϵ exhibits no general trends; its effect is simply too dependent on other parameters.

Consolidated Analysis of NumPoints *Exhibited in Heatmaps 2,4, and 6*

Last but not least, our individual analysis of the heat maps produced a contradiction about the $NumPoints$ parameter and therefore we need to extract a collated insight about the effects of different value ranges of $NumPoints$. In heat maps 2 and 4, we achieve quite intuitively higher scores for lower values of $NumPoints$, whereas the opposite occurs in Heatmap 6. When once again using the magnitude of the scores as a tie breaker, we see that both heat maps 2 and 6 achieve extremely accurate scores - more than .9 - for low and high values of $NumPoints$, respectively. Heatmap 4, on the other hand, achieves a slightly less but still very respectable accuracy of around .7 for lower values of μ. We, therefore, have two witnesses versus one, and consequently conclude that *lower* values of μ, or $NumPoints$, do indeed result in higher scores.

Conclusion: Lower values of $NumPoints$, with respect to the average number of points per cluster in our distribution, result in higher scores.