2018

# Algorithms in protein cavity classification

Georgi Dimitrov Georgiev
*Lehigh University*, gosh.georgiev@lehigh.edu

# Algorithms in protein cavity classification

by

Georgi Georgiev

A Thesis
Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Computer science

Lehigh University
August 2018

Approved and recommended for acceptance as a thesis in partial fulfillment of the requirements for the degree of Master of Science.

Date

Thesis Advisor

Chairperson of Department

## Acknowledgements

I would like to thank my family for encouraging me and supporting me in my decision to study in the US. I will do my best to exceed their expectations and return their kindness.

I would like to thank Lehigh University for accepting me for my undergraduate and graduate degree and supporting me financially through scholarships and teaching assistantships. Without that support, my journey would have been harder or even impossible.

I would like to thank Professor William Best for teaching me the beauty of engineering and the essence of design.

I would like to thank Heidi Wegrzyn, Jeanne Steinberg and Bryan Hodgson for the hard work they put into supporting me and everyone else in the CS department.

I would like to thank Lehigh's CSE faculty for their lessons and guidance.

I would like to thank Professor Brian Chen for giving me the opportunity to work with him. Starting from the summer of my sophomore year, we spent endless hours talking about research, academia and life and our meetings have always been something to look forward to. He encouraged me to explore my interests and was willing to risk starting new projects based on my interests. He taught me how to be a great scholar and how to be a great person. I will always remember his advice and kindness.

# Contents

# List of Figures

# Abstract

The exponential rise of protein structures lets us explore and visualize the sequence and structure of proteins. The structures are a great and underexplored source of insights into the physical and chemical properties of proteins. In order to process and analyze them at scale, we need to come up with a suitable digital representation which will allow structural biologists to analyze them using statistical and machine learning models. A challenge with generating these digital representations is generating them accurately and preserving their accuracy across a series of operations on the structures. This thesis presents an arbitrarily precise algorithm for constructing 3D molecular structure representations and defines a useful set of operations that preserve their accuracy. The thesis also details experiments using those digital representations proving their power and usefulness.

# 1 Introduction

Every year structural biologists publish an increasing amount of macromolecular structure scans of proteins. Despite the abundance of these scans, they are greatly understudied due to the lack of a good representation for computational analysis. The scans, however, contain relational information about atoms which can be used in statistical and machine learning tools to gain insights into the function and properties of proteins. In order to feed the structure scans into statistical and machine learning models, we need to design a representation and operations defined on that representation.

The most commonly used representations for structure scans are collection of points, arcs or triangle meshes. Those representations are mostly used for visualization and the methods that use them typically produce the representations at fixed precision since additional detail does not add new information. In addition, even if the representations are produced at higher precision, the added detail is not biologically significant, which renders them unsuitable for structure comparison. To address this problem, this thesis presents an arbitrarily precise algorithm for constructing structure representations and defines a set of operations which preserve precision after any amount of operations. We also detail experiments using the new representation proving the power and usefulness of the representation.

The arbitrary precision algorithm takes in macromolecular structure and generates an arbitrarily precise representations of those macromolecules. It was necessary to design the algorithm to handle arbitrary precision due to the biological significance of low-level details in molecular surfaces. The algorithm achieves this precision via utilizing all available cores of a machine and scales with respect to an increasing number of cores. It is compatible with previous computational biology tools while preserving the quality of input and output structures. The work defines a series of operations - union, difference, intersection, which are used to productively work with the generated representations. The algorithms superiority is demonstrated by comparing it to existing applications in terms of performance and accuracy as well as in a practical applications.

The experiments using the new representation are a collection of traditional classification algorithms that are compared and evaluated on two datasets of molecular binding cavities in order to assess their applicability to classifying protein binding preferences. The datasets were made by simulating 2 protein subfamilies using molecular dynamics software, selecting generating polyhedral mesh representations of the ligand bindings sites, and then generating a voxel representation of the cavities. The voxelized representations are then split into training and testing datasets and fed into the following machine learning algorithms: decision trees, naive bayes classifier, k nearest neighbor classifier, and support vector machine classifier. The algorithms were selected for evaluation based on previous research with volumetric or protein data. They were evaluated based on an exhaustive leave-k cross validation in which we tested all possible protein combinations from the two superfamilies. We concluded that the best classifier for these datasets is the support vector machine classifier which performs best even with a small subset of the available data and on multiple resolutions.

This thesis is organized as follows: Section 2 will give a short background into generating and working with geometric primitives and will focus on the interface primitive which enables incorporating surfaces generated by other software. It will also show performance and applicability comparisons with other algorithms. Section 3 will describe the design and implementation of an experiment on the applicability of machine learning algorithms on volumetric representations of proteins. And finally, Section 4 will summarize the findings and results of the previous two sections and conclude.

## 2 Exact solid representations of polyhedral surfaces

### 2.1 Motivation

The method of representation is an algorithm for generating and working with arbitrarily precise molecular representations. It consists of three logical parts. The first part is an algorithm which takes in atomic coordinates and generates a power diagram. The power diagram is a voronoi-like diagram which is then used to produce a dual geometric graph where the vertices are atom coordinates and the edges correspond to adjacency between atoms. The dual graph serves as a blueprint for the whole molecular surface. The second part is a collection of geometric primitives which serve as the building blocks of the molecular surface. The third part is a set of operations which enable the geometric primitives to come together into a single molecular surface. The main contribution of this thesis to the larger project is the geometric primitive called MCMesh which interfaces with existing molecular surfaces generated by other software. A second major contribution of this thesis is a comprehensive testing benchmark which proves the superiority and usefulness of the overall algorithm. The parts of the algorithm which were not developed for this thesis, will be paraphrased in order to give context to the thesis but the focus will remain on the thesis contributions.

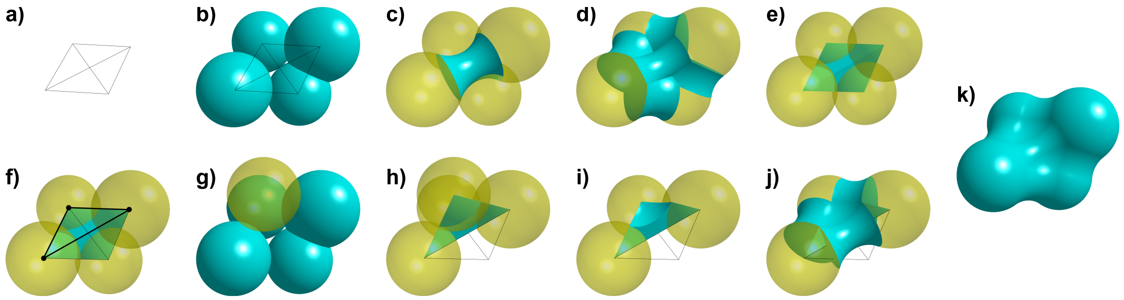


Figure 2.1: Molecular Surface Construction. a) Dual graph of a power diagram on four atoms (graph edges shown with black lines, graph vertices shown as corners). b) Sphere primitives from atoms (teal) shown with dual graph. c) Atoms (transparent yellow) with one spindle (teal). d) Atoms with spindles corresponding to all edges of the dual graph. e) Tetrahedron primitive (teal) with atoms (yellow). f) One triangle of the dual graph (bold lines, black circles) that is not between tetrahedra. g) Solvent sphere (yellow) tangent to three of the atoms (teal). h) New tetrahedron (teal) with corners in the center of the three atoms of the triangle and the solvent sphere (yellow). i) Cup region inside the new tetrahedron and outside the solvent sphere (teal) shown with three atoms of the triangle (yellow). j) Cup, shown with three adjacent spindles (teal) and three atoms of the triangle (yellow). k) finished molecular solid.

The algorithm first acquires a power diagram using the REGTET[3] program. The power diagram is a dual graph where the vertices correspond to atoms from the scan and the edges correspond to atom adjacency. The algorithm constructs the molecular surface by creating a series of primitives and combines them using operations defined on the primitives. The defined primitives are spheres, spindles, tetrahedrons, molecules, and meshes and the defined operations are union, difference and intersection. The algorithm constructs spheres using atom coordinates taken from the vertices of the power diagram, and the edges of the power diagram help create the spindles and tetrahedrons. The constructed spheres, spindles and tetrahedrons are then combined using the defined operations into a constructive solid geometry(CSG) binary tree and sit in the leafs of the tree while each non-leaf node is a defined operation on the primitives. The final result of the CSG tree resolution is a molecule primitive. The only remaining primitive is the mesh(MCMesh) primitive which represents closed triangle meshes. The purpose of the primitive is to interface with existing software whose output is triangle meshes.

The interface between all the primitives and operations is defined through 3 abstract functions which are implemented by each primitive: containsPt, intersectSegment, and findStartingCubes. containsPt generally takes a point and returns a value indicating whether the point is inside the primitive or not. IntersectSegment takes a segment and returns a vector of intersection points. findStartingCubes returns cubes which have some of their corners inside and some of their corners outside of the primitive.

### 2.2 Methods

MCMesh which is a geometric solid primitive that enables working with triangle meshes generated by other software. To construct such a primitive, we first construct a cubic lattice which surrounds the mesh and then we associate all triangles of the mesh to one or more cubes which contain a triangle in parallel. During this step, all cubes that contain a triangle are classified as non-empty and all the ones that do not contain a triangle are classified as empty. After this, we perform a breadth first search to connect groups of empty cubes into groups of adjacent empty cubes which we call connected components. Next, we determine the nested structure of the empty group components by observing the adjacent cubes of a component. We start from the first outermost component which is by lattice design outside of the protein structure and we go inwards into the mesh structure marking even numbered components inside and odd numbered components outside. In the abnormal case where a mesh has an internal void extremely close to the surface, the corresponding cube might get incorrectly classified. We deal with this by projecting a line segment between sets of empty cubes and we count intersections between cubes. An odd number of intersections means the two components have an opposite interior-exterior status, while an even number of intersections means the two components have the same interior-exterior status. The result of all these computations is that we get all connected components and their exterior/interior status.

The way MCMesh interfaces with the rest of the geometric primitives is by implementing the abstract methods that define the The way MCMesh interfaces with the rest of the geometric

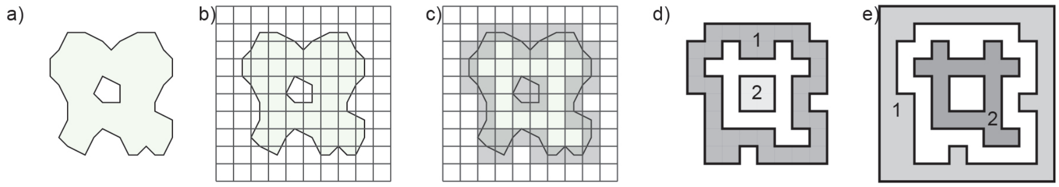


Figure 2.2: Processing Polyhedral Meshes. a) Polyhedral mesh (light green), with boundary triangles drawn as black segments. An internal void (white, center) with boundary line segments. b) A cubic lattice surrounding the mesh. c) Non-empty cubes (gray squares) and empty cubes (transparent squares). d) Two distinct connected components of non-empty cubes (numbered). Two distinct connected components of empty cubes (numbered).

primitives is by implementing the abstract methods that define the interactions between the primitives: containsPt, intersectSegment and findStartingCube.

containsPt takes a point as input and returns whether the point is inside or outside the surface. It decides this by finding the cube on the lattice which the point belongs to if the cube empty, the exterior/interior status dictates the whether its inside or outside. If the cube is non-empty, the status of the point is determined by 5 random segments projected towards the centers of adjacent cubes. The majority of the adjacent cubes status dictates the status of the point.

intersectSegment takes a segment as input and returns a vector of intersections and inside/outside status of each segment. It finds all cubes on the segment, and then collects all the triangles from those cubes. Then we compute all the intersections with those triangles and then propagate the inside/outside status through the segments.

findStartingCubes takes a cubic lattice as input which is different from the computed lattice defined by the construction of the MCMesh primitive. We construct a set of non-empty cubes from each connected component computed on the MCMesh lattice and then we find the cubes in the first lattice that overlap with the one from MCMesh. If the overlapping cubes have inside and outside corners, we mark them as starting cubes.

## 2.3 Implementation information and testing environment

The algorithm is implemented in C and C++. A C wrapper supports REGTET, a fortran program for computing power diagrams. Parallel communication and coordination was achieved in part with Intels Threading Building Blocks template library (TBB) employs a work stealing scheduler to balance computational loads across multiple cores. Benchmarks were performed on a workstation with two Xeon E5-2609 CPUs running at 2.5 Ghz, with 32GB of ram, and on an attached Xeon Phi 7120P coprocessor with 61 cores running at 1.24Ghz and 16GB ram. Xeon Phi and Xeon CPU benchmarks were never run simultaneously.

## 2.4 Results

We test our implementation by measuring its performance computing simple and sophisticated operations. We also test it on a practical application to demonstrate the applicability of the algorithms.

In Section 2.4.1, we measure the performance of computing simple surfaces composed of geometric primitives and we compare the performance with VASP, an established software package that performs similar CSG operations. We perform the experiments on a regular workstation CPU as well as on a manycore coprocessor CPU in order to test the parallelism in our algorithm.

In Section 2.4.2, we measure how accurate the generated surfaces are compared to the trollbase library, an established tool for molecular surface generation in GRASP2 [12], VASP-E [4], and MarkUs [7].

In Section 2.4.3, we test whether the new, more accurate representations are useful in a statistical model and we compare the results with representations generated by VASP.

### 2.4.1 Parallel performance and scaling

In this experiment, we test the performance of our algorithm computing a series of CSG operations and we compare the runtime with VASP. The series of CSG operations is a set of unions on 30 randomly generated spheres, spindles, and tetrahedrons, with mesh outputs to be generated at resolutions 1.0Å, .5Å, .25Å  and .125Å. CSG trees of these unions were balanced binary trees, but imbalanced trees yielded essentially identical runtimes. Since VASP does not use primitive representations, triangle meshes nearly identical to the primitives generated for our algorithm. The runtime of our algorithm and the runtime of VASP are shown in figure 2.3 with CPU core ranging between 1 and 8 CPU cores. For both algorithms, runtimes included the time necessary to generate triangulated meshes of the output, in addition to the CSG operations themselves. We distinguish Xeon CPU cores from Xeon Phi cores by referring to them as CPU and PHI cores.

On a single CPU core, our algorithm required .113 seconds to compute the CSG union on the 30 primitives at 1.0Å  resolution, whereas 9.492 seconds were required for a single core to compute the same union at .125Å  resolution. As the number of CPU cores increased to 8, runtime rapidly diminished to .03 seconds to compute the union at 1.0Å  resolution, and 1.465 seconds to at .125Å. In contrast, single-threaded VASP required 3 seconds to compute union on 30 primitives at 1.0Å  resolution, and 64 seconds at .125Å  resolution. It is clear that our algorithm outperforms VASP, the current state of the art, on this union of geometric primitives.

We ran the same union operations on 8, 16, 32, and 60 Xeon Phi cores. Due to the slower speed of PHI cores relative to CPU cores, runtimes were slower even though the same number of computing threads were used in some cases. As the number of utilized cores increased, runtimes exhibited sublinear improvement (fig. 2.4), because communication overhead increases with the number of parallel threads. Runtimes for unions on coarser resolutions improved less than for finer resolutions. This difference in parallel speedup (fig. 2.5) arises from the fact that the problem size for coarser resolutions is already quite small and communications and setup time outweigh the advantages of parallelism. In contrast, finer resolutions create more computation to be divided, justifying the costs of communications and setup.

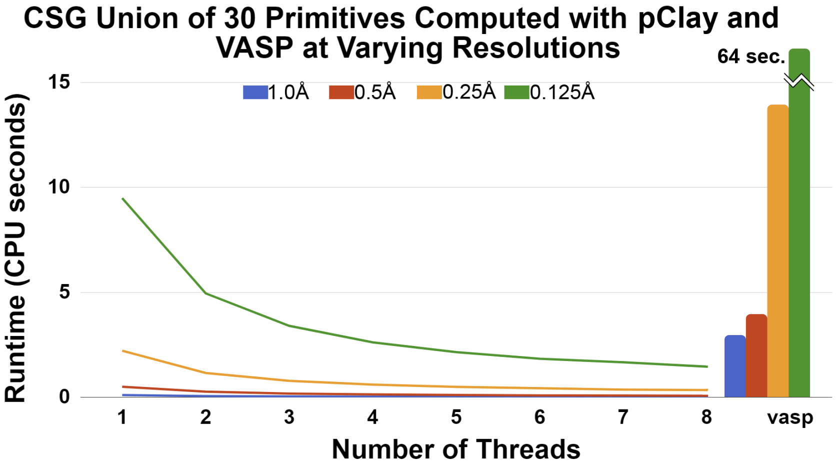


Figure 2.3: Geometric primitive generation Xeon CPU

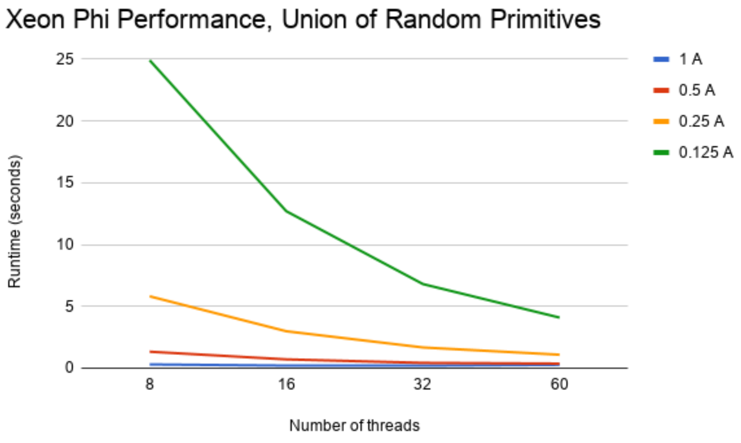


Figure 2.4: Geometric primitive generation Xeon Phi coprocessor

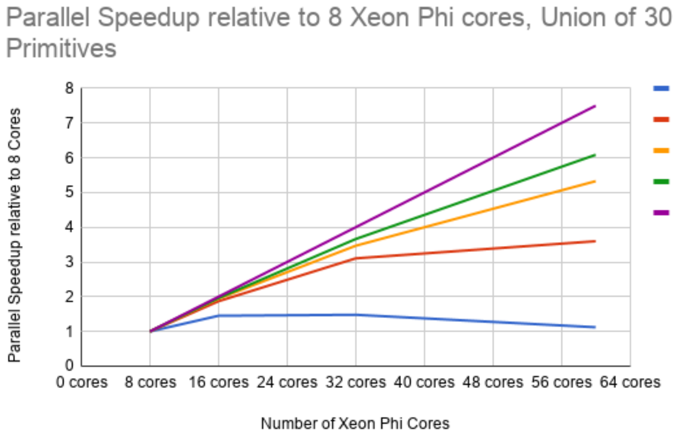


Figure 2.5: Parallel speedup for primitive generation on Xeon Phi

### 2.4.2 Accuracy of molecular solid generation

While the generation of molecular surfaces is not the primary purpose of our algorithm, accurate comparisons require accurate molecular solids. To evaluate the molecular solids produced by our algorithm, we compared them to molecular surfaces generated with the trollbase library, which generates surfaces for several widely used software tools, including GRASP2 [12], MarkUs [7], and VASP [6].

First, we compared the volume within surfaces generated by our algorithm to the volume within surfaces generated by trollbase. Surfaces produced by our algorithm contained .00173% greater volume, on average, than those generated with trollbase. The largest volume difference was observed between surfaces generated for yeast RPN14 (pdb: 3VL1, chain A). That percentage difference was .02343%, and it arose from many small variations, accumulating to a total difference of 11.594Å$^3$. The surface from our algorithm contained 49,460Å$^3$ and the trollbase surface contained 49,471Å$^3$.

Second, we measured displacement distances throughout surfaces generated with our algorithm and those of the same protein generated with the trollbase library. Molecular surfaces approximated from solids produced with our algorithm were polyhedral meshes with an average of 197,718.54 points. The average displacement distance over all surface points, averaged over all proteins, was 0.00383 Å. Average displacement distance varied only within a narrow range, having a standard deviation of 0.0004 Å. The smallest average displacement distance, observed on a UVB resistance protein (pdb: 4DNU, chain A) was 0.00315 Å and the largest average displacement distance, observed on a segment of an acetylcholine receptor (pdb: 1A11, chain A) was .00506 Å. Over the entire dataset, the average maximum displacement distance was .13619 Å. The largest maximum displacement distance in the entire dataset was .22024 Å, observed on proto-oncogene C-FOS (pdb: 2WT7, chain A). In this case and in others, the reason that maximum displacement on any protein can even rise to these modest levels stems from the fact that very thin spindles can occupy volume inside a lattice cube without occupying any corner of the cube, preventing it from being part of the triangular mesh output.

Overall, the volume within of molecular surfaces generated with our algorithm and trollbase are nearly identical and the distances between the two surfaces, evaluated at many points, are very close. These results demonstrate the our algorthm produces very accurate solids.

### 2.4.3 Evaluating our algorithm on existing applications

The added precision of our algorithm creates several new applications. We demonstrate one such application by producing training data for VASP-S, a statistical model for detecting differences in ligand binding specificity with steric causes [5]. VASP-S is trained on the volumes of individual CSG differences computed from cavities with the same binding specificity. This training enables VASP-S to estimate the probability (the $p$-value) that two cavities have similar binding preferences. If $p$ is lower than a threshold $\alpha$, the VASP-S rejects the possibility that two cavities have

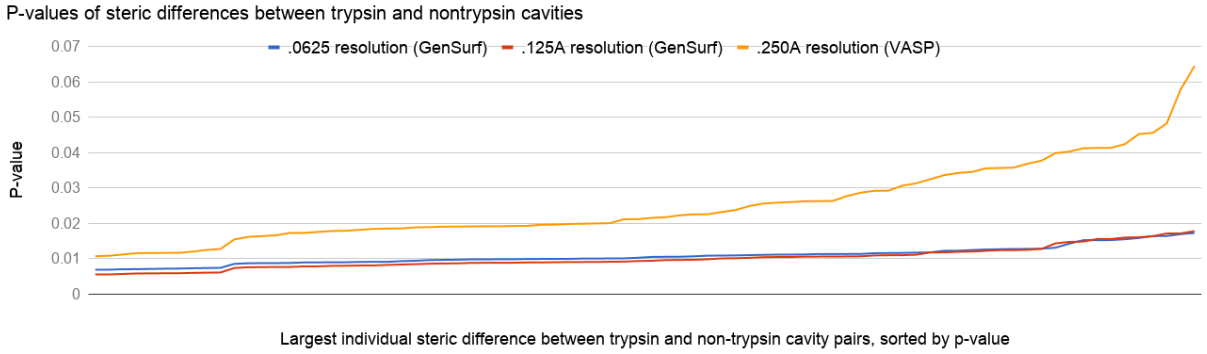


Figure 2.6: Trypsin and non-trypsin fragment p-values

similar binding preferences, and predicts that they have different specificities.

We hypothesize that training the VASP-S model with data generated at finer resolutions will produce more accurate predictions than a VASP-S model trained with coarser data. To test this hypothesis, we used cavities from the trypsins, which all prefer to bind positively charged amino acids. Three training sets were constructed from these cavities by generating all possible CSG differences between all pairs of trypsins. VASP was used to produce a copy of the training set at 0.25 Å resolution and our algorithm was used to produce the same set at resolutions of 0.125 Å and .0625 Å. We then computed CSG differences between every trypsin and every nontrypsin, at these resolutions. Finally, we estimate the $p$-value of the largest CSG difference between every trypsin and every non-trypsin, at all three resolutions (fig. 2.6). Since the non-trypsins prefer either large aromatics or small hydrophobics, we expect VASP-S to produce a low $p$-value these CSG differences.

Using the conservative $\alpha$ threshold of 2%, when trained at 0.25 Å resolution, VASP-S predicts that 43 of the 81 CSG differences between trypsin and non-trypsin cavities had different binding preferences. This discrepancy indicates 43 false negative predictions, where VASP-S incorrectly overlooked cavities with different binding preferences. However, when trained at 0.125 Å or 0.0625 Å resolution, VASP-S correctly predicts that all CSG differences were from cavities with different binding preferences. Statistical models trained with our algorithm had a 0 false negative rate. These results demonstrate that our algorithm can provide precision sufficient to ensure that existing aggregate methods do not lose accuracy by overlooking useful predictions.

# 3 Classification of ligand binding cavities

## 3.1 Motivation

In the previous section, we introduced a part of a larger algorithm which creates arbitrarily precise molecular surfaces and defines a set of operations that we can use to work with them. We described a statistical model that is able to accurately predict if two cavities have different binding affinities. In this section, we describe our experiment where we test if we can train machine learning models on the more accurate representations and successfully predict the binding affinity of an unknown cavity. The reason why this problem is important is that the ability to classify proteins by their binding preferences enables us to predict whether a drug will serve its purpose or not depending on a patients circumstances.

We designed two datasets of protein binding cavities and we trained machine learning models on them. We explain in detail our model selection process in order to document that a variety of machine learning algorithms are viable for the problem and not just the best-performing one. We selected and evaluated a variety of classification algorithms based on previous applications of the algorithm on similar representations or because of their availability. The methods tested are decision trees, naive bayes classifiers, k-nearest neighbor classifiers and support vector machine classifiers. Our evaluation showed that support vector machines perform the best on both datasets but other follow closely behind the larger the dataset size is and the more accurate the molecular surfaces are.

## 3.2 Dataset

The datasets that we created for our experiment are based on the crystal structures of proteins from the serine protease and enolase superfamilies(Figure 3.1 and Figure 3.2). These superfamilies were selected because an established experimental literature clearly delineates at least three subfamilies in each superfamily with distinct binding preferences. Despite the established experimental literature, the superfamilies present two critical problems that needed to be addressed.

The first problem concerns the size of the datasets. The serine protease superfamily has 12 members and the enolase superfamily has 7 members. Machine learning algorithms require substantially larger datasets in order to become effective so we decided to use data augmentation in the form of protein simulation. Protein simulation is a data augmentation method which uses Brownian motion to create small changes in the area of the binding cavity while preserving its function. This method of data augmentation enables us to enlarge our dataset sizes by orders of magnitude and justifies applying machine learning methods to the problem.

The second problem concerns the asymmetric property of the two datasets. The serine protease superfamily contains 3 subfamilies in the ratio of 9:1:2 with respect to proteins. The enolase superfamily also contains 3 subfamilies in the ratio of 4:2:1 with respect to proteins. This asymmetry proved to be problematic in the initial stages of the method evaluation since it heavily skewed the results of the experiments towards the subfamily with the most amount of proteins. We approached this problem by designing a cross-validation where we exhaustively tested all combinations of leaving proteins out of the training set and into the test set. Furthemore, for each protein, its either completely in the training set, or completely in the test set, i.e., each proteins simulations are never separated. We did this to avoid biasing the classification methods towards a specific protein simulation and also to promote each classifiers generalizability.

### 3.2.1 Serine protease superfamily

Serine proteases hydrolyze peptide bonds through the recognition of adjacent amino acids with specificity subsites numbered S1, S2, , S40. Each subsite preferentially binds one amino acid before or after the hydrolyzed bond between S1 and S10. Cavities in our dataset are derived from the S1 subsite, which binds aromatics in the chymotrypsin subfamily [11], positively charged amino acids in the trypsin subfamily [8], and small hydrophobics in elastases [2].

Serine Protease Superfamily:
Trypsins: 1a0j, 1ane, 1aq7, 1bzx, 1fn8, 1h4w, 1trn, 2eek, 2f91,
Chymotrypsins: 1ex3
Elastases: 1elt, 1b0e

Figure 3.1: PDB codes of proteins used from the serine protease superfamily.

Enolase Superfamily:
Enolases: 1ebh, 1iyx, 1te6, 3otr
Mandelate Racemase: 1mdr, 2ox4
Muconate Lactonizing Enzyme: 2pgw

Figure 3.2: PDB codes of proteins used from the enolase superfamily.

### 3.2.2 Enolase superfamily

Proteins in the enolase superfamily catalyze a reaction that abstracts a proton from carbons adjacent to a carboxylic acid [1]. Opposite an N-terminal "capping domain" [13], the C-terminal domain forms a TIM-barrel, which provides a stable scaffold for amino acids that act as acid/base catalysts for several different reactions [1]. Cavities in our dataset were classifed into three subfamilies. Enolases facilitate the dehydration of 2-phospho-D-glycerate to phosphoenolpyruvate [10]. Mandelate racemases convert (R)-mandelate to and from (S)-mandelate [14]. Finally, muconate-lactonizing enzymes reciprocally cycloisomerize cis,cis-muconate to and from muconolactone [1]. Since members of the Enolase family can exhibit open and closed conformations, only structures with the open conformation were used, for consistency.

In addition to established binding preferences, subfamilies in the serine protease and enolase superfamilies were selected because they all exhibit a collection of sequentially nonredundant, high resolution crystal structures. At least one structure in each superfamily had a bound ligand. We were thus able to define bovine chymotrypsin (PDB: 8gch) as the principal homologous structure in the serine proteases and S. cerevisiae enolase (1ebh) as the principal homologous structure in the enolase superfamily.

### 3.2.3 Data augmentation - molecular dynamics simulation

After selecting the proteins for our datasets, we simulate the two protein sets for 100 nanoseconds and we use the results to generate 600 conformational snapshots of each protein. The software used for the simulations is GROMACS 4.5.4[9]. Each protein was simulated in a cubic water box populated using CPC/E(an equilibrated 3-point solvent model), with the protein centered inside the box. The box size container solute protein structure with 1.0 nanometers space between the the protein and the nearest point on the boundary plane. Fully periodic boundary conditions were used throughout the equilibration and simulation steps. Charge balanced sodium and potassium ions were then added to the solvent at a low concentration.

Energy minimization using a steepest descent algorithm is then performed for the entire system. Isothermal-isobaric(NPT) equilibration is performed in four 250 picosecond steps to allow the solvent to equilibrate temperature and pressure prior to the primary simulation. Starting at 1000 kJ/(mol*nm) over the 1 nanosecond minimization period. Backbone position restraints were released for the primary NPT simulation.

System energies were generated at the start of the equilibration phase. Initial temperature at the start of the equilibration phase. Initial temperature was 300 Kelvin and initial pressure was 1 bar. The Nose-Hoover thermostat was used for temperature coupling. The P-LINCS bond constraint algorithm was used to update bonds, electrostatic interaction energies were calculated by particle mesh Ewald summation (PME). The parrinello-Rahman algorithm was used for pressure coupling. All temperature and pressure scaling was performed isotropically.

Full molecular dynamics simulation is started using the atomic positions and velocities of the final equilibration state. The total simulated duration of the molecular dynamics simulation was 100 nanoseconds, with 1 femtosecond steps. p-LINCS and PME were chosen for their parallel efficiency. OpenMPI was used for node and interprocess communication. Simulations was used for node and inter-process communication. Simulations were run on multiple nodes with 16 cores each, with PME and distribution selected by GROMACS.

After simulations were completed, the trajectory file was converted to a simple protein data bank format with atom positions only. The waterbox was removed and at each timestep, the protein was rigidly superimposed to the original orientation. From these timesteps, we selected 600 conformational samples at uniform intervals for our data sets, and then computed frequent regions.

From the gromacs simulation we receive trajectory files, which are basically sequences of protein data bank files concatenated together. We take those trajectory files and we cut them into individual PDB files which we then turn into molecular surfaces using a method similar to GRASP2. GRASP2 surfaces using Van der Waals radii taken from [43] are exceptionally precise approximations of the molecular surface, averaging 384,461 triangles per surface, and triangular area averaging .062 square angstrom on our data set. Some GRASP2 surfaces contain topological discontinuities where single contiguous surfaces are represented with disconnected patches. Input surfaces exhibiting topological discontinuations were first fixed using Polymender.

### 3.2.4 Simulation results

After simulations were completed, the trajectory file was converted to a simple protein data bank format with atom positions only. The waterbox was removed and at each timestep, the protein was rigidly superimposed to the original orientation. From these timesteps, we selected 600 conformational samples at uniform intervals for our data sets, and then computed frequent regions.

From the gromacs simulation we receive trajectory files, which are basically sequences of protein data bank files concatenated together. We take those trajectory files and we cut them into individual PDB files which we then turn into molecular surfaces using a method similar to GRASP2. GRASP2 surfaces using Van der Waals radii taken from [43] are exceptionally precise approximations of the molecular surface, averaging 384,461 triangles per surface, and triangular area averaging .062 square angstrom on our data set. Some GRASP2 surfaces contain topological discontinuities where single contiguous surfaces are represented with disconnected patches. Input surfaces exhibiting topological discontinuations were first fixed using Polymender.

### 3.2.5 Cavity generation

After the simulations are done, PDB files generated and topological discontinuities fixed, we need to extract the binding cavity from the rest of the structure since were only interested in the cavity. Leaving the rest of the structure will only bloat the dimensionality of the problem and will not contribute significantly to improving the classifier accuracy. To extract the cavities, we perform a series of boolean set operations to generate solid representations of the cavities and we put the solid representation of the surface in the cubic lattice.

The way we generate the solid representations of the cavities is a technique initially developed for VASP. It is paraphrased here for completeness. We begin with atomic coordinates from the protein data bank or from a snapshot from an all-atom molecular dynamics simulation. This structure is first aligned to a principal homologous structure that we selected because it has a

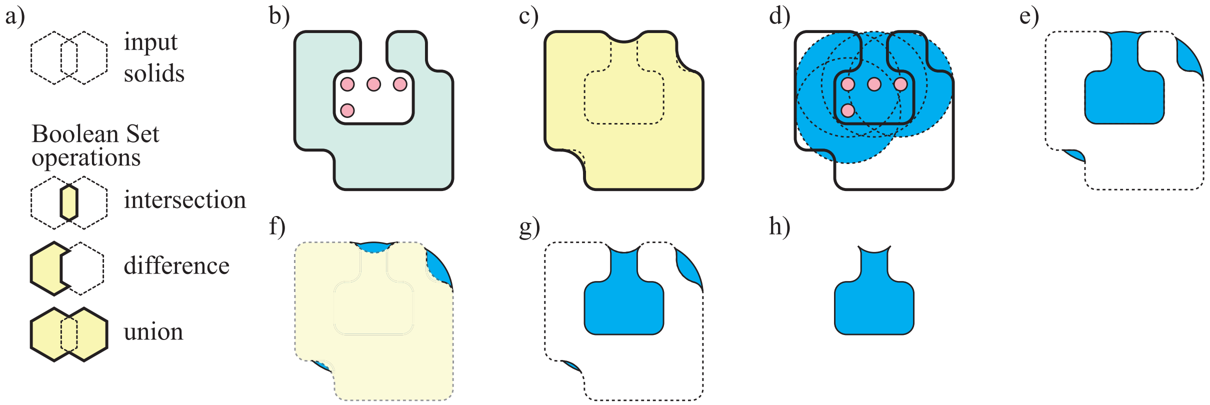


Figure 3.3: Boolean set operations and Cavity Generation. a) Boolean set operations. Input solids are white and bordered by dotted lines. Output solids are yellow with solid lines. b) Molecular surface of a protein (green) with bound ligand (red circles). c) Envelope surface (yellow), superimposed over the molecular surface (dotted). d) Spheres defining the neighborhood of the binding site (blue, dotted lines), superimposed over the molecular surface (black outline). e) Boolean difference of the neighborhood spheres minus the molecular surface (blue) with molecular surface (dotted lines). f) Molecular envelope (transparent yellow) superimposed over the molecular surface (dotted line), and the Boolean difference from part e (blue). g) Intersection of the molecular envelope and the difference from e (blue). h) The binding cavity, with smaller components removed.

bound ligand. Next we use VASP and trollbase to generate a polyhedral mesh approximating the molecular surface. Trollbase follows the standard Shrake-Rupley rolling probe approach. We use a 1.4 angstrom radius probe which approximates the radius of a water molecule so that the surface represents the solvent excluded surface We also generate a molecular envelope using the identical molecular surface algorithm with a prove sphere with radius equal of 5.0 angstrom. The 5 angstrom probe developed first for the discovery of ligand binding cavities in SCREEN represents a ligand excluded region that overlooks ligand binding cavities and clefts. Next, we generate spheres of a 4 angstrom radius, centered on each atom of the ligand in the principal homologous structure. The boolean union of the spheres define the neighborhood of the binding site. We then compute the boolean difference of the sphere union minus the molecular surface to eliminate neighborhood regions that are sterically excluded by the protein. Using the result, we compute the boolean intersection with the molecular envelope, to eliminate regions outside of the binding cleft. Finally, we remove all but the largest disconnected region to eliminate stray particles outside the binding cavity. The resulting geometric solid defines the binding cavity.

### 3.2.6 Voxel representations

After we compute the binding cavity as a geometric solid, we need to further convert it to a vector representation in order to perform the machine learning method evaluation. We take the geometric solid and generate a voxel representation of the solid by quantifying the volume of the cavity. We take the geometric solid and a resolution as input and we generate a cubic lattice with cube size equal to the resolution parameter. The lattice is slightly bigger than the geometric solid as to fit it completely. For each cube, we compute the volume of geometric surface inside of it and quantify it on a scale of 0 to 1 where 0 represents an empty cube and 1 represents a cube full of cavity volume. The three dimensional vector that we get from the cubes on the cubic lattice is then flattened and given as input to the machine learning algorithms.

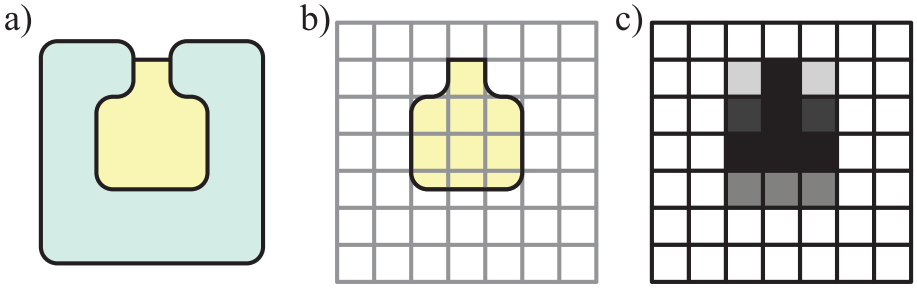


Figure 3.4: Translating a volumetric representation of a binding cavity into voxels. a) The molecular surface of a protein (green), with binding cavity (yellow). b) The superposed lattice of voxels (grey). c) The volume of intersection between each voxel of the lattice and the binding cavity. Darker squares indicate larger volumes of intersection; lighter squares have less intersection.

## 3.3 Methods

After we augment each protein, extract and vozelize their cavities, we split the datasets into a training sets and a test set. The training set is given to a variety of machine learning algorithms in order to assess the applicability of the algorithms. The machine learning algorithms that we evaluated were decision trees, naive bayes classifier, k-nearest neighbor classifier and support vector machine classifier. We decided to evaluate these methods based on previous research involving protein data and volumetric data. Some of the methods are simple, others are complex and while it might make sense to simply use the more complex methods, we decided to include the entire evaluation because the simple methods still gave us important insights into the structure of our datasets. Due to the asymmetry of the datasets and our data augmentation method, we had to design an elaborate cross-validation method to test the methods. The cross-validation method tries every single combination of splitting the dataset into a training set and a test set, i.e., it is exhaustive. In addition, it avoids overfitting on a single snapshot of a protein, generalizing over the function of the subfamilies and not over a specific protein.

Decision trees are a machine learning algorithm that learns from its input variables by recursively separating the input into subsets. Each split represents a node on a tree and the recursion ends when all subsets contains examples that belong to the same label. Those subsets are all leaf nodes. The model classifies an unknown examples variables and classifying the example by the leaf subset that it ends up in.

Naive bayes classifier are a machine learning model which assumes that the input features are statistically independent of each other and uses Bayes theorem to determine the classification of an example. The algorithm uses the training set to calculate the posterior probabilities for each class and uses those probabilities to predict other examples. The three varieties of naive bayes classifiers that we use in this paper are Gaussian, multinomial, and Bernoulli varieties which are differentiated by their assumptions about the input.

K-nearest neighbor classifier is a machine learning model which takes a training set and a parameter k, and then for each test example, finds the k nearest neighbors based on a given distance metric and predicts the label of the example by a majority vote.

Support vector machines are a machine learning model which constructs a hyperplane or a set of hyperplanes which maximize the distance between examples of different classes. The training set can be either linearly separable or non linearly separable.

### 3.3.1 Cross-validation

To avoid selection bias, we assessed the classification accuracy of each image recognition algorithm using exhaustive leave-k-out cross validation. Since there are three binding preference categories in both superfamilies, we describe our validation procedure on a given $k$ using $A$, $B$, and $C$ as abstract names for each subfamily. Vertical bars, e.g. $|A|$, refer to the number of proteins in a subfamily, and lower case letters, e.g. $a_i$ refer to all snapshots of an individual protein in a subfamily. First, we generate the set of all $\binom{|A|}{k}$ combinations of $|A| - k$ proteins in subfamily $A$, and call this set of sets $\bar{A_k}$. We also generate $\bar{B_k}$ and $\bar{C_k}$, which are the set of all $\binom{|B|}{k}$ combinations of $|B| - k$ proteins in $B$, and the set of all $\binom{|C|}{k}$ combinations of $|C| - k$ proteins in $C$. Then, for each iteration of our leave-k-out validation, we generate every combination of one member of $\bar{A_k}$, one member of $\bar{B_k}$, and one member of $\bar{C_k}$ to form our tripartite training sets. After training the classifiers, we test them on all snapshots of the $k$ proteins left out of each subfamily training set for that iteration of the experiment.

In cases where $k$ is greater than the size of the subfamily, we always keep one member of the subfamily in the training set. Without this requirement, training would be clearly be impossible. So, for example, in the case of leave-3-out validation, if the chymotrypsin subfamily is subfamily $C$, then $\bar{C_k}$ will contain only one subset, and the subset will contain snapshots for only one protein.

All snapshots for a given protein are always entirely included in a training set or entirely included in the test set, but never in both. Training on some snapshots and then testing on others from the same simulation would misleadingly bias recognition results to the positive, because a protein is always easier to recognize from other snapshots of the same protein. As a result, the scale of the validation experiment is goverened by the number of combinations of proteins, and it is unrelated to the number of snapshots.

### 3.3.2 Method evaluation and accuracy measurement

In all classification results that we compare, we measure accuracy as the number of correct predictions, where the snapshot of a protein is classified with the correct binding preference, divided by the total number of predictions made.

On the enolase superfamily, leave-k-out validation was performed with values of k ranging from 1 to 3. On the serine protease superfamily, leave-k-out validation was performed with values of k ranging from 1 to 8. The largest value of k in each case is limited by the size of the largest subfamily in each superfamily. Larger values of k would produce identical experiments because there would be no additional proteins to leave out.

All classifiers were exhaustively validated with data generated at 2, 1 and 0.5 angstroms. Examining a range of resolutions reveals to what degree classification accuracy is achieved by finer representations of the data, in exchange for additional computational time and storage.

All classifiers were exhaustively validated with data generated at 2.0Å, 1.0Å and 0.5Å resolutions. Examining a range of resolutions reveals to what degree classification accuracy is achieved by finer representations of the data, in exchange for additional computational time and storage.

## 3.4 Implementation details

The experiment was performed using Python scikit-learn package on a Ubuntu 15.04 system with a 8 CPU 2.8 Mhz Intel i7 processor and 16GB RAM. MD simulations were run on Lehighs Corona cluster with 6128 processors and 2 gigabytes of memory per core.

## 3.5 Results

We evaluated the accuracy of a decision tree classifier, a nearest neighbor classifier, a naive bayes classifier, and a support vector machine classifier for distinguishing conformational snapshots of proteins in different subfamilies of the enolase and the serine protease superfamilies. On the enolase superfamily, leave-k-out validation was performed with values of $k$ ranging from 1 to 3. On the serine protease superfamily, leave-k-out validation was performed with values of $k$ ranging from 1 to 8. The largest value of $k$ in each case is limited by the size of the largest subfamily in each superfamily. Larger values of $k$ would produce identical experiments because there would be no additional proteins to leave out.

### 3.5.1 Enolase superfamily

Decision tree classifiers trained on the enolase dataset exhibited a range of accuracies between .49 and .76 (Fig. 3.5). Accuracy did not appear to be closely related to resolution, but the decision tree trained with $k = 3$ was most accurate.

The nearest neighbor classifiers trained on the enolase dataset are shown in Figure 3.6. The accuracy of the classifiers varied between .5 and .67, and generally increased with finer resolutions. The classifier trained with $k = 2$ was most accurate.

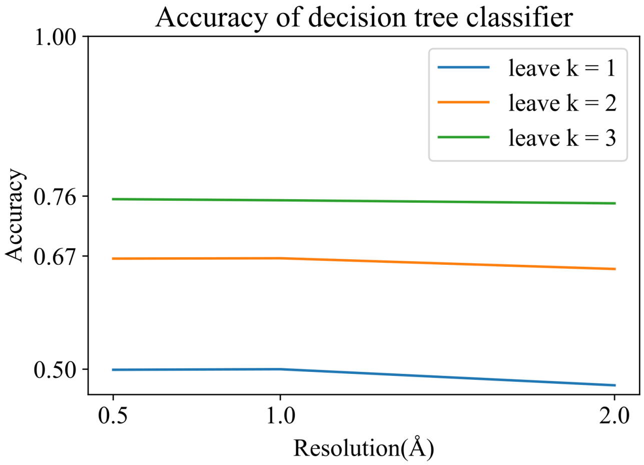


Figure 3.5: Accuracy of decision tree classifier on enolase dataset

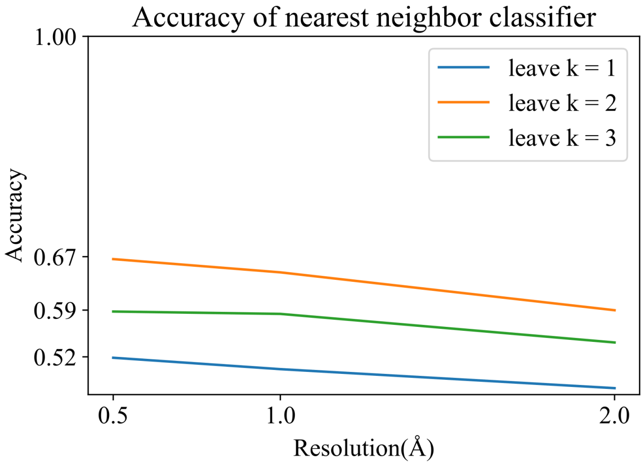


Figure 3.6: Accuracy of k-nearest neighbor classifier on enolase dataset

Naive bayes classifiers exhibited a range of accuracies between .5 to .75 (Fig. 3.7). Accuracy was almost unaffected by change in resolution. Classifiers trained with more proteins left out were slightly more accurate.

The support vector machines trained on the enolase dataset are shown in Figure 3.8. The accuracy of the SVM's varied between .1 and .71. SVMs of all tested basis functions performed well with $k = 2$ with accuracy degrading slightly with coarser resolutions, though the best performing SVM was trained with a linear basis function and $k = 2$. Among the best performing classifiers, those trained and tested on finer resolutions performed most accurately.

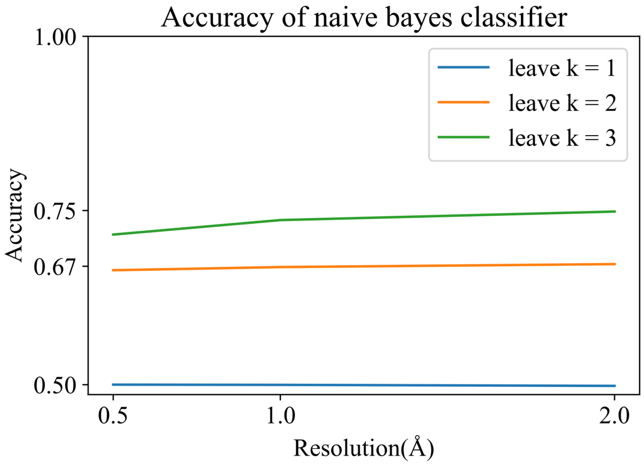


Figure 3.7: Accuracy of Naive Beayes classifier on enolase dataset

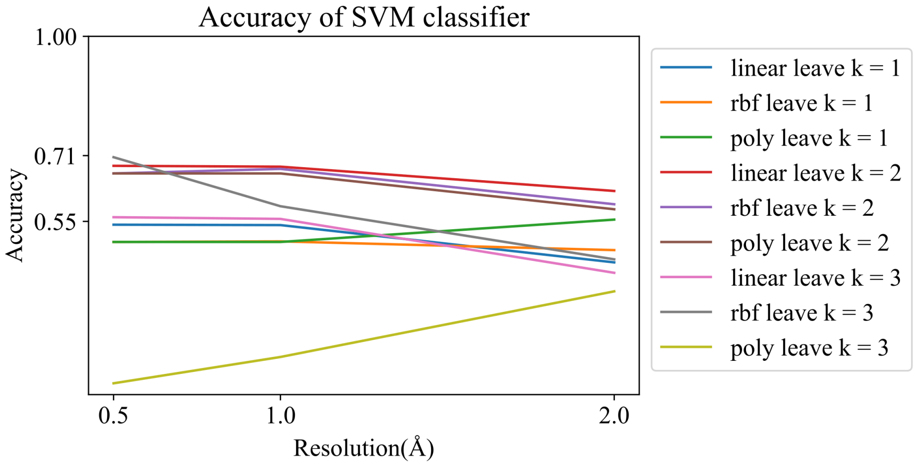


Figure 3.8: Accuracy of SVM classifier on enolase dataset

### 3.5.2 Serine Protease Superfamily

Decision tree classifiers exhibited a range of accuracies between .33 and 1.0 (Figure 3.9). Decision trees generally performed better at resolution 1.0 than at the other two resolutions, though performance was very mixed, and with values of $k = 5$. Classifiers trained at extreme values of $k$ performed less well.

Figure 3.10 illustrates the accuracies of nearest neighbor classifiers trained on the serine protease dataset. Nearest neighbor classifiers performed with accuracies ranging from .76 to 1.0, and they always performed better with finer resolutions. Classifiers trained with values of $k = 7$ and in the upper middle range of the values tested performed best, while lower values of $k$ performed

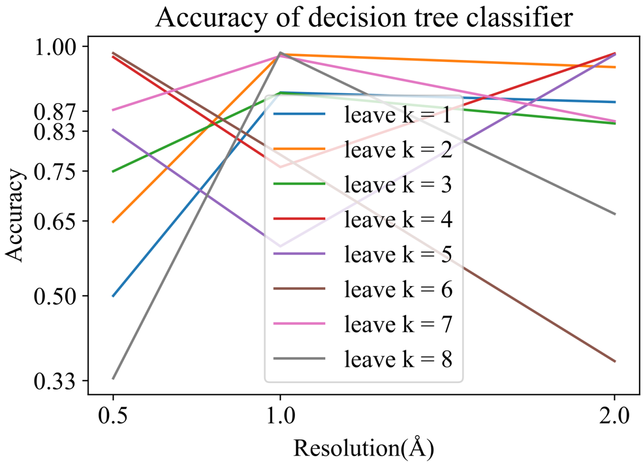


Figure 3.9: Accuracy of decision tree classifier on serine protease dataset

worst.

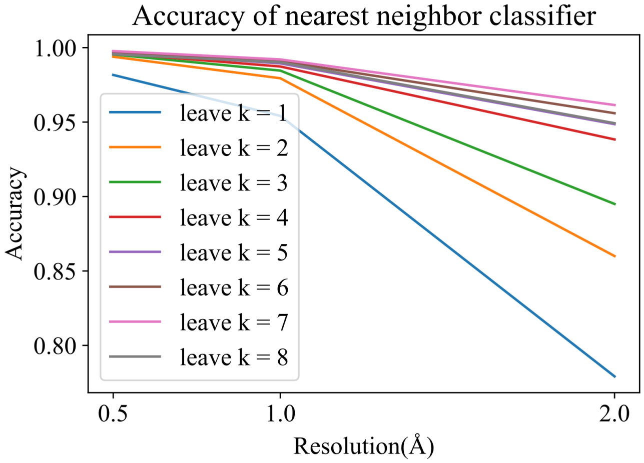


Figure 3.10: Accuracy of k-nearest neighbor classifier on serine protease dataset

Naive bayes classifiers trained on the serine proteases (Fig. 3.10) exhibited a range of accuracies between .80 and .99. Generally, naive bayes classifiers were less accurate with coarser resolutions, and performed best with $k$ values at the upper middle range of the values tested. The classifier trained by leaving out seven proteins performed best, followed closely by the one trained by leaving out six.

Figure 3.12 plots the classifier performance of support vector machines trained and tested on the serine protease dataset. These classifiers exhibited accuracies between .95 and 1.0. While the lowest performing SVMs tended to improve with coarser resolutions, the best performing SVMs

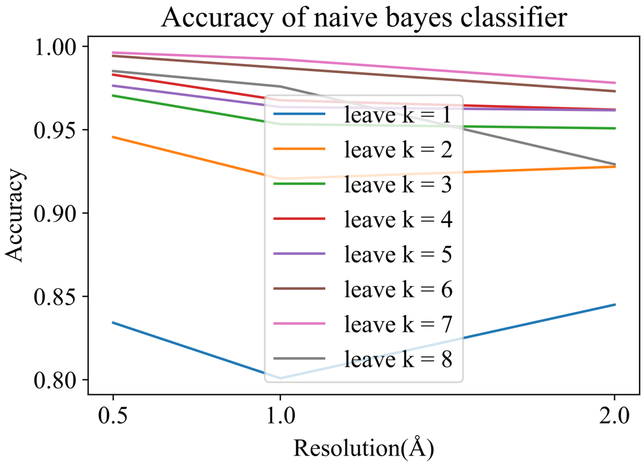


Figure 3.11: Accuracy of Naive Bayes classifier on serine protease dataset

were uniformly accurate at all resolutions. These best performing SVMs applied a radial basis function and were generally trained with values of $k$ at or near 7.

Confusion matrices detail the predictions made by SVMs on enolases and serine proteases (Figure 3.13). Here, we can see that predictions were largely accurate for all subfamilies of the serine proteases, and also for the enolase subfamily, but not for the mandelate racemaces. Given that the mandelate racemases had only two members, all exhaustive validations would have reduced it's training set to one protein, which may have been insufficient for accurate training.

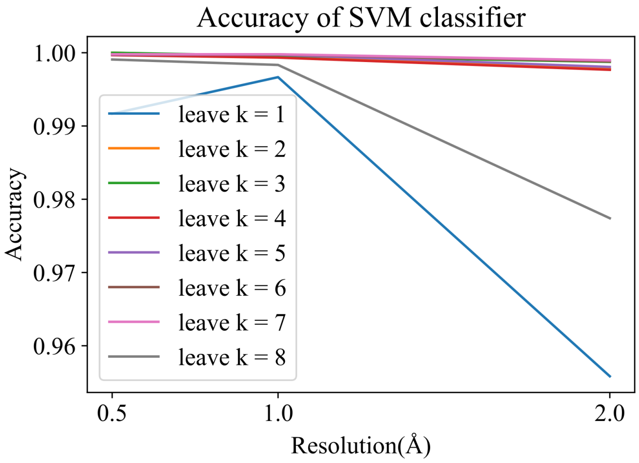
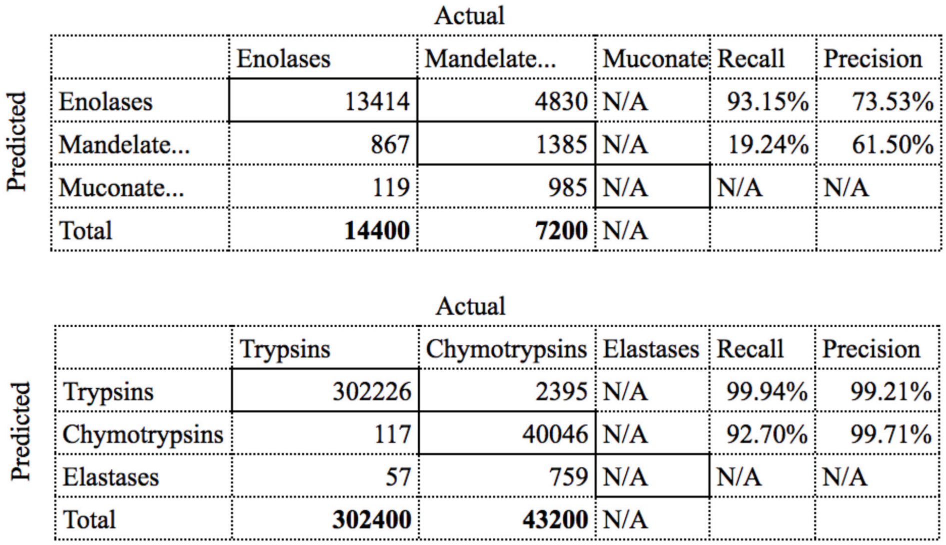


Figure 3.12: Accuracy of SVMs classifier on serine protease dataset



Actual

| Predicted | | Enolases | Mandelate... | Muconate | Recall | Precision |
|---|---|---|---|---|---|---|
| | Enolases | 13414 | 4830 | N/A | 93.15% | 73.53% |
| | Mandelate... | 867 | 1385 | N/A | 19.24% | 61.50% |
| | Muconate... | 119 | 985 | N/A | N/A | N/A |
| | Total | **14400** | **7200** | N/A | | |

Actual

| Predicted | | Trypsins | Chymotrypsins | Elastases | Recall | Precision |
|---|---|---|---|---|---|---|
| | Trypsins | 302226 | 2395 | N/A | 99.94% | 99.21% |
| | Chymotrypsins | 117 | 40046 | N/A | 92.70% | 99.71% |
| | Elastases | 57 | 759 | N/A | N/A | N/A |
| | Total | **302400** | **43200** | N/A | | |

Figure 3.13: SVM performance confusion matrix.

# 4 Conclusions

## 4.1 Exact solid representations of polyhedral surfaces

Section II introduced an algorithm for generating arbitrarily precise molecular surfaces, and defined operations which enable working with and analyzing the generated surfaces. This thesis benchmarked the algorithm implementation by performing an extensive analysis on the performance when generating artificial primitives, real molecular surfaces, operations on real surfaces, and lastly, the algorithm tested the importance of the additional accuracy on a statistical application.

The results of the benchmark point out that the implementation is efficient across a variety of operations ranging from simple to complex. The produced representations are nearly identical to the representations produced by other software. Finally, the additional accuracy is proven to be crucial to achieving 0 false negative predictions in a statistical application measuring the binding similarity of proteins.

The arbitrarily precise algorithm generating molecular surfaces is an important cornerstone to developing experiments and applications using protein structures. The algorithm enables scientists to quickly generate a representation and use the operations defined on the molecular surfaces

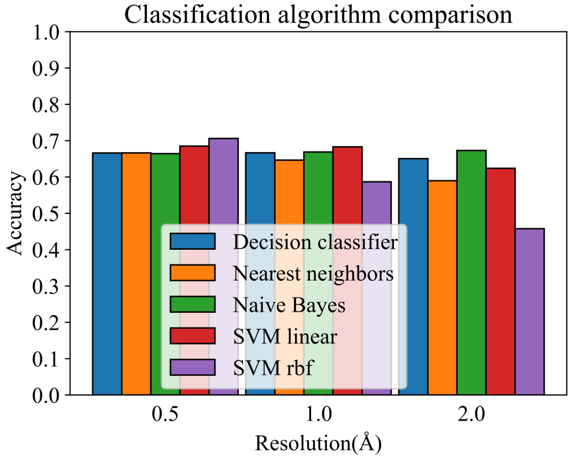


Figure 3.14: Comparison of classifier performance on enolase proteins at several resolutions.

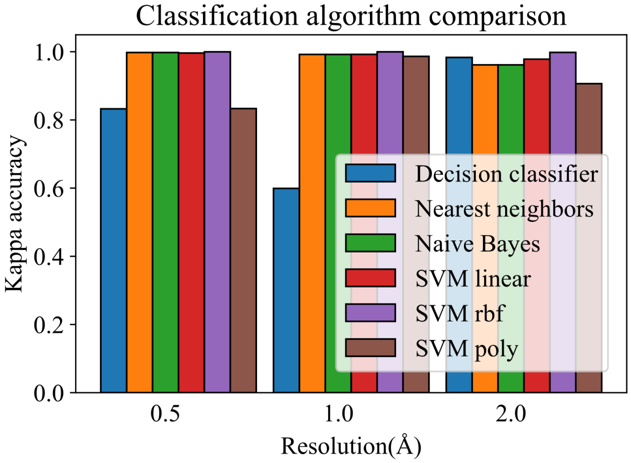


Figure 3.15: Comparison of classifier performance on serine proteases at several resolutions.

to extract and compare different parts of the protein. In addition, the precision of the algorithm is preserved after any amount of operations due to the geometric primitive representations used to model atoms and inter-atom space. The additional precision provided by the geometric primitives plays an important role in making sure the surfaces are accurate and the low-level detail - significant. The interoperability with other software adds additional flexibility to the algorithm.

These capabilities enable us to gain important insights into what alters protein binding behaviour. These insights will be invaluable to anyone studying how drug therapies can be evaded, how molecular interactions change, and how protein therapies can be designed for improved specificity.

## 4.2 Classification of ligand binding cavities

Section III explained the design, simulation and generation of two datasets of molecular cavities and performed an extensive machine learning model evaluation with the ultimate goal of predicting protein binding affinity. The evaluation included decision tree classifiers, naive bayes classifiers, k-nearest neighbor classifiers and support vector machine classifiers. The evaluation tested whether machine learning algorithm can be successfully applied to cavity classification and that in particular.

The evaluation showed that the SVM classifier with linear kernel performs as well or subtly better than SVM classifiers with other kernels or other classifiers. Performance across all classifiers degraded as precision decreased which means that the higher precision is crucial to the power of the classifiers. These results also mean that molecular structures represented with sufficient precision and then converted to a volumetric representation can be successfully classified by machine learning algorithms.

The practical implications of being able to classify a protein cavity into a category is that a category defines a certain behaviour shared by all proteins in the category. As long as there are some experimental examples of a group of proteins exhibiting a binding specificity, a scientist will be able to define that as a category. A class of proteins can then be data augmented using dynamics simulation to build a sufficiently large dataset. A classifier trained on that dataset can immediately predict what a cavity will bind to, accelerating the slow and arduous process of drug design by orders of magnitude.

# References

[1] Patricia C Babbitt, Miriam S Hasson, Joseph E Wedekind, David RJ Palmer, William C Barrett, George H Reed, Ivan Rayment, Dagmar Ringe, George L Kenyon, and John A Gerlt. The enolase superfamily: a general strategy for enzyme-catalyzed abstraction of the $\alpha$-protons of carboxylic acids. *Biochemistry*, 35(51):16489–16501, 1996.

[2] Gunnar I Berglund, Arne O Smalas, Heidi Outzen, and Nils P Willassen. Purification and characterization of pancreatic elastase from north atlantic salmon (salmo salar). *Molecular marine biology and biotechnology*, 7(2):105–114, 1998.

[3] Javier Bernal. Regtet: A program for computing regular tetrahedralizations. In *International Conference on Computational Science*, pages 629–632. Springer, 2001.

[4] Brian Y Chen. Vasp-e: Specificity annotation with a volumetric analysis of electrostatic isopotentials. *PLoS computational biology*, 10(8):e1003792, 2014.

[5] Brian Y Chen and Soutir Bandyopadhyay. Vasp-s: A volumetric analysis and statistical model for predicting steric influences on protein-ligand binding specificity. In *Bioinformatics and Biomedicine (BIBM), 2011 IEEE International Conference on*, pages 22–29. IEEE, 2011.

[6] Brian Y Chen and Barry Honig. Vasp: a volumetric analysis of surface properties yields insights into protein-ligand binding specificity. *PLoS computational biology*, 6(8):e1000881, 2010.

[7] Markus Fischer, Qiangfeng Cliff Zhang, Fabian Dey, Brian Y Chen, Barry Honig, and Donald Petrey. Markus: a server to navigate sequence–structure–function space. *Nucleic acids research*, 39(suppl_2):W357–W361, 2011.

[8] László Gráf, A Jancso, László Szilágyi, György Hegyi, Katalin Pintér, Gábor Náray-Szabó, József Hepp, Kálmán Medzihradszky, and William J Rutter. Electrostatic complementarity within the substrate-binding pocket of trypsin. *Proceedings of the National Academy of Sciences*, 85(14):4961–4965, 1988.

[9] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3):435–447, 2008.

[10] Karin Kühnel and Ben F Luisi. Crystal structure of the escherichia coli rna degradosome component enolase. *Journal of molecular biology*, 313(3):583–592, 2001.

[11] Kazuyuki Morihara and Hiroshige Tsuzuki. Comparison of the specificities of various serine proteinases from microorganisms. *Archives of biochemistry and biophysics*, 129(2):620–634, 1969.

[12] Donald Petrey and Barry Honig. Grasp2: visualization, surface properties, and electrostatics of macromolecular structures and sequences. In *Methods in enzymology*, volume 374, pages 492–509. Elsevier, 2003.

[13] John F Rakus, Alexander A Fedorov, Elena V Fedorov, Margaret E Glasner, Brian K Hubbard, Joseph D Delli, Patricia C Babbitt, Steven C Almo, and John A Gerlt. Evolution of enzymatic activities in the enolase superfamily: L-rhamnonate dehydratase. *Biochemistry*, 47(38):9944–9954, 2008.

[14] Susan L Schafer, William C Barrett, Abraham T Kallarakal, Bharati Mitra, John W Kozarich, John A Gerlt, James G Clifton, Gregory A Petsko, and George L Kenyon. Mechanism of the reaction catalyzed by mandelate racemase: structure and mechanistic properties of the d270n mutant. *Biochemistry*, 35(18):5662–5669, 1996.

## Biography

Georgi Georgiev was born to Dimitar and Zlatinka Dimitrovi in Varna, Bulgaria in May of 1994. He graduated from Mathematical High School Petar Beron and attended Lehigh University for his bachelors and masters degree in May 2017 and August 2018, respectively.