Machine Learning Applications to Protein Variant Effect Prediction

by

Jeff Soules

a thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
Computer Science
New York University

May, 2019

Abstract

Proteins are microscopic machines whose activity forms the basis of all life processes. If a mutation causes variation in the typical amino acid sequence of a protein, the protein's normal biological function may be compromised. Variant Interpretation and Prediction Using Rosetta (VIPUR) uses sequence and structural data features to predict whether a mutation is deleterious to the protein's function.

VIPUR was originally released with a curated set of protein variants as its training data. As released, it achieved 80% accuracy on this data set. However, the original design was tightly coupled to a logistic regression classifier, so other machine learning techniques could not be easily tested.

The reimplementation of VIPUR presented in this work offers a modular design that can be extended with classifiers built on any machine learning approach. It establishes a methodologically sound basis for experimentation with new classifiers, data features, and data sets.

This work examines the predictive power of the data features in the original VIPUR training set, and establishes a high baseline for classification performance based on one strongly predictive feature category.

The present work includes classifier modules built with four different machine learning approaches—logistic regression, support vector machines, gradient-boosted forests, and neural networks. These represent the two model types considered in the original VIPUR work, and two more recent classifier types. The modules are trained with automated hyperparameter cross-validation and rigorously evaluated with k-fold cross validation, establishing a baseline of performance for future experiments.

Results show very slight improvement over the original logistic regression method, consistent with the dominance of a small handful of features in determining classification results. Potential new data features and sources are discussed, which can be used in the new VIPUR design without modification while maintaining backwards compatibility with previously trained classifiers.

Contents

Listing of figures

Acknowledgments

I COULD NOT HAVE ARRIVED at writing this thesis without the guidance and support of many, many people. Thank you first to Prof. Rich Bonneau and Dr. Julia Koehler Leman, for taking me on and turning me loose on such an exciting project, and also to Dr. Vladimir Gligorijevic for very helpful methodological guidance. Thanks also to Profs. Iddo Drori and Ernie Davis, for giving me the machine learning and mathematical tools that are the foundation of this project. In my non-student life I owe gratitude to Darin Phelps and Torie Atkinson, who have, at work and at home, tolerated my absences to do this work.

1

Introduction

Proteins are the foundation of nearly every life process in living things.[34] A protein is a biologically functional macromolecule[34] whose three-dimensional shape and electrochemical properties determine its interactions with other or-

ganic and inorganic compounds, and ultimately its biological function. When mutations arise that cause one or more constituent amino acids to be replaced with others, the protein's function may be affected.

VIPUR—Variant Interpretation and Prediction Using Rosetta—is a machine learning tool designed to predict whether a mutation will result in compromise of the protein's function.[6] VIPUR showed noticeable improvement over alternative methods at the time of its release in 2015.[5] However, computational biology and machine learning are rapidly changing fields. Ongoing research is needed to ensure VIPUR's performance remains state-of-the-art, and to extend its scope to broader classes of proteins.

The present work has three principal goals: first, to develop an improved VIPUR tool which offers a modular platform for ongoing experimentation and development; second, to provide methodologically sound benchmarks for the methods considered in the original VIPUR work; and finally, as proof-of-concept, to conduct experiments using current state-of-the-art machine learning methods to attempt to improve upon VIPUR's performance with its original data set.

1.1 Background

Proteins are biochemical structures consisting of long chains of 20 basic amino acid types in combination. Chemical and electrostatic interactions along the amino acid chain and with the immediate cellular environment cause the chain to fold into complex shapes and adopt the protein's characteristic three-dimensional conformation. Mutations in the protein can cause changes in the amino acid se-

quence, potentially leading to conformational changes that disrupt the protein's typical function.

One major category of mutation is *missense mutations*, in which one amino acid is substituted for another. These are distinct from *silent* mutations, in which a change in DNA sequence results in the same amino acid sequence (and thus no change to the realized protein), and *nonsense* mutations, which cause premature termination of protein construction (and usually a nonfunctional protein).[34] VIPUR presently considers only missense mutations.

Because of their importance to biological functions, proteins are a principal subject of study for biologists. As a result, extensive collections of amino acid sequences are available. There also exist somewhat less extensive sources for the resulting three-dimensional structures, although many of these are homologous models or otherwise not empirically verified. Determining form from sequence alone is an ongoing problem. Tools such as Rosetta,[28] an extremely powerful three-dimensional protein modeling suite, can nonetheless provide insight into the quality of three-dimensional models, whether verified from the lab, predicted from homologous structures, or created *de novo*.

Machine learning—or, somewhat less mysteriously, applied computational statistics—is the field of computer science which studies algorithms whose behavior depends more upon training data than on explicit instructions from a programmer.[37]

Machine learning can be broadly divided into the fields of *supervised learning*, in which there exists a *training set* of data with known-correct labels;

reinforcement learning, in which data is associated with some distant information signal (reward or penalty) that falls short of an explicit label; and *unsupervised learning*, which attempts to detect patterns in data without attaching preconceived meanings to those patterns.[35] Regardless of field, data are customarily modeled as *feature vectors*, or collections of observed values, which characterize each observation sample: such feature vectors can then be considered points in a k-dimensional *feature space*.[9]

In the supervised learning context, a scalar (real-valued or categorical) label value is attached to each of these points. The task of the machine learning algorithm is then to discover a function which maps the feature vectors $\vec{x} \in \mathbb{R}^k$ to the labels y, formalized as $f(\vec{x}) \to y$.[37] It is generally assumed that $\vec{x}$ and y are drawn from some joint probability distribution, so that this relationship is consistent.

Since the space of "all functions" is both infinite and very large, most approaches make assumptions about the class of functions f (or *hypotheses*[29]) which may be considered. For instance, the familiar linear regression model considers functions of the form $f(\vec{x}) = w_1x_1 + w_2x_2 + \ldots + w_kx_k = y$: it is assumed that f is a linear function of the features. This function space is still infinite, since any values could be chosen for the coefficients w; but for many data sets there exist principled and computable ways to determine optimal coefficients within the class of functions.

The linear regression example above introduces a new category of unknowns $\vec{w}$, known as *parameters* (or in some contexts *weights*). If the family of

functions describing the relationship between the input data $\vec{x}$ and the output label y is fixed, the only variable left is the values of the parameters. Finding the optimal values for these parameters, in order to maximize correctness over both the known training set and the unknown data encountered in the wider world, becomes the crux of most machine learning problems.

"Optimal" implies a standard of judgment. The quality of a solution is judged based on its *error*, represented as the sum of the values of a *loss function* over a data set. The loss function or error function is a formula that quantifies how wrong an answer is, by measuring the discrepancy between the value $\hat{y}$ produced by the model function and the correct value y.[9]

Error is in turn divided into two kinds:[29] *empirical error*, which is the error on some known set of data, and *generalization error*, which is the difference between the empirical error and the expected error on any arbitrary data set drawn from the same joint distribution. Discrepancies between the two may arise from statistical biases—the training set may not be truly representative of the underlying distribution—or from *noise*: inaccuracy in measurements and potential mislabeling of data. In the worst case, it may be that there is no strong correlation between the features and the labels.

It is desirable to have a model which will work well for future data, not just the data already seen, or for which answers are already known. Models which show a strong discrepancy between empirical error and (estimated) generalization error are said to suffer from *overfitting*: they fit the known data well, but perform poorly in practice, because their results depend upon relationships

in the training set that are not consistently present in real-world data.

1.2 Prior Work

VIPUR is not unique in attempting to predict the effects of mutations. However, at the time of its release VIPUR was unusual in both attempting to generalize to non-human proteins, and in relying on a combination of sequence and structural data.[5] Most competing approaches centered on sequence-only features, particularly variants of the PSSM.[5] Of these, Provean[13] and PolyPhen2[1] appear to have been the strongest competing approaches at the time of VIPUR's release. VIPUR outperformed both these tools on both Area Under Precision-Response Curve (AUPR) and (substantially) on Receiver Operating Characteristic (ROC) measures. Additionally, PolyPhen2 was limited to human proteins as of the original VIPUR work.[6]

1.2.1 Challenges in Comparing Classifiers

This comparison raises the first significant challenge in comparing different predictors of mutation deleteriousness: each predictor is trained on its own data set which is curated with its own standards for what mutations are given a 'deleterious' label. Some tools define deleterious to mean pathogenic, or causing disease in the organism. Others measure changes to an organism's phenotype—which can include disease states like diabetes, but also incidental features, like variations in hair color. For the purposes of the VIPUR Training Set (and classifiers trained with it), the 'deleterious' label focuses on proteins themselves:

protein records are labeled deleterious if they show disrupted function, as measured in stability, changes to active site or protein-protein interfaces, or folding.[6]

Given the challenges in comparing across classifiers and across data sets, the present work focuses on performance measured against the original VIPUR tool and training set.

1.2.2 Performance of Original VIPUR

The original VIPUR work claimed generalization accuracy of 81%, rising to 94% accuracy for very-high-confidence predictions.[5] In addition to accuracy, VIPUR was evaluated on AUPR and ROC measures. The score of the fully trained VIPUR logistic regression classifier was found to be 0.872 AUPR and 0.831 ROC. Equivalent measurements for an optimally trained radial-basis-function SVM were found to be 0.835 AUPR and 0.784 ROC.[5] The original work used chi-squared tests to claim that VIPUR scores did not show bias due to data source, model source, domains, species of origin, structural context, molecular function or biological process (as measured by Gene Ontology labels), or amino acid transition.

The original VIPUR work attempted to estimate generalization performance using random resampling, averaging 100 randomly-chosen 80% train/20% test splits. The present study uses a k-fold cross-validation approach instead, since it is more systematic, consistent, and representative. Either approach presents a challenge in splitting the data set without causing information leak-

age from the test set into the training set. These challenges will be discussed further in chapter 3.

1.2.3 Components of VIPUR

The original VIPUR consisted of four main components: (1) a well-curated, broadly sourced training data set, the VIPUR Training Set (VTS); (2) a feature extraction pipeline which uses Rosetta[28] and PROBE[41] for structural features and PSIBLAST[11] for sequence features; (3) a feature selection and training pipeline using those features; and (4) a three-part classifier set, consisting of one classifier trained on the complete feature set, one trained on the structural features alone, and one trained on the sequence features alone.

1.2.4 VIPUR Feature Extraction

The VIPUR feature extraction pipeline generates a set of 106 features for each protein variant input.[5] Of the features, 5 are derived from the amino acid sequence. The other 101 are nearly all score function components from Rosetta, a general-purpose protein structure modeling tool.[28]

The sequence features include 4 PSSM-based measurements derived from PSIBLAST, and a summary amino acid dissimilarity feature new to VIPUR, `aminochange`. The `aminochange` term categorizes each of the 20 amino acids which occur in natural proteins into one of seven groupings. It assigns a 1 to the feature if the amino acids appearing at point of variation belong to the same group in both wild-type and variant proteins, and a 2 otherwise.

The *Position-Specific Scoring Matrix* (PSSM) is a standard measure of sequence likelihood. It is computed from a broad sample of the amino acid identities in a particular windowed context, and records the log-likelihood of a particular amino acid type appearing in this context. The PSSM features in VIPUR compare the log-likelihood of the acid at the varying position, in both the wild-type sequence (`pssm_native`) and in the variant being examined (`pssm_variant`), as well as `pssm_differrence`, the difference between the two, and `pssm_information_content`, the information content of the position.

The remaining 101 features are derived from the observed or predicted three-dimensional protein structure. 17 come from Rosetta `ddg_monomer`, 83 from Rosetta FastRelax, and 1 (solvent-accessible surface area (ACCP)) is generated using PROBE. The Rosetta-based features attempt to capture the relative stability between the native/wild-type protein structure and the variant's structure in a set of scalar measurements.[26]

To generate these scoring features, the protocols begin with a curated structure for the wild-type protein. This structure is relaxed 50 times, and the scores of the resulting models are recorded. The protocol then substitutes the variant residue into the curated structure, and again runs 50 steps of structural refinement. The result is a range of score term values for both the native and wild-type protein structure. The Q1, Q2, and Q3 values of these distributions are compared between wild-type and variant, resulting in three quartile values for each score term. Additional terms identify proteins for which Rosetta expects the variant will cause a dramatic change in overall conformation.[5]

All features in VIPUR are normalized to zero mean and unit standard deviation before classifier training.

These salient features of the VIPUR tool have been preserved from its first release into the present version. The internals of the tool, however, have been completely rewritten. The next chapter provides an overview of these changes.

2

Software Engineering Challenges

One major goal of the present project is to develop VIPUR as a software tool that is portable and easily extensible. VIPUR must support a wide array of ongoing experiments on a methodologically sound basis. To achieve this, I present a complete rewrite of the VIPUR training and classification engine,

along with pluggable modules for several classifiers, a rebuilt cross-validation module for training and evaluation, and extensive data collection of results to enable subsequent data analysis and graphing. Whenever possible, the classifier modules presented here rely on standard libraries, which provide commonly accepted reference implementations of the machine learning methods used. The new VIPUR is extensively configurable through command-line options, so behavior is no longer determined by hard-coded values. Finally, the new VIPUR presented here is written in Python 3, since Python 2 will cease to be supported at the end of 2019.

Collectively, these features simplify experimentation with different settings (particularly in in cluster environments), allow backward and forward compatibility as new data features are added, and ease adoption of VIPUR by a broad range of new users.

2.1 Providing Structured Data Representations

The present work provides structured representations of VIPUR protein data. Each VIPUR protein variant record consists of 106 real-valued numeric features, identified by UniProt ID, PDB ID, and Variant ID, as well as a "deleterious/neutral" class label (for training data).

In the original VIPUR, a set of records was represented as a fixed matrix of values. Features could only be identified by column order, and code in distant parts of the application would refer to record identifier values or the class label according to column position. This was both difficult to read, and brittle: re-

ordering columns in the data file would cause a saved classifier to begin reporting incorrect predictions. Further, representing features by column order alone creates challenges in adding new features, which is an area of active research in the VIPUR project.

The present VIPUR uses an object-oriented design wherever possible. Individual data samples are now represented by a `VipurFeatureVector` object, which stores identifying information in dedicated fields, and stores feature values in a key-value dictionary keyed by feature name. Storing features by name instead of order offers many benefits: raw and normalized values of each feature can be recorded and manipulated together; classifiers can explicitly check for feature compatibility with input data (and raise a meaningful error in the event of a mismatch); on-disk representations of data can be reformatted more safely; and most importantly, new features can be added without invalidating existing classifiers.

Taken altogether, these changes will facilitate adoption by a wider user base and easier expansion of VIPUR.

Along with the named feature system, I have also introduced a new class and file, the `VipurFeatureDictionary`. This is a central store for the list of features known to a particular version of VIPUR, along with their descriptions, and classification into the "Sequence" or "Structure" categories. This information is used to explain VIPUR features and provide compatibility versioning.

Many hard-coded values have been replaced with enumerations. For instance, the "Deleterious" and "Neutral" label classes are now represented as

`VipurLabel.DELETERIOUS` and `VipurLabel.NEUTRAL`. This allows developers to use human-readable labels for hard-coded data values, which reduces programmer error.

2.2 Applying Modularization to Classifiers and Data Formats

To allow diverse classifiers to be used interchangeably, the present version of VIPUR establishes a *loosely coupled internal interface*. As illustrated in figure 2.1, all classifier modules inherit from the `VipurBinaryClassifier` abstract base class, which provides a consistent interface for training and evaluation functions. The structure of this class is partially inspired by the interface of `Scikit-Learn`'s classifiers,[32] with additional functionality to handle the VIPUR three-part classifier model (in which separate classifiers are trained on structure-only, sequence-only, and combined feature sets) and to provide integrated cross-validation of parameters.

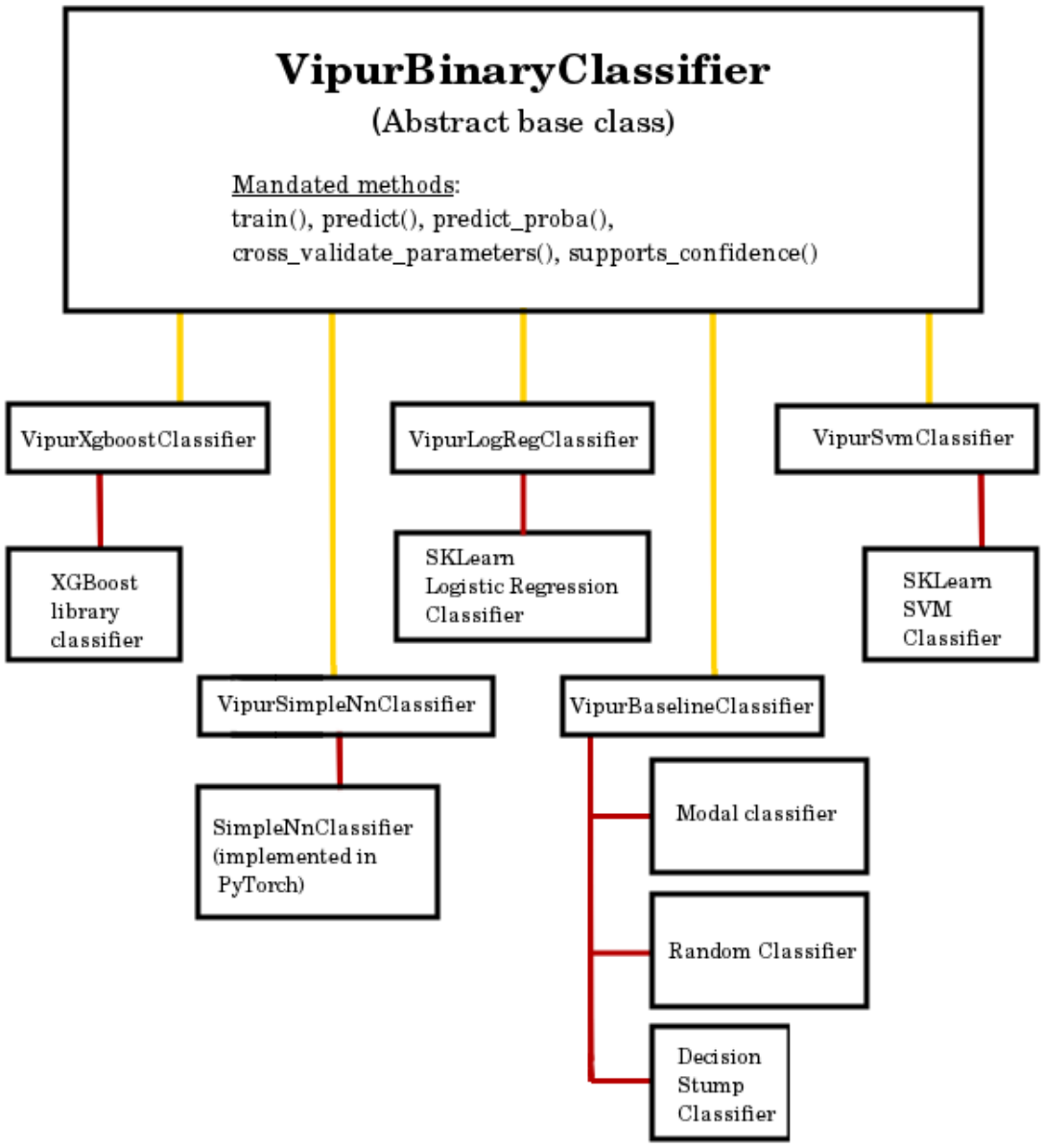

Figure 2.1: An Entity-Relationship Model diagram illustrating the structure of the modular classifiers presented in this work. Yellow lines indicate inheritance, while red lines indicate composition.
All VIPUR classifier modules must inherit from a base class which enforces a standard interface used by the training components. Additional methods may be incorporated as needed by the module. Individual classifier modules have instances of classifiers from reference implementations, or classifiers implemented locally when a pure library implementation is not available.

Any classifier extending `VipurBinaryClassifier` is expected to build an appropriate set of classifiers in its constructor, and to support `train`, `predict`, `predict_proba`, `cross_validate_parameters`, and `supports_confidence` methods. The `predict` method returns a binary prediction of type `VipurLabel`, while `predict_proba` follows `SciKit-Learn` in returning an array of probabilities P, where P_k is the classifier's estimate of the probability that the sample belongs to class k. `supports_confidence` returns a true/false value indicating whether the classifier is capable of providing probability estimates (and thus AUPR/ROC metrics). Finally, the `cross_validate_parameters` method allows each classifier, when supplied with an appropriately arranged data set and a configured `VipurCrossValidator` object, to find and set optimal values for the classifier's hyperparameters.

2.3 Making Use of Standardized Machine Learning Implementations

Prior to the current version, VIPUR had a hard-coded logistic regression classifier. While the classifier was trained with the `SciKit-Learn` library, it was persisted to disk only as a set of feature weights in feature file order, and the classification task was carried out with custom-written code, incurring a maintenance cost and creating the possibility of coding errors.

The current implementation avoids reimplementing classifier logic whenever a commonly accepted reference implementation exists. To that end, the current logistic regression classifier now uses the `SciKit-Learn` implementation throughout (not just in training). The SVM classifier also uses the `SciKit-Learn`

reference implementation. The `XGBoost` library provides the gradient boosting classifier, and the neural network models are implemented in `PyTorch`. For the latter two classifier types, a minimal amount of additional interface code allows the use of standard `SciKit-Learn` training and cross-validation functions. Using reference implementations ensures that these algorithms are implemented efficiently and correctly, and reduces the possibility of misleading experimental results due to implementation error.

Finally, each classifier is now aware of the features it was trained upon. When a persisted, pre-trained classifier set is hydrated from disk for use on new data, it requests the features it needs from the `VipurFeatureVector`s it evaluates. If they are not available, the classifier can proactively raise an error. This ensures that stored classifiers are either compatible, or give useful error messages. Disk persistence is achieved through the `cPickle` serialization module (standard in Python 3) which allows complex models to be written to disk and moved between computing environments without extensive module-specific data representation design or code.

2.4 Support for Multi-Learner Approaches

In addition to the modularization improvements described above, the current VIPUR redesign also includes support for capturing a classifier's predictions (both classification and probability) on the input data. These classifier predictions can then be recorded as new features which supplement the original data set. Adding these supplemental features to the data set allows easy use of

stacked multi-learner approaches, beyond individual classification modules. In a stacking approach, the predictions of individual classifiers form a "committee of experts" whose opinions can guide the training of a new meta-learner, which can both make its own predictions, and learn the conditions under which individual experts perform well.

The implementation avoids information leakage by recording predictions on each data point while it is part of the test fold in a cross-validation scheme. (At no point do the new features record the predictions of a model that was trained on the specific samples it is predicting.) The resulting predictions are stored and output as new feature columns in a separate data file, which can be reused within VIPUR. The new data fields are also flagged so that they are not normalized when VIPUR imports the new data file.

The present work considers ensemble methods in the form of gradient-boosted forests, but a stacking approach using heterogeneous learners (as described here) remains to be explored in future work.

Thanks to the improvements detailed above, the new VIPUR will smoothly handle new data sets and new features when they become available. The experiments related in the present work, however, are based on the existing Vipur Training Set. The next chapter analyzes this data set in detail.

3

Analysis of the VIPUR Training Set

Creating a new set of training data for VIPUR is an ongoing project outside the scope of the present work. Instead, the present work has focused on building a tool which is flexible enough to evaluate classifiers trained with new data and new features once they are available. I have applied the resulting

tool to benchmark the classifier types studied in the original VIPUR,[6] as well as several new methods which were not considered. The experimental results presented in this paper rely entirely upon the *VIPUR Training Set*, or VTS, introduced by Baugh.[5] This chapter will explore the benefits, as well as some of the limitations, of this data set.

3.1 Origins of the VIPUR Training Set

One of the major achievements of the original VIPUR work was the creation of a well-curated data set for the deleteriousness-prediction problem. As discussed in Baugh 2017,[5] existing data sets, and the methods they support, suffer from strong bias effects due to species specificity, dominance of deleterious or neutral variants in the sample space, and even over-representation of a narrow set of individual proteins. Baugh hypothesized that loss of function should be generalizable regardless of mutation source or target species, and created a data set that drew upon diverse species and sources of variation. This data set, the Vipur Training Set (VTS), includes protein sequences and annotations sourced from the UniProt database,[39] supplemented with structures from the Protein Data Bank,[8] ModBase,[33] and SwissModel.[40] The VTS looks specifically at *missense point mutations*, mutations which cause the replacement of one amino acid with another. It uses gold-standard function loss labels drawn from expert annotation.

The resulting VTS is composed of roughly 1/4 variants drawn from HumDiv[1] and 3/4 variants drawn from UniProt, making up a data set of 9,477 vari-

ants of 2,637 domains in 2,444 soluble proteins. As Rosetta's scoring function for soluble proteins was more advanced than that for membrane proteins at the time of the VTS' compilation, the data set excludes membrane proteins. The VTS is comprised of 5,901 human variants, 1,635 non-human eukaryote variants, 1,725 prokaryote variants, 122 variants from archaeobacteria, and 94 viral protein variants.

3.2 Non-Uniformity of the VIPUR Training Set

Traditional theory of supervised machine learning assumes the training data are representative of the data which will be tested.[29] This can take the form of a strong IID assumption, asserting that all data are drawn independently and identically distributed from the same underlying distribution; or it may be a less rigorous assumption, with correspondingly less generalizable stated results.

The VTS is heterogeneous by design, drawing from many sources to assemble a data set covering a wide range of organisms. The objective is to represent the full range of proteins found in nature, rather than any specific class of organism or area of study, so that VIPUR can learn features which broadly characterize proteins in general, rather than being focused on any subset. While diversity was one of the goals of the VTS, its internal variation has not been studied in detail. If there are significant internal divisions within the data set, and if they are representative of divisions within set of viable proteins as a whole, this information could guide the development of more tailored classifier models.

Each variant in the VTS is described by a set of 106 features. This creates a feature space of sufficiently high dimension to rule out a straightforward unsupervised search for clusters within the data set. However, the data can be reduced to sufficiently low dimension to visualize, through methods such as Principal Component Analysis.[25]

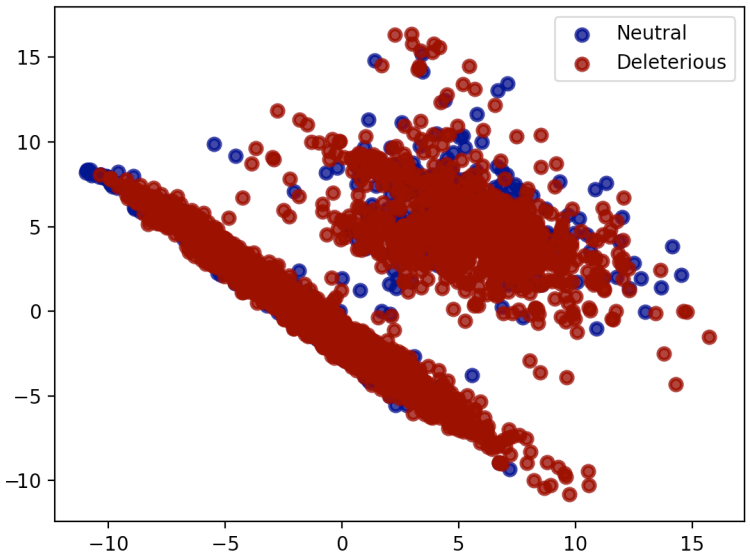

Figure 3.1: VIPUR Training Set reduced to two dimensions under Principal Component Analysis. Principal components explain 0.099 and 0.088 of the variation.

Figure 3.1 shows the VTS reduced through PCA to two dimensions, with colors showing the deleterious/neutral classification of the samples. The plot suggests that there are at least two pronounced clusters in the VTS, and that deleterious and neutral mutations are thoroughly mixed within both clusters.

Baugh[5] found that many of the features in the VTS were correlated, and hypothesized that dimension reduction through feature selection might improve classifier performance. Applying the same PCA analysis to the reduced feature

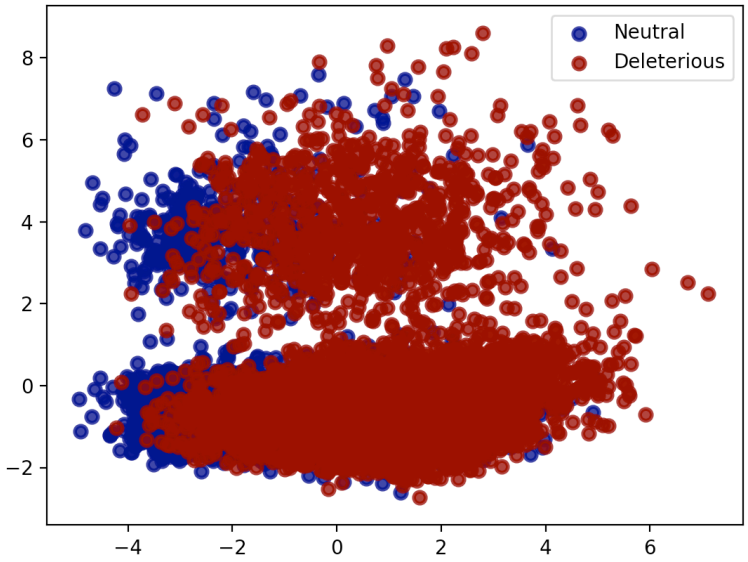

Figure 3.2: VIPUR Training Set reduced to two dimensions under Principal Component Analysis, post-feature-selection. Principal components explain 0.17 and 0.13 of the variation.

set, as shown in figure 3.2, does show areas enriched for particular deleteriousness labels, especially in the upper cluster. However, the classes are obviously not separable by a surface in this space, and the two large clusters in the data remain unexplained.

The two principal components shown together account for 20-30% of the total variation in the sample, so these clustering effects might disappear under a different means of dimension reduction. It is also not clear what, if any, biophysical property these clusters could represent; they may be batch effects or otherwise correlate to the data source. Further investigation is required.

Regardless of the source, the VTS is markedly non-uniform over the VIPUR features. This implies a challenge to any classifier trained on the entire data set: many methods we consider will perform less effectively on a mul-

timodal distribution, and unexpected complexity may be required for classifiers which can handle such an uneven surface. This non-uniformity also offers an opportunity, however: if the observed clustering does represent a consistent fact across larger sets of proteins, it could be fruitful to analyze these clusters more thoroughly in future work, to train classifiers specific to the underlying protein groupings.

3.3 Feature Selection

The original VIPUR work noted that many of the 106 features are correlated. There are particularly strong correlations[5] among the four PSSM-derived features, and (unsurprisingly) among the different quartile measurements for the structural features. To reduce the effects of training on multiple correlated features, Baugh chose[5] to perform *feature selection*—reducing the features actually considered by the classifier to a subset of the ones available.

3.3.1 Feature Selection in Original VIPUR

Baugh's method for selecting features was to train 100 classifiers on randomly chosen 80-20 splits of the VTS. For each split, a classifier is trained on four-fifths of the data and evaluated on the remaining fifth to estimate generalization accuracy. The specific classifier used was a logistic regression classifier with an L1-norm regularization penalty, which is known to promote sparsity in model weights, resulting in feature selection.[19]

Baugh chose to include a total of 20 features, on the assertion that this

number of features proved to generalize best. The 20 features with support (that is, with nonzero learned feature weights) in the largest number of trial models were used as the final VIPUR feature set.

This approach to the optimal feature subset problem is nonstandard. Additionally, no explanation is offered for the assertion that the optimal number of features is 20. Best-subset regression[16]—exhaustively searching all 2^n possible combinations of the n features—is the only certain way to find the optimal feature set, but it is computationally unfeasible for more than a few dozen features, so it cannot be the source of the claim.

Of iterative feature selection models, the traditional alternatives to best-subset regression are the stepwise methods: forward-stepwise regression,[16,21] in which a best-performing feature set is grown one-by-one; and recursive feature elimination,[21] in which a classifier is trained with all available features, the least predictive features are removed one by one, and the classifier retrained, until classifier performance degrades. However, either of these approaches would also have yielded the set of features, not just an optimal *number* of features; so they, too, cannot have been the source of the claim.

3.3.2 Feature Selection in Present Work

Feature selection is best performed by cross-validation, like any other hyperparameter.[31] Ideally one would train the target model with different feature subsets and evaluate the subset choice by considering the validation set performance. Unfortunately, implementing feature selection of this kind for most

classifiers—particularly for highly parameterized neural network and regression tree models—would result in unacceptably long training times.

Nevertheless, the original VIPUR work showed performance improvement with feature selection. Thus the present work attempts two approaches to classifier-agnostic feature selection, both using logistic regression.

In the first, I trained a classifier using a scarcity-promoting L1 regularization penalty, and selected those features with support in the resulting model. In the second, I trained a logistic regression classifier under an L2 regularization penalty, and performed feature selection using recursive feature elimination with cross-validation as implemented in `SciKit-Learn`.[32] For both methods, the strength of the applied regularization is a hyperparameter set by cross-validation. The hyperparameter space searched is constructed by an initial round of powers-of-ten logarithmic search, followed by two rounds of refinement search over ± half the last determined order of magnitude, which is sufficient to see the improvement in the loss term converge below 1%. Regularization strength is chosen separately for each of the two methods.

I tested both approaches on several different splits of the VTS, with different fold counts from 5 to 20. The results showed that the set of features selected is highly sensitive to both the feature selection method and to the subset of the VTS over which it is applied: feature selection is unstable over different splits.

3.3.3 Feature Selection Is Unstable

To systematically investigate the sensitivity of feature selection to data splits, I tested both the above methods over four iterations of five-fold splits, and tabulated the features selected. This allowed each method the opportunity to perform feature selection 20 times on different subsets of the VTS. (The same four sets of splits were used for both of the feature selection methods.)

L1-penalized logistic regression classifiers have a reputation for promoting scarcity in feature sets. Despite this reputation, the L1 models in this study retained more features than the L2 models: 40 on average for L1, vs 24 for L2. The most common result for L1 was 25 features (occurring in 11 of 20 trials); the remaining 9 models chose large sets of 59 or 60 features as significant. For the L2 models, the most common number of features selected was 22 (at 5 of 20 models), while 5 models chose 18-20 features, and 8 models retained 28-30 features. (See appendix for several views of this data.)

The feature sets chosen by the two models often fail to overlap. For instance, six features were retained by all 20 of the L1 models, but by none of the L2 models. (The features in question were `maxsub` and some quartile measures relating to backbone and side chain hydrogen bonding, and distribution of prolines). By contrast, the `maxsub2.0` feature was retained by 18 of the 20 classifiers trained with L2 penalty, but was retained by only 9 of the L1 classifiers. In many instances the L1 selection retained 59-60 features, while the L2 selection retained only 18-20 when looking at the same data. This further highlights the instability of feature selection and its sensitivity to method.

Four of the five sequence-only features were retained in every classifier model trained. The fifth, the `pssm_native` feature, was retained by every L2 model and no L1 models. This is almost certainly because of its correlation with other PSSM metrics: given two correlated features, a model penalized with an L1-norm regularization term can shift all the weight from one to the other. Under an L2 regularization scheme this would result in a large penalty, so it is more favorable to retain both features at lower weights. This is the only instance I have found that suggests an influence from feature correlation.

Category	**Count**	**Criteria**
Most Important	15	Retained in all or nearly all models
Commonly Selected	12	Retained at least 9 times by both models
Low Importance	32	Retained by only one model, or only rarely
Uninformative	43	Never selected by any model

Table 3.1: Categorization of structural features into groups based on the frequency with which they are selected by both feature selection methods. The cutoff of 9 selections was chosen because many more features were selected 9 times than 10 times.

Table 3.1 categorizes structural features into four groups. Roughly 15% of the structural features fall into the "Most Important" category. The top two categories collectively represent a quarter of the structural features.

My results do not fully support Baugh's original feature selections.[5] That set of 20 features included all 15 of the "Most Important" features and 3 of the "Commonly Selected" features. It also contained one feature which was selected almost exclusively by the L1 model, and one was never retained by any of my models at all.

Baugh claims that feature selection improves the model by reducing overfitting. Contrary to this assertion, these results demonstrate that many of the structural features simply do not have significant predictive value.

These results are important for two reasons. They support the overall contention that current VIPUR performance is limited more by the available features and data than by machine learning methods. They also illustrate that at present, any choice of features will be somewhat arbitrary. Limiting the feature set does seem to improve results, but with no principled way to choose features automatically, I argue for treating feature selection as a pure hyperparameter and not an integrated part of the VIPUR pipeline.

For the purposes of the experiments in the present work, I have chosen to use 27 features—those categorized as "Most Important" and "Commonly Selected"—as the selected subset of features. This seems an appropriate balance between including features with some predictive utility and excluding those which are more marginal or which may have negative utility.

3.4 Cross-Validation Methodology

Much of the present work depends on cross-validation. As such, division of the VTS into representative folds is of paramount importance. Baugh notes two important requirements of cross-validation splits for the VTS: first, the label bias must be maintained; and second, proposed splits must not include multiple variants of the same protein in different folds (as this would unfairly bias the results, by training on examples which are direct analogues of the test data).[5]

Standard `SciKit-Learn` library routines exist for providing a stratified k-fold cross validation split (which would preserve label bias) and a grouped cross-validation split (to account for situations where samples have a natural grouping that might result in bias), but there is no easy way to combine the two. Instead, I have implemented cross-validation split assignment in the `VipurCrossValidator` class. My implementation groups proteins together according to UniProt ID, then assigns the groups to different data folds in decreasing order of number of variants, to ensure that large groups of related proteins do not dominate any individual fold. The resulting candidate splits are then sorted according to their own label bias, and groups of proteins are swapped from the folds whose label bias is farthest from the overall dataset's. Because the protein groups are chosen for swap at random from the set of inappropriately biased samples, every step brings the folds more in line with the bias of the overall data set. The result is that label bias tolerances as low as 0.5% can be reached nearly instantaneously. The procedure will stop if offsetting groups of proteins cannot be found past a certain limit. However, in practice this stopping criterion is not needed, as the data set is sufficiently large and granular to accommodate the swaps.

Several challenges remain, however. First, groups are identified by UniProt ID. But there is no guarantee that UniProt IDs are assigned with sufficient rigor to identify all related proteins; we may have variants that should travel together which are not detectable under this system, because they have not been labeled consistently. Second, as discussed above, the VTS is not uniform; without more

insight into the divisions within the VTS, we cannot to ensure both (or all?) natural groups are represented equally in each fold. Third, it is possible the swapping process could result in larger protein groups collecting together in some folds (though I did not observe this). Finally, my method does on some occasions make swaps of slightly different numbers of proteins, leading to minor drift in the total number of records appearing in each fold. The magnitude of the drift is on the order of 1% and unlikely to have noticeable effect, but a future revision may attempt to limit this possibility.

3.4.1 Cross-Validation and Evaluation

Given its importance in evaluating the results that follow, I will quickly review cross-validation, and how the new VIPUR's results are evaluated.

To ensure an accurate estimate of the classifier's generalization performance,[9] the model should be tested on data it has not been trained against. The natural inclination is to take a representative share of the data set, and hold it aside as a *test set*, to be evaluated once we have a fully trained classifier. However, this risks that the test set may have some systemic bias which makes all of the estimates inaccurate or which favors certain classifiers.

Instead, we split our data into k even folds (five in the case of the present experiments) and go through the process of optimizing hyperparameters and training the model k times, once for each fold. This allows computation of both average accuracy of each model, and the variance of the results obtained, which together hopefully provide a stronger predictor of generalization performance

than a single holdout set could provide.

In order to provide the best model possible for each test set, we also tune hyperparameters through cross-validation, as follows. Assuming a 5-fold split, one split will be held out as the test set. With the remaining four folds, we can train four models: each one will be trained on three of the four training folds and tested on the fourth, which serves as a *validation set*. We can use the estimate provided by the validation set to choose the optimal settings, then train a classifier with those optimal settings using all four training folds, before finally evaluating on the originally held-out test set. This process is repeated for each of the k outer folds, to make sure the entire data set has been used as test.

In the present VIPUR implementation, the 'test' mode automates this cross-validation process. To support this functionality, the `VipurCrossValidator` class reproduces consistent train-test splits with an optional holdout, and the `VipurBinaryClassifier` interface provides a hook for each classifier to use cross-validation to tune any hyperparameters which are relevant to it.

Having discussed the structural framework of the new, modular VIPUR, and the data set used for experiments, I now turn to the classifier modules themselves. The following chapters will discuss baseline approaches and formalize the two classifier types used in the original VIPUR, then turn to newer methods first applied to the VTS in the present work.

4

Baseline Classification Approaches

The original VIPUR work considered two main types of classifiers: logistic regression classifiers and support vector machines, both traditional approaches to machine learning problems which have been in common use for decades. In this chapter I present a re-evaluation of these two methods in the context of the

new VIPUR. First, I will discuss three very basic approaches which provide a baseline performance against which all other classifiers should be judged.

4.1 Modal, Random, and Decision Stump Baselines

The goal of the baseline classifiers is to characterize the data itself in the absence of any sophisticated approach—since claiming 95% accuracy is much less impressive for data with a 96% label bias. These methods do not attempt to be strong predictors.

The specific approaches we discuss are *modal* selection (always return the most commonly represented category); *random* selection (flip a weighted coin, whose weight is determined by the label bias of the data); and *decision stumps*. The idea behind the latter method is to always return the class suggested by the single most-predictive feature.

The results of these methods are summarized in Table 4.1:

Classifier	**Avg Accuracy**	**Std Deviation**	**Best Feature**
Modal	60.56%	0.24%	n/a
Random	52.09%	0.87%	n/a
Stump	75.96%	2.10%	`pssm_difference`

Table 4.1: Performance of baseline estimators: modal (always apply the most frequent class label), random (coin flip weighted by the data label bias), and decision stump (always choose the label predicted by the single most-predictive feature).

The results of the modal and random baselines are straightforward and unsurprising. The decision stump, however, has astonishingly high accuracy and mer-

its further explanation.

4.1.1 Decision Stump

The idea behind a decision stump[22] is to simply follow whatever the most predictive feature says. We determine the single most predictive feature by training a logistic regression model and choosing the feature with the largest weight as "best." This process is very stable; in dozens of rounds of tests it always yielded the `pssm_difference` term as the most informative feature.

Having identified the single most significant feature, we then learn a single-feature logistic regression classifier using it.

As table 4.1 shows, this feature is strongly predictive; a classifier built using this single feature has an accuracy of 76% as against a modal baseline/label bias of 60.5%. The original VIPUR work claims a generalization accuracy of 80-81%;[5] thus, three-fourths of VIPUR's improvement over a modal classifier rests on this one feature.

Other PSSM features seem to be similarly powerful, with all but `pssm_native` appearing in every set of selected features, and `pssm_native` itself appearing in every feature set trained using the L2 method. Additionally, repeating the decision-stump baseline with the `pssm_difference` feature deliberately excluded always yields a different PSSM feature.

With this baseline in mind, we explore the two classifiers considered for the original VIPUR work.

4.2 Logistic Regression Classifier

Logistic regression models are closely related to linear regression methods, except with the goal of producing a binary categorical outcome rather than predicting a real value.[24] In both cases, the central notion is a linear combination of the weighted elements of the feature vector. Logistic regression then passes the result through a function that maps the resulting score to a probability.

4.2.1 Formulation of Logistic Regression

We consider a set of data samples n, each represented as a real-valued vector of dimension k, where k is the number of features. Thus, for the VTS, each variant is a vector $\{\vec{x}_i \in \mathbb{R}^{106}\}$, $1 \leq i \leq 9447$. We represent the categorical labels as corresponding values of $y \in \{0, 1\}$, where a value of 0 represents a neutral variant and a value of 1 represents a deleterious variant. Our goal is to learn a function $f(\vec{x}) \rightarrow y$ mapping feature vectors to categorical values.

"Features" may be some kind of direct measurement or observation of the data; they may also be a different *basis function*[9]—there is nothing preventing us from treating the square of a measurement as a feature, in place of (or alongside) the measurement itself, for example.

Logistic regression models the probability of the data sample being correctly classed in the positive category as a weighted sum of the features, passed through a *sigmoid function*, the logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$, which converts the corresponding value to the range $(0, 1)$ so it can be interpreted as a probability.[9]

Formally:

$$P(y = 1|\vec{x}) = \frac{1}{1 + e^{-(\vec{w}^T \vec{x})}}$$

where we also assume a unit-valued "feature 0," $x_0 = 1$, to allow the overall bias term to be included in the weight vector $\vec{w}$. The probability of the sample falling into class 0 is complementary.

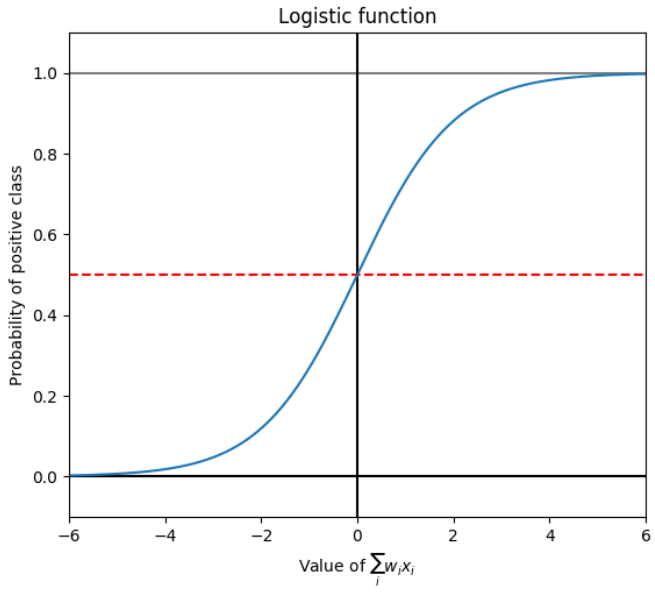

Figure 4.1: An illustration of the logistic function. Values of $\langle \vec{w}, \vec{x} \rangle \in (-\infty, \infty)$ are mapped to values in $(0, 1)$ with a natural interpretation as probability values. The red dotted line indicates the value $y = 0.5$, and acts as a crossover point from negative class prediction to positive class prediction.

Training the logistic regression classifier thus reduces to finding the optimal feature weight vector $\vec{w}$ which maximizes this probability. The total probability of correctly classifying every sample in the data set can be expressed as the product of the probability of correctly classifying each of the m samples i in

the data set,

$$\prod_{i=1}^{m} P\left(y_i\right)^{y_i} \left(1 - P(y_i)\right)^{(1-y_i)}$$

where we take $P(y_i) = P(y_i = 1|\vec{x}, \vec{w}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$. Exponentiation to y_i and $1 - y_i$ selects only the probability for the correct classification of the data point, since y_i will be 0 and $1 - y_i$ will be 1 if the correct categorization is 0, and vice-versa. Thus the term describing the model's probability prediction for the incorrect class is replaced with 1, and does not influence the overall product.

Because the logarithm function is monotonically increasing, the value of x which maximizes a function is the same as the value of x which maximizes the log of that function: $\arg\max_x \log f(x) = \arg\max_x f(x)$. So we can take the logarithm to turn the product into a summation, and also change the sign to yield a loss function for minimization:

$$\mathbb{L}(\vec{w}, \vec{x}) = -\sum_{i=1}^{m} y_i \ln P(y_i) + (1 - y_i) \ln(1 - P(y_i))$$

Since we cannot change the data $\vec{x}$, the problem reduces to choosing the weight vector $\vec{w}$ minimizing this loss.

4.2.2 Regularization

This formulation of the loss function allows a natural explanation of regularization. The goal of regularization is to discourage overfitting, by promoting models which do not place exaggerated weights on individual feature terms. The two simplest models of regularization are L1 and L2 regularization, which are both

characterized by adding a norm of the weight vector to the loss function:

$$\mathbb{L}(\vec{w}) = -\sum_{i=1}^{m} [y_i \ln P(y_i) + (1 - y_i) \ln(1 - P(y_i))] + \lambda \|\vec{w}\|_1$$

for L1 normalization and

$$\mathbb{L}(\vec{w}) = -\sum_{i=1}^{m} [y_i \ln P(y_i) + (1 - y_i) \ln(1 - P(y_i))] + \lambda \|\vec{w}\|_2^2$$

for L2 normalization. Here $\| \cdot \|_1$ denotes the L1 norm, or $\sum_i |w_i|$ over the elements of the vector $\vec{w}$, and $\| \cdot \|_2$ denotes the L2 (Euclidean) norm $\sqrt{\sum_i w_i^2}$ over the elements of $\vec{w}$. In both cases, we have added a term which counts extra weight as extra loss. This causes the optimization to favor smaller weights. Moreover, since an L1 norm penalizes the sum of all weights, it will favor reducing weights of less predictive features to 0 (thus promoting scarcity and feature selection). By contrast, since each weight is squared in the L2 norm, this formula more heavily penalizes models which concentrate weight over a small number of features.

In both cases, the value λ is a hyperparameter set by the model designer which determines the strength of the regularization. For the present investigation it will usually be encountered as $c = \frac{1}{\lambda}$, and will be set by cross-validation.

4.2.3 Fitness to Purpose

Logistic regresssion classifiers offer the advantages of being straightforward to build and train, and straightforward to interpret—the weight assigned to each

feature specifies that feature's importance to the final classification.

However, the logistic regression classifier also imposes certain strong assumptions on the data. The most obvious is that logistic regression models are linear in the weights of the features: the exponentiation in the sigmoid function will always be a sum of weighted features. Capturing interdependence between features requires extensive feature engineering to add a new feature that expresses the relationship. Similarly, applying nonlinearity to individual features would require hand-crafting an appropriate basis function.

These limitations greatly disfavor the use of categorical values for features (since the ordering of the categories imposes strong assumptions about how they should be weighted) and limit the usefulness of raw-data features, which cannot directly influence the weights of other, more derived, features.

4.2.4 Performance

Table 4.2 summarizes the performance of several logistic regression classifiers. Figure 4.2 illustrates these accuracies in the context of the data set and decision stump baseline performance.

Table 4.2: Performance of the logistic regression classifier under L1 or L2 regularization penalty, with the complete feature set or the selected feature set.

Classifier	**Avg Accuracy**	**Std Deviation**	**C**
L1, all features	79.87%	1.74%	0.014 - 0.05
L2, all features	79.84%	1.67%	0.005
L1, selected features	80.36%	1.76%	0.14 - 0.6
L2, selected features	80.29%	1.77%	0.012 - 0.05

Performance of all models is comparable within model variation. There

is an improvement of around four percentage points over the decision stump baseline. Using the selected feature subset improves accuracy by around half a percentage point. Feature selection also reduces the overall amount of regularization required for best results (lower values of C correspond to stronger regularization).

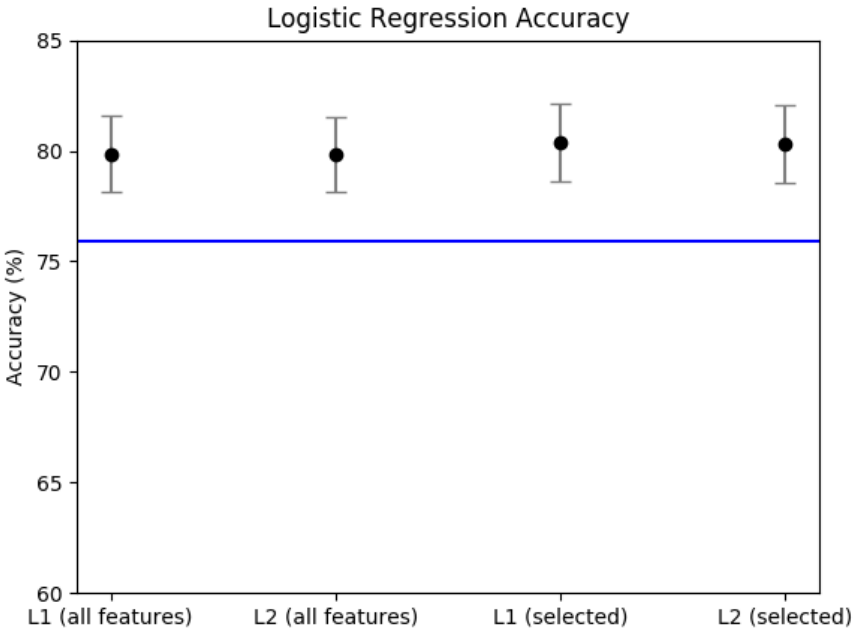

Figure 4.2: Accuracy ($\pm$ 1 standard deviation) of logistic regression classifiers trained with L1 and L2 regularization penalties, with full feature set and selected feature subset. Blue line indicates decision stump baseline (75.9% accuracy). Y-axis begins from 60%, the label bias of the sample.

Figure 4.3 visualizes the AUPR and ROC curves for the L1 classifiers, which performed slightly better than L2 classifiers for either data set. The figure highlights the variation existing in the data set, with performance on the first fold exceeding one standard deviation from the mean.

The results cited here are comparable to, if slightly below, the 81% accuracy quoted in Baugh's original work.[5][6] As one additional check, I reran this test using the set of features selected in the original VIPUR work.[5] The L1 classifier had a mean accuracy of 80.46% with standard deviation 1.9%, and the L2 classifier yielded average accuracy 80.5% with standard deviation 1.93%: again comparable to, if slightly below, the originally reported accuracy.

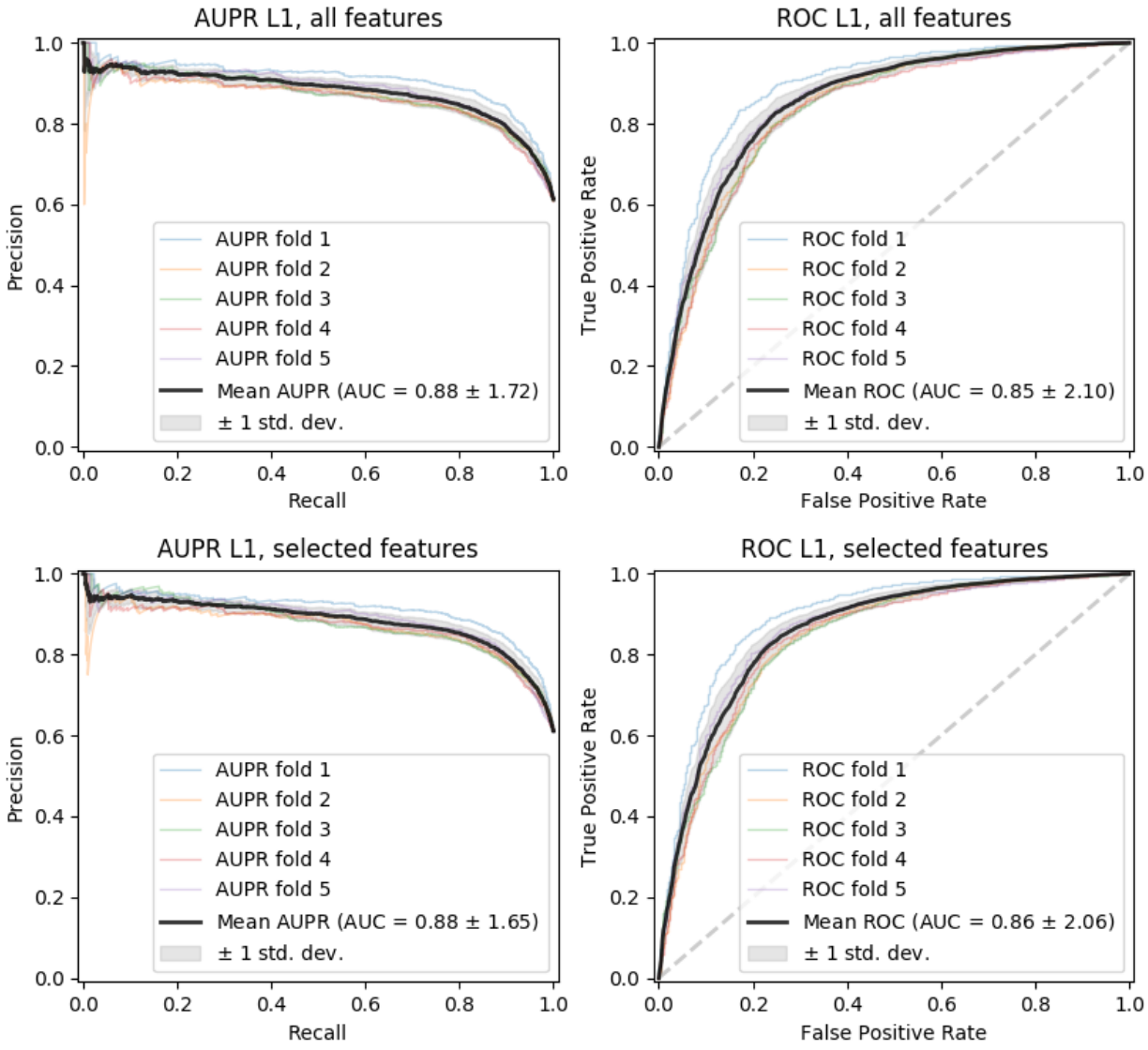

Figure 4.3: AUPR and ROC curves for logistic regression classifiers trained with L1 regularization penalty, with and without feature selection. L1 models show slight but insignificant accuracy improvement over L2 models for either feature set. "Fold" refers to the identity of the cross-validation test partition.

4.3 Support Vector Machines

Support vector machines determine an optimally separating, or largest-margin, hyperplane between two classes of data samples in some high-dimensional space. The "support vectors" refer to the set of data points which lie on the margin of the linear (hyperplanar) class boundary;[14] essentially, the set of points from the two classes which are closest together. In simplest terms, we want to divide the two classes with a line that is maximally far from the nearest instances of either class. Two relevant extensions to this framework are also discussed in the formalization below.

4.3.1 Formulation

The goal of a support vector machine model is to determine an optimal *hyperplane* separating data points in the two classes. A hyperplane is a generalization of a line to higher-dimensional space. Formally, a hyperplane in a $p-$dimensional space is a "flat affine subspace of dimension $p-1$."[24] (Most of this discussion follows James *et al.* 2013). The hyperplane, like a line in 2D space, is a set of points; specifically those $\vec{x} \in \mathbb{R}^{P-1}$ for which $\vec{w}^T\vec{x} = 0$ (where we again use an implicit unit value $x_0 = 1$ so that a scalar offset can be applied in compact form). Since the hyperplane has dimension $P-1$, the remaining dimension becomes in effect a number line, and every sample vector in the space falls into one of three places on that line. The sample vector either lies on the hyperplane (in which case the equation to 0 holds), or it lies on either side of the hyperplane (in which case the weighted element sum will be > 0 or < 0). If

we express the classifications as $y_i \in \{-1, 1\}$, then this equation can be written as $y_i(\vec{w}^T\vec{x}) > 0$, which would correspond to a hyperplane for which all samples are correctly classified.

To find the hyperplane creating maximum margin thus means finding the weight vector $\vec{w}$ which maximizes M for:

$$\|\vec{w}\|_2^2 = 1$$
$$y_i(\vec{w}^T\vec{x_i}) \geq M \; \forall i \in \{1, \ldots, n\}$$

Normalizing the weights to 1 ensures that the left-hand side of the inequality corresponds to the perpendicular distance of each point from the separating hyperplane. Thus M represents the margin, or distance between the hyperplane and those points nearest to it on either side.

The sample vectors supporting the hyperplane (the support vectors) will then be exactly those samples which lie on the margin: the set of samples for which the equality holds.

The first extension to this framework is to observe that not all data sets are actually linearly separable; even if they are, exact linear separation may result in much narrower margins than could be achieved if we ignored a few outliers. A wider margin is desirable because we expect that a solution with wide margins will generalize better: by avoiding a tight fit to the available data, we avoid overfitting as well. Figure 4.4 illustrates this idea.

The problem of tight margins due to outlying data points can be resolved

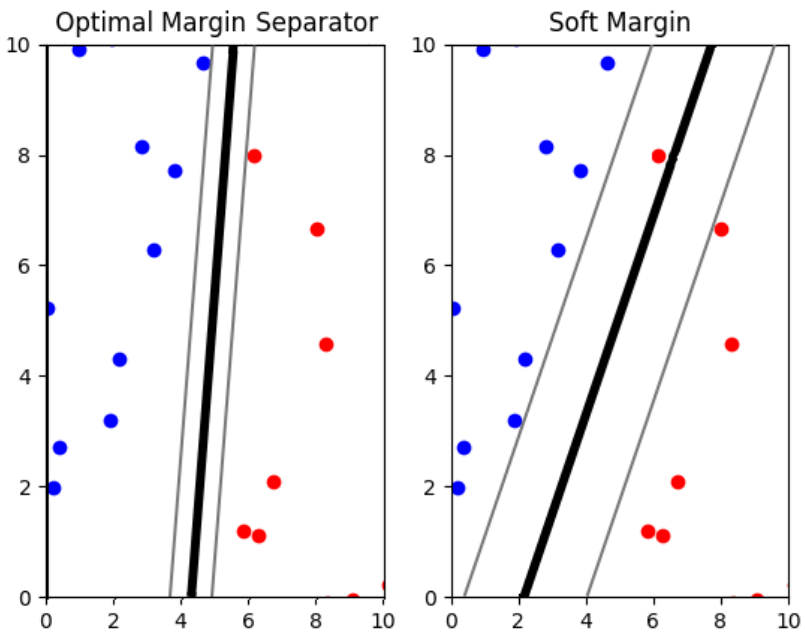

Figure 4.4: Optimal-margin perfectly separating hyperplane (left) and soft margin hyperplane with larger margin (right). The "supports" are the two blue and red dots touching the grey lines; the margin is the perpendicular distance from black lines to grey lines. The left image shows the largest margin that allows for perfect separation. The right image shows that a much larger margin can be recovered by accepting the misclassification of one red dot.

by adding a misclassification error term ε_i which records the amount by which each point has been misclassified:

$$\|\vec{w}\|_2^2 = 1$$
$$y_i(\vec{w}^T\vec{x_i}) \geq (M - \varepsilon_i)\,\forall i \in \{1, \ldots, n\}$$
$$\varepsilon_i \geq 0, \sum_i \varepsilon_i \leq C$$

where ε_i measures the degree to which each sample violates the margin, and C

sets an overall misclassification budget (a hyperparameter indicating how flexible we will be to ensure a greater margin). This is known as the *soft margin*[14] support vector machine, or *support vector classifier*. The `SciKit-Learn` library used in the present work implements support-vector classifiers.[32]

4.3.2 Basis Functions and High-Dimension Projection

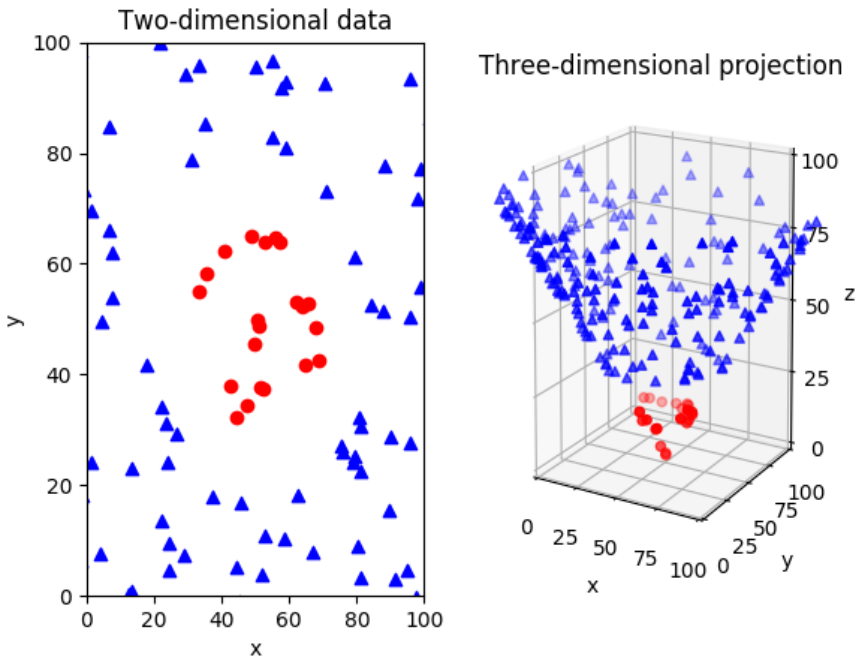

Figure 4.5: An example of projection into a higher-dimensional space. In the left figure, the red and blue classes are obviously not linearly separable. In the right figure, the points have been projected into a three-dimensional space in which they can be separated by the plane $z = 20$. The basis function $\sqrt{(x-50)^2 + (y-50)^2}$ provides the z-axis value.

Samples may not always be linearly separable, even with the soft margin, and the relationship may simply not be linear in the features proper. We can solve this problem by mapping the data points to a higher-dimensional space in

which they *are* linearly separable, through a *change of basis function*. As indicated in the definition of features at the beginning of this chapter, features are not restricted to direct observations of the samples, but can also be functions of direct observations.

The addition of these new supplementary (basis-function) features is how SVM classification can correspond to a projection into a higher-dimensional space: the 106 features of $x \in \mathbb{R}^{106}$ become 212 features when the square of each element of x is included in the feature set along with the elements themselves.

4.3.3 Kernels and the Kernel Trick

Mapping a sample to a high-dimensional space by computing many functions of each feature can become computationally expensive. Fortunately there are a few facts that reduce the computational burden. First, the optimization problem (and the entire classifier) depend solely on the support vectors: samples beyond the margin do not actually influence the classifier in any way.

Second, the solution to the optimization problems above depends only on the *inner product*,[14] a category of functions which map two vectors in a (possibly infinite-dimensonial) space to a scalar distance between them—the most familiar inner product being the dot product.

With these two facts in hand, we can evaluate any support vector machine simply by taking the inner product between the new sample x being tested and the supports x_i:

$$f(x) = k + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

where $\langle \cdot, \cdot \rangle$ corresponds to an inner product (which, again, can be understood as a similarity function between the points), and S is the set of support vectors.

The inner product can further be replaced with a *kernel function*, a class of functions extending the inner product with slightly more complex relations. An appropriately chosen kernel will correspond to a mapping into the higher-dimensional space discussed above, without actually projecting all the samples into that higher-dimensional space, a technique known as the *kernel trick*.

Popular kernels implemented in the SciKit-Learn library include linear, polynomial, radial-basis, and sigmoid kernels,[32] formulated respectively as:

$$\begin{aligned} \text{Linear:} \quad & \langle \vec{x}, \vec{x_i} \rangle \\ \text{Polynomial:} \quad & (\gamma \langle \vec{x}, \vec{x_i} \rangle + r)^d \\ \text{RBF:} \quad & \exp\left(-\gamma \|\vec{x} - \vec{x_i}\|^2\right) \\ \text{Sigmoid:} \quad & \tanh \gamma \langle \vec{x}, \vec{x_i} \rangle + r \end{aligned}$$

Clearly there is a richer set of hyperparameters for SVM models than for logistic regression models. In addition to the d, γ, and r terms above, which correspond to the dimension of polynomial kernels, the locality of a data point's effects, and the offset, there is also the C or margin-softness term, which can have a strong impact on the generalization of the trained model. While I have tried to explore likely values for these, I have not tested every possible combination. Further experimentation might yield further incremental improvements in performance, though it is unlikely to show radical improvement.

4.3.4 Fitness to Purpose

SVMs offer several potential advantages over conventional logistic regression models. The most obvious differences are increased nonlinear flexibility (for a given amount of feature engineering) and reduced computational overhead for training and evaluating complex nonlinear relationships. In general, the code support in SVM libraries exposes a more powerful set of classifiers—assuming correct choice of kernel function and hyperparameters.

However, SVMs also present several drawbacks. The most obvious is the lack of easy interpretation. In practice, since we learn weights on different sets of inner products, rather than on features directly—and since the feature comparison may be in some high-dimensional space that is never explicitly referred to—SVM models are not as straightforward to explain as logistic regression models. SVMs also do not naturally provide probability predictions: we can tell how far a point is from the margin to qualitatively identify confident and less confident predictions, but it is computationally expensive and somewhat subjective to quantify. Thus, precision-recall data are not natural outputs of this classifier. Finally, the flexibility and power of the models is a two-edged sword: more hyperparameters means more computation to search for optimal settings, and more risk of overfitting.

Given the roughly equivalent performance, Baugh chose to use the logistic regression classifier for the original VIPUR on the grounds of easy interpretability.[5]

Finally, while the formulation given above appears quite different from

that for logistic regression, there are deep similarities between the two approaches.[24] (The construction of a weight vector multiplying the sample vector might already suggest this relationship.) For our purposes, the biggest difference is the ready availability of premade kernels (and the corresponding implicit nonlinearity) when working with libraries implementing SVMs.

4.3.5 Performance

Table 4.3: Performance of SVM classifiers.

Classifier	**Avg Acc**	**Std Dev**	**Hyperparameters**
RBF, all features	**79.41%**	1.17%	C: 0.9 - 8, γ: 9e-4 - 1.3e-3
Sigmoid, all feats	78.90%	1.37%	C: 11-600, γ: 7e-6 - 9e-5
Linear, all feats	78.89%	1.30%	C: 8e-4 - 6e-3
Poly 2, all feats	71.39%	0.313%	C: 9e-3 - 0.9, γ: 0.013 - 0.13
Poly 3, all feats	76.72%	1.00%	C: 1.1e-3 - 800, γ: 1e-3 - 0.11
RBF, selected feats	**80.42%**	1.27%	C: 0.8 - 100, γ: 6e-4 - 9.3e-3
Sigmoid, sel feats	79.46%	1.88%	C: 1.4e-3 - 1400, γ: 3.6e-5 - 16000
Linear, sel feats	79.33%	1.54%	C: 1.3e-3 - 100

Table 4.3 shows the performance of SVM classifiers. (C in this context refers to the margin tolerance, not a regularization coefficient.) There is considerable variation in the optimal hyperparameter settings between folds, often by more than an order of magnitude. Polynomial kernels underperformed other kernel types by such a wide margin that they were not tested with the selected feature subset.

Figure 4.6 illustrates the performance of the different classifiers. Polynomial kernels clearly underperform. There is a slight advantage for a selected feature subset; however this results in increased variance in the resulting model

performance. Models trained with radial basis function kernels perform noticeably better than others.

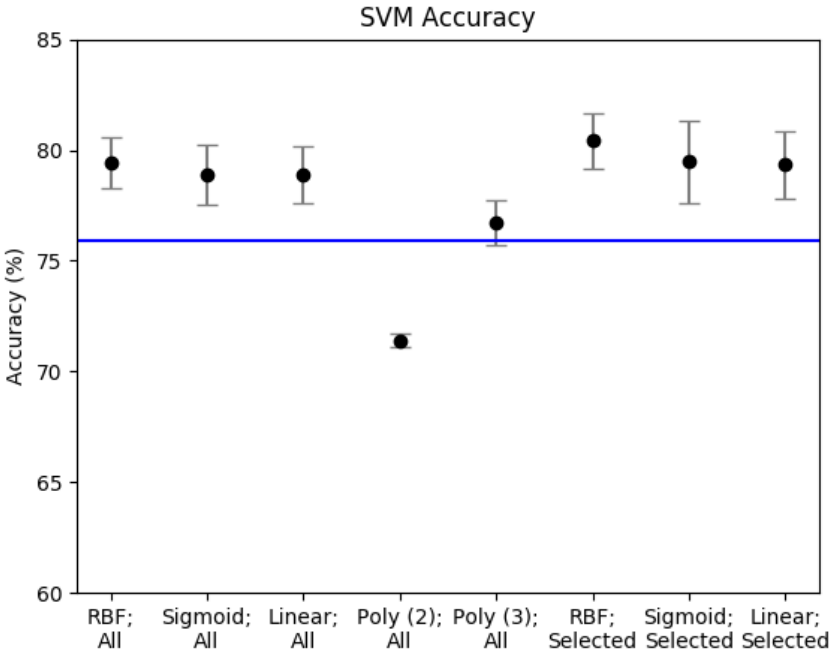

Figure 4.6: Accuracy of tested SVM classifiers, $\pm$ one standard deviation. Each record is labeled with the kernel and feature set used. Poly (2) and (3) indicate second- and third-degree polynomial kernels, respectively. Blue line shows decision stump performance baseline.

Without feature selection, as in Baugh's original work, the SVM models do not outperform logistic regression models. However, with a selected subset of features, the best-performing radial basis function model does achieve a very slight (0.6%) improvement over the best logistic regression model. The difference is well within the margin of error. The SVM models tested do show lower variability than the logistic regression models.

5

Comparison of New Approaches

The previous chapter considered methods coming directly out of statistical analysis, which generally map features directly to classifications. This chapter considers methods which look to more complex relationships between data samples and classifications. These include more nonlinearity, combining

multiple models, and highly parameterized neural network models. These models are generally more powerful than the ones considered previously: if more powerful hypothesis sets are capable of generating improved results on this data set, they will be found here.

5.1 XGBoost

XGBoost[12] is a departure from the models considered thus far, in two ways. First, it belongs to a class of solutions known as *ensemble methods*. These are approaches which do not try to learn a single high-performing model, but instead train a series of weaker models, and treat their consensus position as a single collective model.[35]

Second, the underlying classifiers trained in XGBoost are *decision* (or *regression*) *trees*, which are quite different from the weighted-feature-vector approaches of the previous chapter. While logistic regression and SVM models do one operation simultaneously on all the features and collect the result, decision trees make a decision by looking at multiple small subsets of the features in an ordered series of decisions,[37] dividing the data set at each node into groups of maximal homogeneity.

Below I formulate both of these concepts, then review the application of XGBoost to the VIPUR training set.

5.1.1 Decision Tree Formulation

A *decision tree* is a structured model used for classification or regression.[35] Decision trees are also sometimes referred to as *CART* or "Classification And Regression Trees," although this is properly the name of a specific formulation (the one used in XGBoost).

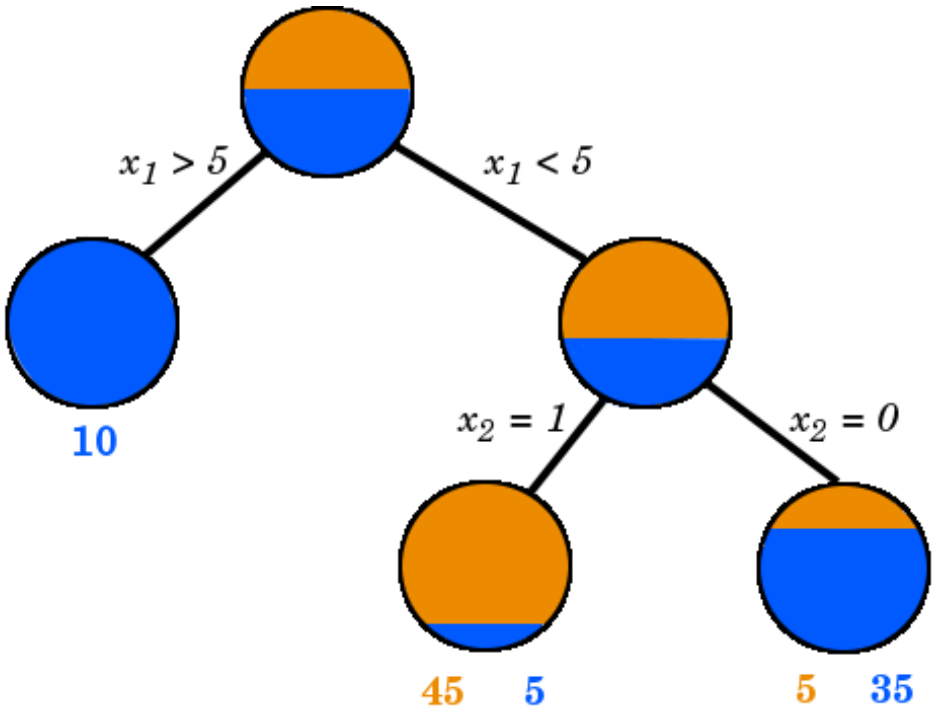

Figure 5.1: Example of a simple decision tree. The first split considers feature x_i. Values less than 5 are sent to the left node. Since this node is entirely blue, any test points sent to this node would be predicted blue with 1.0 probability. Values greater than 5 are sent to the right node, where feature x_2 was found to result in the greatest separation of category elements among the features remaining. From the distribution of training points, any sample for which $x_1 < 5, x_2 = 0$ would be predicted to fall into the blue class with 0.875 probability, and any sample with $x_1 < 5, x_2 = 1$ would be predicted blue with probability 0.1.

The decision tree (as in figure 5.1) categorizes data by applying simple tests to a feature vector over multiple ordered decision points. If this model is

visualized as a tree, the inner nodes represent decisions, and the leaf nodes represent either a score[12] or the probability of a particular categorical classification, represented as the fraction of examples meeting the decision criteria which fall into the class of interest.[30]

In learning the tree, after each split we must decide whether and how to split further. There is clearly no need to split a node whose members are all correctly categorized; and some implementations (like XGBoost) impose a maximum depth hyperparameter beyond which no further splits will be considered.

If these stopping criteria have not been reached, we will split the node if doing so provides a sufficient increase in quality. Split quality can be measured in several ways. One option is the misclassification rate (a simple proportion of the elements in the node that are misclassified). More sophisticated approaches use a value like *information gain*,[30 35] equivalent to the loss in *information-theoretic entropy*, over the split.

Within information theory, entropy[38] is formally defined as the negative of the probability of an outcome, times the log of this probability. Given a random variable V with possible outcomes v_k, we define entropy as:

$$H(V) = -\sum_k P(v_k) \log P(v_k)$$

where $P(v_k)$ is the probability of the random variable V taking on the value v_k. Effectively, this quantifies the randomness of a distribution of outcomes.

If we consider V to be the classification of a randomly chosen member

of the node's set into one of the two classes, then we can express the node's entropy as the sum of the ratio of exemplars of each class to the total, times the log of this ratio:

$$H(\text{node}) = -\left(\frac{p}{p+n}\right)\log\left(\frac{p}{p+n}\right) - \left(\frac{n}{p+n}\right)\log\left(\frac{n}{p+n}\right)$$

where p and n indicate the number of data points at that node falling into the positive and negative classes, respectively.

The information gain over a split is the difference between the parent node and the entropies of the two child nodes. This provides a natural link between the increase in purity of the child nodes, and the probability of one class label correctly describing all the elements in that node.

Most approaches to building decision trees choose to split along the feature that results in the greatest increase in node quality. There is, however, a risk that this greedy approach will not actually result in the best-fitting possible tree: as with the optimal-feature-subset problem discussed in section 3.3, there is always a risk that some combination of individually less-optimal features could collectively provide better performance than one stronger feature, but a greedy approach cannot always discover these relationships.

The same feature can appear in multiple nodes in different paths along the decision tree; and real-valued features may have different cutoff points in different decision nodes. This automatically incorporates non-linearity into the model, since the exact splits being made depend on the path taken, rather than solely on the linear change in the value of the feature.

5.1.2 Boosting Formulation

Unless data really are generated by a set of decision rules, it is unlikely that we will train a single tree which makes good predictions in the general case.

Gradient boosting methods come out of the multi-learner paradigm reinvigorated by AdaBoost in 1997.[17] This class of meta-classifier centers the observation that a diverse collection of weak, but better than random, classifiers will converge to collectively form a strong classifier. Such a collection of classifiers is known as an *ensemble*, and is usually produced by an iterative process in which individually-weak classifiers are trained and added to the collection.

There are several methods to choose what learners to add to the ensemble. In a wisdom-of-crowds sense, any collection of weak learners could lead to improved results; but polling new models at random is inefficient. Moreover, it has a potential flaw: to make an improved collective model, the weak models need to be complementary. Otherwise, the product is computational groupthink, where several weak learners reinforce each others' incorrect predictions.

The original AdaBoost algorithm resolved this problem by increasing the relative weights of those samples misclassified by the previous addition.[17] This encouraged the new model to be different from, and complementary to, the existing collection.

Subsequent work has explicitly considered the collection of learners as a model for which a loss function can be written and optimized. Instead of training the next model on a reweighting of the data set, we can learn a model that explicitly compensates for the discrepancies between the current ensemble's es-

timates and the data.[20] Formally, we add to the ensemble a t^{th} new classifier $f_t(\vec{x})$ which minimizes the ensemble loss:

$$\mathbb{L}^t = \sum_i^n l\Big(y_i, \hat{y_i}^{(t-1)} + f_t(\vec{x_i})\Big) + \gamma T + \frac{1}{2}\lambda\|w\|^2$$

where $\hat{y_i}^{(t-1)}$ is the current ensemble's prediction for sample $\vec{x_i}$, $l(\cdot,\cdot)$ is an appropriately chosen residual loss function, T is the number of leaf nodes in the new model, w is the set of scores on each leaf, and γ and λ are hyperparametric regularization strengths.[12]

Instead of training directly to the loss function, under *gradient boosting* we train to the gradient of the loss function with respect to the classifications, following standard gradient descent procedures. This is the approach implemented in XGBoost.[12]

Once a new model has been built to this target, a scaling factor is chosen for each leaf that minimizes the loss for the samples in that leaf[20] (which can be thought of as a leaf-specific step size along the gradient). The final prediction for any instance in one of these ensemble models is the sum of the scores of the leaves into which the instance is placed by each of the trees.

5.1.3 Avoiding Overfitting in Complex Models

Between the native non-linearity in decision/regression trees and the power of adding a potentially boundless number of models, gradient-boosted approaches are at high risk for overfitting. Yet as powerful as the approach is, there is an

equally broad array of tools within XGBoost to combat overfitting.

I have already mentioned setting maximum tree depth, which also limits the total number of features considered by each tree.

In addition to stopping tree growth, XGBoost also features a learning rate parameter, which controls the extent to which new models influence the ensemble. Learning rates below 1 (also known as a shrinking factor) effectively slow descent along the gradients, which generally leads to better convergence.[16]

XGBoost also makes use of techniques common to random forest approaches:[16] dropout of rows and columns from the feature matrix. In column dropout, each new model is allowed to consider only some (hyperparametrically set) fraction of the available features. Column dropout allows models to consider features that might otherwise be eclipsed by one or two features with coincidentally strong predictive power on the data set. Dropping out rows also avoids overfitting by randomly removing some of the training observations. This reduces the impact of outlying data points. Bagging,[10] or bootstrap aggregation, is a particularly sophisticated approach to diversify models in ensembles: a fraction of the data samples are removed from the training set and replaced with randomly chosen duplicates of the other rows. This helps compensate for outlier bias in the sample data.

Finally, model weights can be regularized under the L1 or L2 penalties discussed previously, or an ElasticNet[42] combination of the two, if desired.

5.1.4 Fitness to Purpose

As of this writing, the XGBoost implementation of gradient boosting is considered state-of-the-art for data sets of extracted features,[12] and is an algorithm of choice for many top performers in online machine learning competitions.[20]

The specific advantages of the approach include easily integrating categorical features (since splits need not depend on a linear multiple of feature values), and naturally incorporating both nonlinearity and feature selection into the model creation process without any special effort. Finally, these models are very powerful: they are the only models tested which produced classifiers that could achieve perfect accuracy on the VTS when trained on the entire VTS.

The drawback of any powerful approach is the need for careful hyperparameter tuning to avoid overfit. Moreover, while decision trees themselves give clear explanations for their decisions, a forest of several hundred trees does not.

5.1.5 Performance

Table 5.1 presents results for the XGBoost models tested. Regularization weights, learning rate, and per-tree column dropout rates are set using cross-validation search for each model.

Figure 5.2 shows the accuracy of those models trained on the complete feature set, with max tree depth of 6, 15, or 30 nodes and total forest size of 150, 300, or 450 trees. Performance differences among the models are limited, well within the margin of error.

These models do show slight improvement in accuracy compared to the

Tree Count	Max Depth	Data Subset	Avg Acc	Std Dev
150	3	1	80.13%	1.38%
150	6	1	80.30%	1.70%
150	15	1	80.24%	1.56%
150	6	0.6	80.29%	1.20%
150	15	0.6	80.49%	1.11%
150	30	0.6	80.53%	1.63%
300	6	0.6	80.37%	1.57%
300	15	0.6	80.49%	1.42%
300	30	0.6	80.67%	1.74%
450	6	0.6	80.60%	1.58%
450	15	0.6	**80.81**%	1.62%
450	30	0.6	80.76%	1.52%
300	6	0.6	80.55%	1.49%
300	15	0.6	80.63%	1.42%
300	30	0.6	80.61%	1.49%
450	6	0.6	80.57%	1.34%
450	15	0.6	80.49%	1.35%
450	30	0.6	80.68%	1.47%

Table 5.1: Performance of XGBoost classifier models on the Vipur Training Set.

logistic regression models. Sample dropout improves generalization accuracy. Accuracy tends to improve with increased maximum tree depth, although variability of the results increases somewhat faster. Increasing the total number of trees in the model improves accuracy, but larger forests of deep trees (max 30 nodes) show lower estimated generalization accuracy than smaller trees at this forest size, possibly due to overfitting.

The models in the top portion of table 5.1 were trained on the complete feature set. Those below the line were trained on the reduced feature set, often leading to a slight (0.15%) improvement in accuracy. These trees are trained on only 27 features, so few trees are likely to reach the maximum depth of 30.

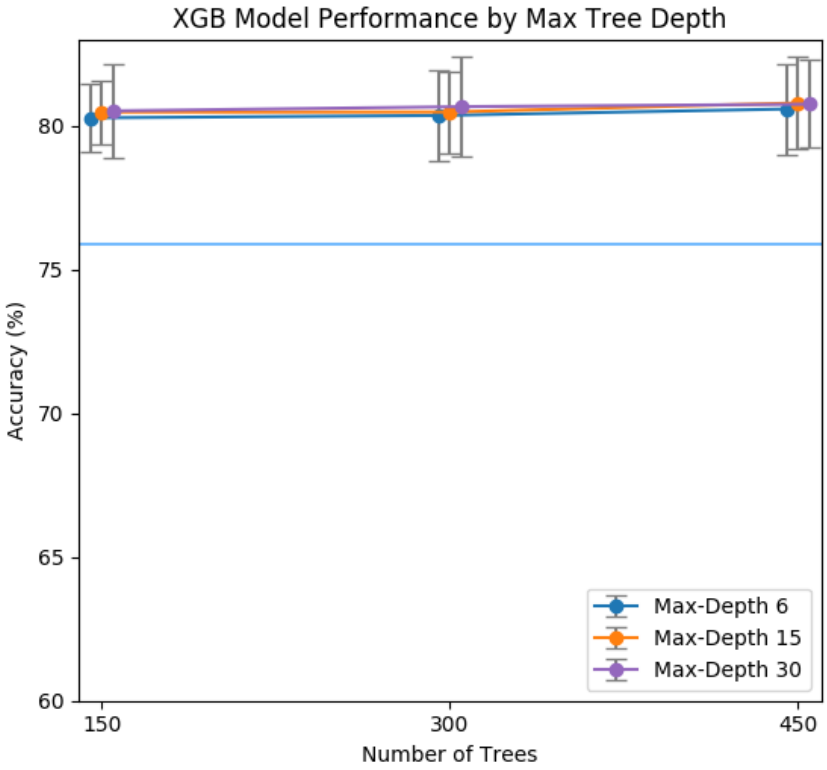

Figure 5.2: Accuracy (± one standard deviation) of XGBoost models trained on complete feature set, with 40% row data dropout per tree and maximum tree depth of 6, 15, or 30 nodes, against total number of trees in the forest. Blue line indicates accuracy of decision stump baseline.

Models trained on the reduced feature set are more accurate than their counterparts at 300 trees, but do not benefit from larger forests. This is consistent with the suggestion that many features are not strongly predictive—and models improve by not considering them—but that some marginal features do provide a slight amount of additional information in edge cases.

5.2 Neural Network Classifiers

Neural networks are general function approximators. Given enough training data, an appropriately structured neural network can theoretically approximate any finitely discontinuous computable function.[27] In practice, discovering the correct network structure and acquiring sufficient training data are nontrivial; complex tasks require extensive engineering of novel architectures, which are an area of tremendous ongoing research.

However, learning a classifier from the current VIPUR feature set is not expected to require elaborate or novel architectures. The new VIPUR framework can accommodate classifiers of arbitrary complexity, but the present investigation considers only fully connected networks with one to two hidden layers and a single output node.

5.2.1 Formulation of Neural Networks

Neural networks consist of many interconnected artificial neurons, as illustrated in figure 5.3. Each neuron has several inputs and a single output, which feeds either to a terminal node or to one or more additional neurons.

The neuron's output is determined by an *activation function*[9] whose input is the weighted sum of the inputs (and, optionally, some retained value of

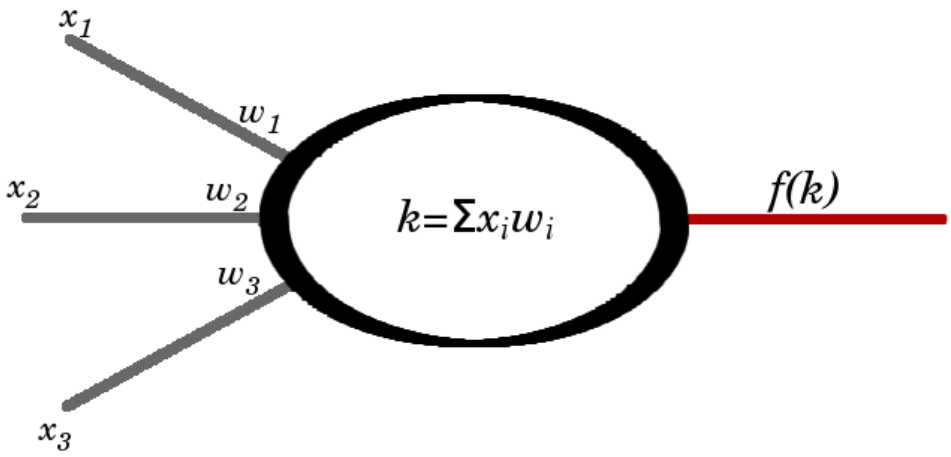

Figure 5.3: Image of a single artificial neuron. Each input value x_i has its own weight w_i, which is learned during training. The neuron computes the dot product of the input vector $\vec{x}$ and weight vector $\vec{w}$ and applies an activation function $f(\cdot)$ to the resulting scalar value, which is passed on to the next neuron.

the neuron's previous state).[27] Popular activation functions include:

$$\begin{aligned}
\text{Linear:} \quad & f(\vec{x}, \vec{w}) = \vec{w}^T \vec{x} \\
\text{Step:} \quad & f(\vec{x}, \vec{w}) = \begin{cases} 0 & \vec{w}^T \vec{x} \leq k \\ 1 & \vec{w}^T \vec{x} > k \end{cases} \\
\text{Sigmoid:} \quad & f(\vec{x}, \vec{w}) = \frac{1}{1 + \exp(-\vec{w}^T \vec{x})} \\
\text{ReLU:} \quad & f(\vec{x}, \vec{w}) = \max(0, \vec{w}^T \vec{x})
\end{aligned}$$

Nonlinearity arises from the non-linear activation functions (ReLU being the most commonly used function in practice) and the complexity of the net-

works. Because the neurons are connected in layers, the nonlinearity of individual activation functions combines to allow approximation of arbitrarily complex functions. Moreover, because neurons are typically connected to many other neurons, the network can capture joint effects among an arbitrary number of input features.

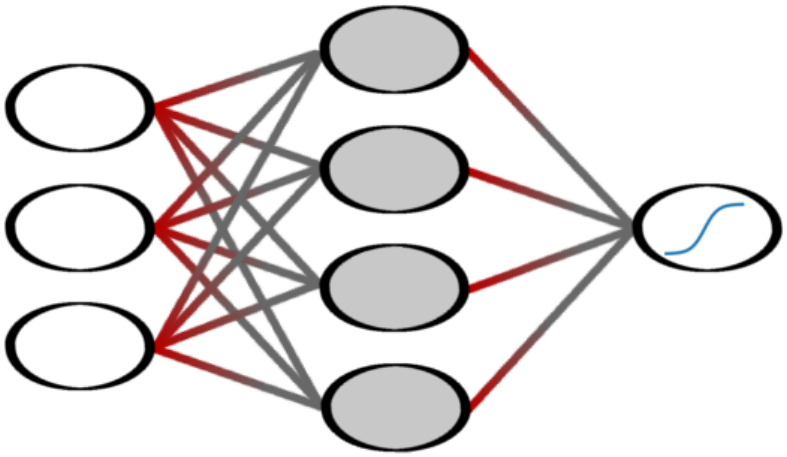

Figure 5.4: Image of a simple neural network with a three-dimensional input vector. The input layer is represented as three nodes, fully connected to the four nodes in the single hidden layer (nodes in grey). The hidden layer nodes connect to a single output neuron which applies a sigmoid function (hidden nodes are typically ReLU). The gradient along the neuron connections highlights that the output of one node becomes the input of the next layer's nodes.

Figure 5.4 illustrates a single-hidden-layer fully-connected network. A network is "fully connected" if every neuron in each layer receives input from every neuron in the previous layer; the "hidden layer" is so called because it does not produce output which is directly visible to the user. Networks of this type (although with many more nodes in the hidden layer) are the only network

topologies explored in the present work, although the new VIPUR code can accept any network design that matches the interface.

5.2.2 Training of Neural Networks

The weights for each neuron connection are the trainable parameters for the network. These weights are trained through a process known as *backpropagation*. We apply a loss function to the output layer and the known correct response, then compute the gradient of this loss function with respect to each neuron's weights. Since the input of each layer is the output of the layer that came before it, we can work backwards through the layers, finding the gradient of the loss with respect to the prior layer by use of the chain rule.[9][16] Finally, we update the weights by subtracting the gradient times a hyperparametric learning rate factor.

The gradient may be determined, and the weights updated, based on the average loss over the entire set of training samples (*batch gradient descent*), or over a fixed number of samples (*minibatch gradient descent*), or even after every training sample (*stochastic gradient descent*).[16] The choice of update frequency is also a hyperparameter, with smaller minibatch sizes allowing for more frequent training but more risk of idiosyncratic updates due to outlier samples.

5.2.3 Fitness to Purpose

While neural networks are capable of learning any function, their most prominent applications have been in fields where the input includes some minimally

processed form of raw data. For instance, image processing networks look directly at pixel representations and learn their own extractors to recognize shape features.[7] Modern natural language processing networks often consider words or vector embeddings of words, alongside (or sometimes in place of) the traditional extracted features such as part-of-speech assignment and semantic tree parsing that characterized earlier approaches to natural language understanding.[18]

Applying a neural network model to an extensively feature-engineered data set—such as the output of a Rosetta run—does not leverage the greatest strengths of the paradigm. Networks excel at feature discovery: presenting engineered features reduces the scope of this activity. Moreover, it is notoriously difficult to provide a meaningful explanation of why a network arrived at a particular conclusion;[36] the logistic regression model still has an advantage in this area.

However, neural network models offer the potential to incorporate new, raw-data features in future work. To the extent that there are meaningful complex interactions between the features in the VTS, the neural network is best positioned to discover them, since neural network models explicitly combine the elements of the input feature set.

5.2.4 Performance

The neural network models considered in the present work are simple ones, featuring one or two fully connected hidden layers with ReLU activation, and a single-node output layer with sigmoid function activation, whose output indi-

cates a deleterious/neutral probability prediction.

Shape	Dropout	Mean Epochs	Avg Acc	Std Dev
30	0.3	50	70.51%	2.83%
70	0.3	50	74.62%	1.98%
150	0.3	50	77.31%	1.56%
200	0.3	50	77.55%	1.44%
300	0.3	50	78.64%	1.97%
500	0.3	50	78.82%	1.52%
750	0.3	50	79.25%	1.66%
1500	0.3	50	79.53%	1.61%
2000	0.3	50	79.52%	1.61%
2500	0.3	50	**79.88%**	1.48%
3000	0.3	50	79.59%	1.63%
3500	0.3	50	79.75%	1.76%
1500	0.5	50	79.28%	1.56%
1500	None	50	**80.02%**	1.84%
1500	0.3	50	80.24%	1.77%
1500	0.5	50	80.30%	1.48%
3000	0.3	50	**80.47%**	1.81%
3000	None	50	80.13%	1.89%

Table 5.2: Accuracy of Neural Network classifiers with a single fully-connected hidden layer. Performance cited for final epoch of training. Models in the first two segments of the table were trained over the complete feature set. Models in the bottom section of the table were trained with the post-selection feature set. "Shape" refers to the number of neurons in the hidden layer.

Table 5.2 shows the results from the neural network structures tested. "Shape" indicates the number of nodes in each hidden layer; "dropout" indicates the proportion of nodes from the previous layer which are randomly turned off during training (to reduce overfitting); "epochs" indicates the total number of complete passes through the training data carried out over the course of training. All models are trained using the Adagrad optimizer.

Learning rate and minibatch size are set by cross-validation. There is

Shape	Dropout	Epochs	Avg Acc	Std Dev
100; 100	0.3; 0.3	50	73.70%	1.65%
150; 150	0.3; 0.3	50	74.47%	1.98%
250; 250	0.3; 0.3	50	76.63%	1.76%

Table 5.3: Accuracy of Neural Network classifiers with two fully-connected hidden layers. Performance cited for final epoch of training. These models underperform single-layer models of equivalent layer size or total neuron count. "Shape" indicates the number of neurons in the first and second hidden layers.

considerable variation in the optimal values for these variables, consistent with the non-uniformity of the VTS.

Figure 5.5 shows that performance increases with network complexity. However, the benefit of using a broader hidden layer starts to taper off rapidly once the hidden layer contains at least 1000 nodes. Values above 4000 were not tested: while there is little sign of overfitting in the results presented, overfit becomes a major concern once the number of hidden nodes approaches the order of the amount of training data, as each neuron can memorize one of the input values.

It is expected that complex models, given enough data, will begin to overfit if trained too long. Ordinary best practice is to monitor the performance on the validation set and use early stopping[19] to stop training the model once signs of overfit become evident. However, for the present work, early stopping criteria showed a very high variance in training—sometimes as much as 20 epochs (of 50)—which led to variation in final model performance as a result, particularly for the largest networks tested. Thus, all models presented in this work were trained to 50 epochs. Figure 5.6 shows the average validation loss for the models trained with early stopping; it indicates little concern about overfit.

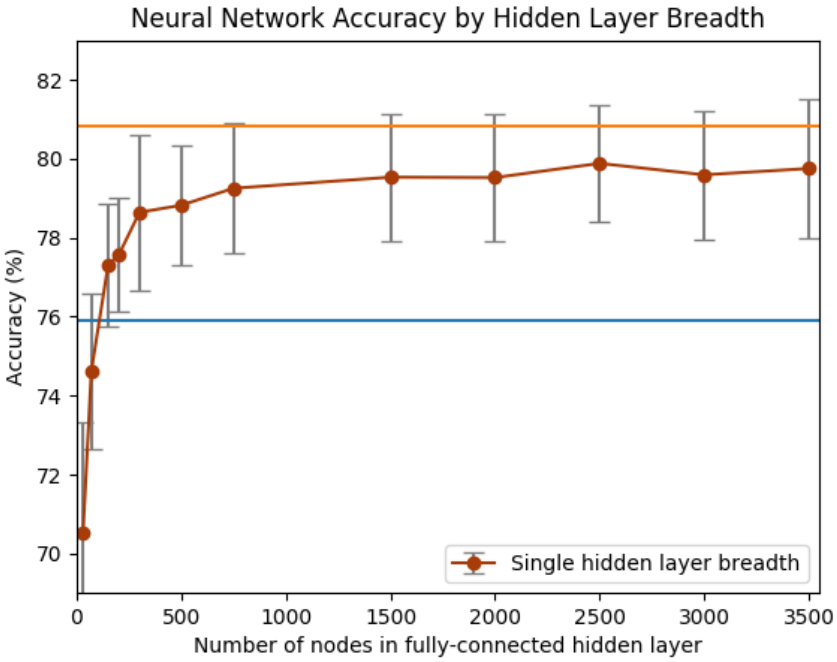

Figure 5.5: Best accuracy achieved by neural network models against the number of nodes in a single fully-connected hidden layer. Blue line indicates decision stump baseline; orange line represents average best accuracy achieved by the top-performing XGBoost model. Performance initially increases rapidly with added complexity, but later improvement is greatly attenuated.

Moreover, as figure 5.7 illustrates, all models tested showed convergence after 20-30 epochs.

Figure 5.6 deserves some further comment. Even on the validation set, all models reach a steady-state loss level (for most models, after only one or a handful of epochs). The models are clearly learning, as demonstrated by the visible convergence path for the larger network models. However, we do not see the subsequent increase in validation loss that would be expected from an overfit model. The pattern most closely resembles the phenomenon of vanishing gradients, in that the network has ceased to meaningfully update in either di-

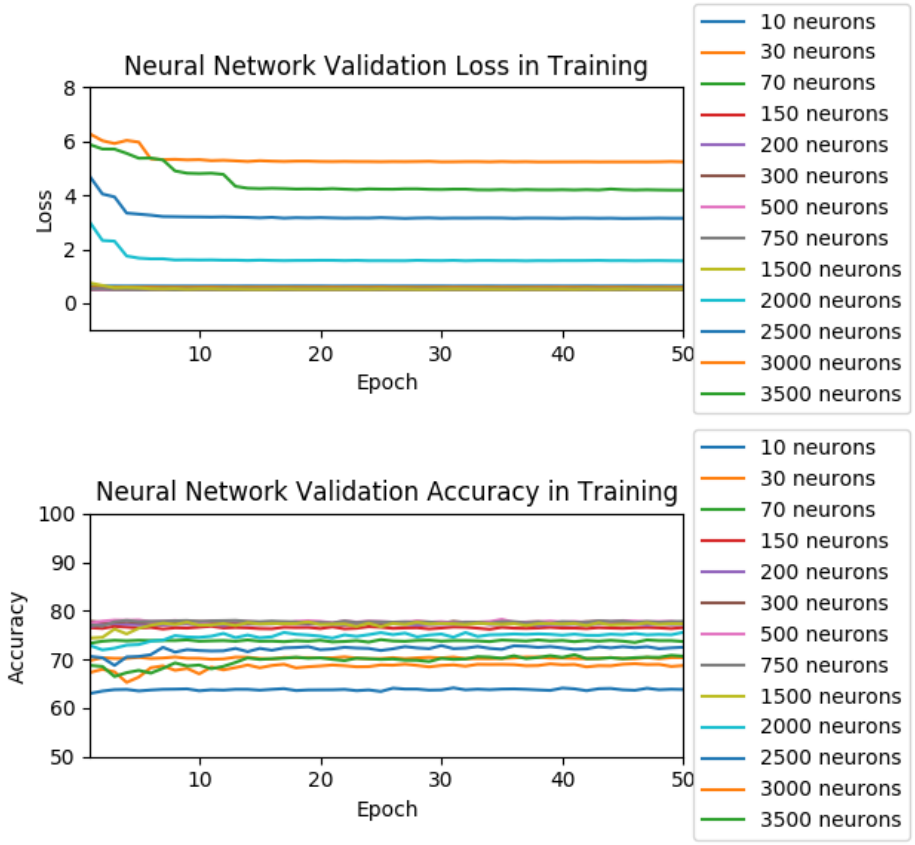

Figure 5.6: Performance of networks on validation set during training. Top plot shows loss, while bottom plot shows accuracy. Both demonstrate no sign of compromised generalization performance due to overtraining. Note that the most complex networks do require somewhat more epochs to converge.

rection. This would also be consistent with the lower performance of two-layer networks compared to the single-layer networks shown in the figure.

The present models were trained using the Binary Cross-Entropy loss function. It is possible that an updated loss function, such as losses approximating the Earth-Mover Distance,[4] would lead to ongoing training. Further research is needed.

A surprising amount of model complexity is needed for these classifiers

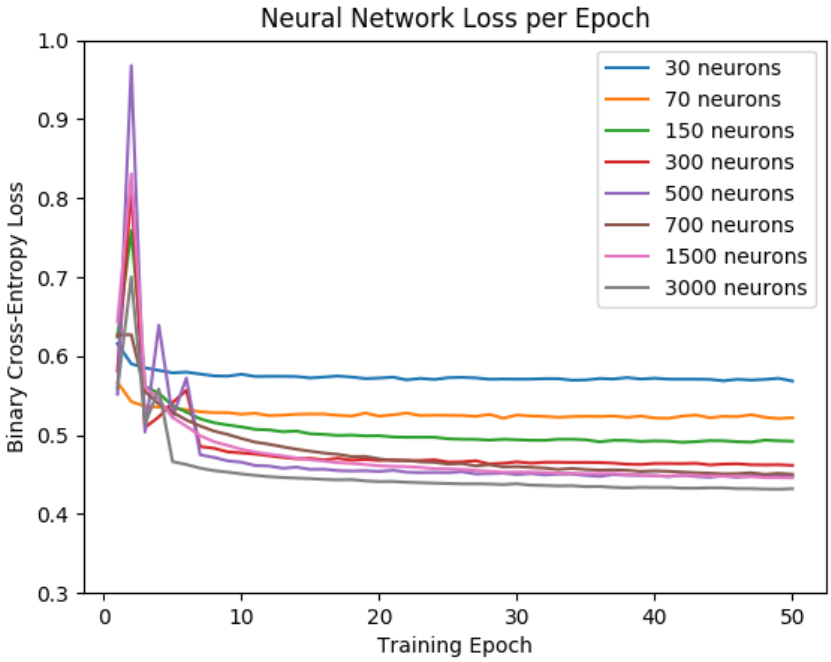

Figure 5.7: Plot of neural network training loss against epoch for selected networks, demonstrating rapid model convergence in all cases. All networks consist of a single fully-connected hidden layer and single output layer, with 30% dropout between the two.

to improve upon the decision stump baseline; over 2000 hidden-layer nodes are required to improve upon the logistic regression classifier models.

Somewhat surprisingly, higher dropout leads to reduced performance; and turning dropout off entirely sometimes yields improved performance. This suggests, again, that overfitting is less of an issue in these models than the disappearance of loss signal. Of course, it may also be a function of the complex sample space.

6

Results

As the results discussed throughout this work demonstrate, all methods explored seem to converge to an accuracy between 80-81%. This chapter compares the best performers from each classifier methodology and discusses possible reasons for this convergence.

Model	Avg Acc	Std Dev
Decision Stump (baseline)	75.96%	2.10%
Logistic Regression, L1, selected features	80.36%	1.76%
SVM, RBF kernel, selected features	80.42%	1.27%
XGBoost, 450 trees, max depth 15, all features	80.81%	1.62%
Neural Network, 1 hidden layer of 3000 neurons, 30% dropout, selected features	80.47%	1.81%

Table 6.1: Comparison of highest-accuracy methods from each methodology, listing average accuracy over five cross-validation folds, and standard deviation.

Table 6.1 compares the results of the highest-accuracy classifiers from each model type. All best performers converge around the 80% accuracy level, and all results are within one standard deviation of each other.

The similarity of the top performers is further illustrated in figure 6.1, which shows each top performer's average accuracy ± one standard deviation. The blue horizontal line indicates the decision stump baseline, and the red line is the label bias. The left plot shows the range from from 60% to 85%, to make standard errors visible. The right plot shows the full range, for scale.

All classifier types except for the baselines and SVM support probability predictions. Figure 6.2 shows the AUPR and ROC results of each top performer on the same axes. Results are nearly indistinguishable on this metric as well, with all curves falling within much less than one standard deviation of each other.

One area in which these classifiers are distinct is in their tendency toward false negatives and false positives. Table 6.2 illustrates these trends. A

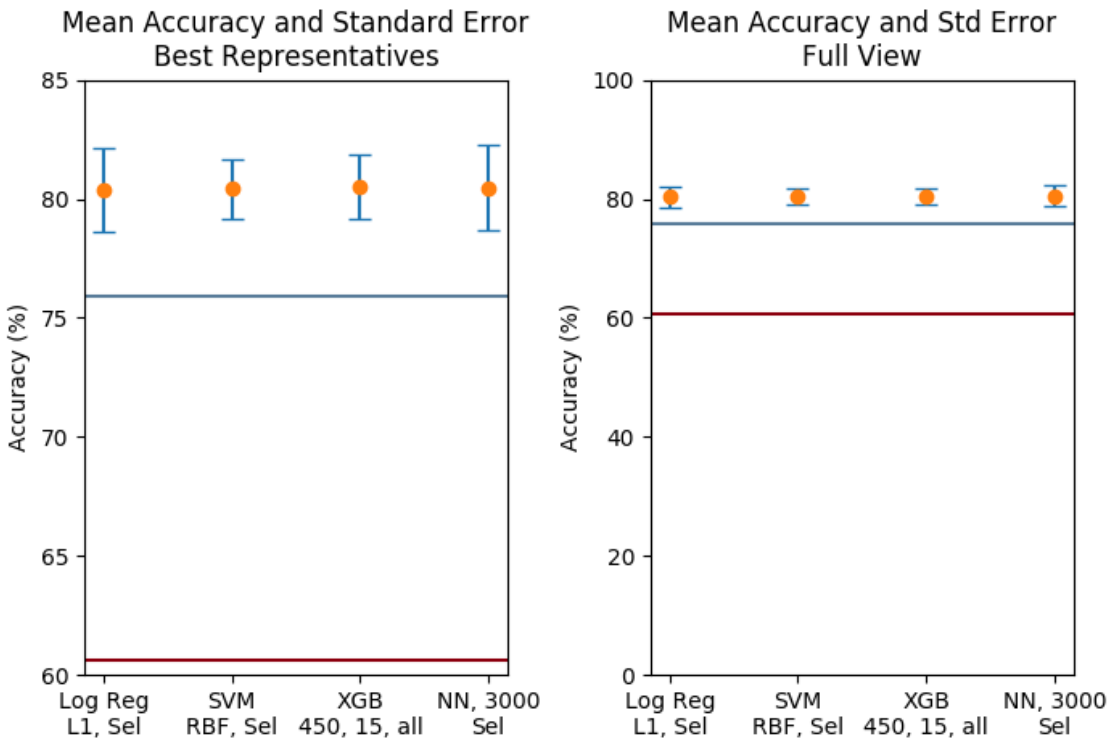

Figure 6.1: Mean accuracy ($\pm$ one standard deviation) for best-performing examples of each classifier type. Classifiers are logistic regression under L1 penalty, support vector machines with a radial basis function, XGBoost with 450 trees of maximum depth 15, and neural networks with a single 3000-node fully connected hidden layer. All but the XGBoost models were trained on the feature subset.

collection of classifiers with roughly equivalent performance, but which make different mistakes, suggests the possibility of training a stacked learner or meta-learner which could outperform each of the individual classifiers. As discussed in chapter 2, the current reimplementation of VIPUR includes functionality to support this approach by supplementing the data set with predictions of other classifiers; however, this approach has not yet been explored.

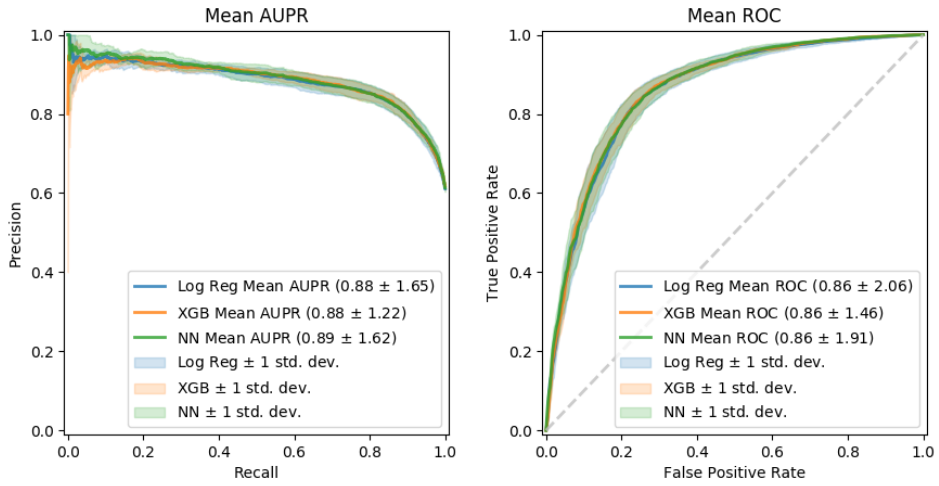

Figure 6.2: AUPR and ROC curves for best-representative classifiers from all supported classifier types.

6.2 Feature Selection Improves Accuracy

Operating on a selected subset of features improves accuracy—very slightly—for three of the models studied. Table 6.3 lays out the improvement in results due to limiting the feature subset, for the best-performing classifiers of each type.

The original VIPUR work[5] showed that restricting the feature set resulted in improvement in logistic regression models. SVM models also benefit from restricting the feature set. More modern methods, such as neural network models and XGBoost, are expected to benefit less from pre-training feature selection, because selection and feature dropout are incorporated into their design. Restricting the feature set does in fact reduce the accuracy of the XGBoost models, while it (surprisingly) improves the performance of the neural network.

In the neural network case, reducing the size of the input layer vastly reduces the number of weights that must be learned, which improves the effective-

Classifier	FP Count	FN Count	+/− Ratio
Decision Stump (baseline)	164.4	291.2	0.56
Logistic Regression, L1 Penalty, selected features	207.4	164.8	1.26
SVM (RBF), selected features	174.6	196.6	0.89
XGBoost, 450 trees, max depth 15, all features	192.0	171.8	1.12
Neural network, 1 fully connected hidden layer of 3000 neurons, selected features	213.0	157.2	1.35

Table 6.2: Bias in false positives and false negatives for the top performers of each classifier category. All non-baseline models show bias toward false positives, except for the RBF SVM classifier and the decision stump.

Classifier	Accuracy (All)	Accuracy (Selection)	Improvement
Log Reg L1	79.85%	80.36%	0.51%
SVM (RBF)	79.41%	80.42%	1.01%
XGB, 450 trees, depth 15	80.81%	80.49%	-0.32%
Neural Network, 3000 neurons	79.59%	80.47%	0.88%

Table 6.3: Improvement of performance due to feature selection.

ness of network training and the power of the network relative to the inputs. In the case of XGBoost, feature selection reduces accuracy for larger forests (450 trees), but improves it for 300-tree forests. This is consistent with large forests making use of some of the more marginal or situational features (see table 3.1) toward the end of forest training.

6.3 No Approach Significantly Outperforms

The above results show, at best, only marginal improvement compared to the original VIPUR. Modern, highly flexible machine learning frameworks do not

noticeably improve upon results from a logistic regression classifier. The results discussed in section 5.2.4 and in figure 5.5 are of particular interest, showing a quasi-asymptotic relationship between increased network power and increased performance.

There is an open question of why these neural network models stop training. If the reason for this can be discovered and overcome, there is still hope that this approach might produce more substantial improvement over the other methods. Stacking solutions also remain to be tried.

Overall, though, the limited improvement from the methods discussed above suggests that further development of VIPUR will be dependent upon improved data and improved feature discovery. The final chapter explores some of the possibilities for meeting these needs.

7

Conclusion and Future Work

The original VIPUR work was quite successful: its 80-81% accuracy, with even more accurate results for the highest confidence levels, exceeded competing methods at the time of its release. VIPUR continues to provide useful information to guide lab research. Unfortunately, the present work suggests that apply-

ing more advanced machine learning techniques to the existing data will result in at best incremental improvements over the original VIPUR.

The VIPUR project depends upon a set of strong assumptions: that there exist features which are characteristic of all proteins generally, across species and kingdoms; that these features can be extracted by known methods; that these features can be isolated from individual protein structures independently of the protein's natural environment and context; and that these features can be consistently interpreted in a way that leads to accurate deleteriousness predictions. The first and fourth of these assumptions are foundational; but the middle two offer considerable room for improvement. We can expand the methods used to extract features—possibly by incorporating structural features not directly extracted using Rosetta—and incorporate protein interaction data to avoid looking at proteins in isolation.

7.1 Data is the Limiting Factor

What is needed is not better methods, but better data. This includes both better features, and more variant records.

Feature improvements can come from several sources. Improvements to the Rosetta scoring function in the years since the VIPUR release[2] may improve the predictive power of some structure features.

We also need to explore new sets of features that have stronger and more generally applicable predictive effects than the ones presented in the Vipur Training Set.

One promising feature is whether the mutation occurred in a functional region of the protein (defined as a region which comes into contact with an interaction partner). Work in the Bonneau Lab under Vladimir Gligorijevic has made significant progress in protein function prediction using neural networks. It is hoped that we can use salience mapping[23] to identify regions which are significant for the network's predictions. Predictive significance could act as a proxy for the significance of the protein domain to its biological interactions.

Conversely, since inherently disordered regions of proteins are already less rigidly structured than the rest of the protein,[15] mutations to those regions may be less sensitive to structural changes, and thus less sensitive to point mutations. Knowing whether a mutation falls into such a region could guide the decisions made by the rest of the classifier, if the classifier is sufficiently sophisticated to capture interactions between features.

Baugh's `aminochange` feature, which looks specifically at residue identities,[5] proved to be a strong predictor. Other features based on residue identity may be similarly effective. I would like to expand this feature to look at windows of residues on either side of the mutation. Julia Koehler Leman has suggested the use of PSSM over a residue window, which is particularly appealing given the demonstrated power of PSSM values at the point of mutation itself.

Beyond these summarized features, I would like to include residue identities of the mutation itself and of its neighbors as one-hot-encoded features in the input set. This will greatly increase the number of features, and capturing the complex interactions among these features is likely to require a neural-

network-based classifier. However, feature discovery is precisely the area where neural networks have outperformed across diverse fields in recent work. Neural network approaches to structure prediction problems have already been shown to be both powerful and fast.[3] Deleteriousness prediction should be no an exception.

Finally, more data is needed, from broader sources. The primary sources of the VTS—UniProt, the PDB, and HumDiv—have continued to grow in the years since the VTS was assembled. New databases have also been developed. In particular, data about protein-protein interactions is available, which is essential training information for some of the higher-order features discussed above. VIPUR will be substantially improved by collecting data more broadly.

Whatever new data and features are identified, the new VIPUR in the present work will accommodate them with minimal coding changes. As new data is acquired, the current VIPUR implementation will be able to incorporate them and integrate them seamlessly into both the classifiers investigated here, and any new ones we wish to test, and evaluate the entire solution on a rigorous and consistent basis.

A

Feature Selection Details

This appendix details the VIPUR features which were selected or rejected through feature selection. Full details of the biochemical meaning of each feature are available in Baugh 2016.[5]

Feature	L1 count	L2 count	Total
aminochange*	20	20	40
pssm_difference*	20	20	40
pssm_information_content*	20	20	40
pssm_native	0	20	20
pssm_variant*	20	20	40

Table A.1: List of VIPUR sequence features, with counts of the number of times the feature was selected under each methodology. Features marked with an asterisk appear in the original VIPUR selected feature set.

Feature	L1 count	L2 count	Total
ddg_fa_pair*	20	20	40
ddg_fa_rep*	20	20	40
ddg_fa_sol*	20	20	40
ddg_hbond_sc*	20	17	37
probe_accp*	20	20	40
quartile_dslf_cs_angQ3*	20	20	40
quartile_gdtmm3_3Q1*	20	20	40
quartile_gdtmm4_3Q3*	20	20	40
quartile_pro_closeQ2*	20	20	40
quartile_ramaQ3*	20	19	39
quartile_scoreQ1*	20	20	40

Table A.2: List of very commonly selected VIPUR structural features, with counts of the number of times the feature was selected under each methodology. Features marked with an asterisk appear in the original VIPUR selected feature set.

Feature	L1 count	L2 count	Total
ddg_dslf_ss_dst	9	9	18
ddg_hbond_bb_sc*	20	11	31
ddg_total_score*	20	9	29
maxsub2.0	9	18	27
quartile_allatom_rmsQ2	9	9	18
quartile_dslf_ca_dihQ1	9	20	29
quartile_dslf_ca_dihQ3	9	20	29
quartile_fa_pairQ1*	20	9	29
quartile_fa_solQ3	9	9	18
quartile_gdtmm7_4Q2	9	14	23
quartile_gdtmm7_4Q3	9	15	24
quartile_ramaQ1	9	14	23

Table A.3: List of 12 structural features which were selected by both models at least 9 times out of 20. These are considered moderately high-value.

Feature	L1 count	L2 count	Total
quartile_allatom_rmsQ3	9	5	14
quartile_gdtmm1_1Q2	8	1	9
quartile_gdtmmQ2	9	1	10
quartile_p_aa_ppQ3*	20	2	22

Table A.4: List of features which were selected by both selection methods, but generally rarely.

Appearances	Features
20	maxsub, quartile_fa_solQ1, quartile_hbond_bb_scQ3, quartile_hbond_scQ3, quartile_pro_closeQ1, quartile_pro_closeQ3
9	ddg_dslf_cs_ang, ddg_fa_atr, ddg_hbond_lr_bb, ddg_hbond_sr_bb, ddg_ref, quartile_dslf_cs_angQ1, quartile_dslf_ss_dstQ1, quartile_dslf_ss_dstQ3, quartile_fa_intra_repQ3, quartile_fa_repQ1, quartile_fa_repQ2, quartile_gdtmm2_2Q2, quartile_gdtmm2_2Q3, quartile_gdtmm3_3Q3, quartile_hbond_lr_bbQ1, quartile_hbond_scQ1, quartile_hbond_sr_bbQ1, quartile_hbond_sr_bbQ3, quartile_refQ1, quartile_refQ2, quartile_refQ3
8	quartile_scoreQ2
4	quartile_hbond_scQ2
1	quartile_gdtmm1_1Q1, quartile_scoreQ3

Table A.5: List of 31 structural features that were only selected by the L1 selection method.

The following features were never selected by either selection method:

ddg_dslf_ca_dih, ddg_dslf_ss_dih, ddg_fa_dun, ddg_p_aa_pp, ddg_pro_close, quartile_allatom_rmsQ1, quartile_dslf_ca_dihQ2, quartile_dslf_cs_angQ2, quartile_dslf_ss_dihQ1, quartile_dslf_ss_dihQ2*, quartile_dslf_ss_dihQ3, quartile_dslf_ss_dstQ2, quartile_fa_atrQ1, quartile_fa_atrQ2, quartile_fa_atrQ3, quartile_fa_dunQ1, quartile_fa_dunQ2, quartile_fa_dunQ3, quartile_fa_intra_repQ1, quartile_fa_intra_repQ2, quartile_fa_pairQ2, quartile_fa_pairQ3, quartile_fa_repQ3, quartile_fa_solQ2, quartile_gdtmm1_1Q3, quartile_gdtmm2_2Q1, quartile_gdtmm3_3Q2, quartile_gdtmm4_3Q1, quartile_gdtmm4_3Q2, quartile_gdtmm7_4Q1, quartile_gdtmmQ1, quartile_gdtmmQ3, quartile_hbond_bb_scQ1, quartile_hbond_bb_scQ2, quartile_hbond_lr_bbQ2, quartile_hbond_lr_bbQ3, quartile_hbond_sr_bbQ2, quartile_omegaQ1, quartile_omegaQ2, quartile_omegaQ3, quartile_p_aa_ppQ1, quartile_p_aa_ppQ2, quartile_ramaQ2

References

[1] Adzhubei, I. A., Schmidt, S., Peshkin, L., Ramensky, V. E., Gerasimova, A., Bork, P., Kondrashov, A. S., & Sunyaev, S. R. (2010). A method and server for predicting damaging missense mutations. *Nature Methods*, 7(4), 248–249.

[2] Alford, R., Leaver-Fay, A., Jeliazkov, J. R., O'Meara, M. J., DiMaio, F. P., Park, H., Shapovalov, M. V., Renfrew, P. D., Mulligan, V. K., Kappel, K., Labonte, J. W., Pacella, M. S., Bonneau, R., Bradley, P., Dunbrack, Jr., R. L., Das, R., Baker, D., Kuhlman, B., Kortemme, T., & Gray, J. J. (2017). The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *Journal of Chemical Theory and Computation*, 13, 3031–3048.

[3] AlQuraishi, M. (2019). End-to-End Differentiable Learning of Protein Structure. *Cell Systems*, 8.

[4] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. arXiv:1701.07875.

[5] Baugh, E. H. (2017). *Predicting the Effects of Protein Variants using Structural Modeling, Large-Scale Data Integration, and Machine Learning*. PhD thesis, New York University, New York, NY, USA.

[6] Baugh, E. H., Simmons-Edler, R., Müller, C. L., Alford, R. F., Volfovsky, N., Lash, A. E., & Bonneau, R. (2016). Robust classification of protein variation using structural modelling and large-scale data integration. *Nucleic Acids Research*, 44(6), 2501–2513.

[7] Bengio, Y. (2009). *Learning Deep Architectures for AI*. Now Foundations and Trends.

[8] Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., & Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Research*, 28(1), 235–242.

[9] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer.

[10] Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2).

[11] Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: architecture and applications. *BMC Bioinformatics*, 10(421).

[12] Chen, T. & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16 (pp. 785–794). New York, NY, USA: ACM.

[13] Choi, Y. & Chan, A. P. (2015). Provean web server: a tool to predict the functional effects of amino acid substitutions and indels. *Bioinformatics*, 31(16), 2745–2747.

[14] Cortes, C. & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

[15] Dunker, A. K., Silman, I., Uversky, V. N., & Sussman, J. L. (2008). Function and structure of inherently disordered proteins. *Current Opinion in Structural Biology*, 18(6), 756–764.

[16] Efron, B. & Hastie, T. (2016). *Computer Age Statistical Inference*. Cambridge University Press.

[17] Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119–139.

[18] Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

[19] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[20] Gorman, B. (2017). A kaggle master explains gradient boosting. Kaggle.com official blog. Accessed from http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/.

[21] Hocking, R. R. (1976). The Analysis and Selection of Variables in Linear Regression. *Biometrics*, 32(1), 1–49.

[22] Iba, W. & Langley, P. (1992). Induction of One-Level Decision Trees. In *Proceedings of the Ninth International Conferece on Machine Learning* Aberdeen, Scotland: Morgan Kaufmann.

[23] Itti, L., Koch, C., & Niebur, E. (1998). A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11), 1254–1259.

[24] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R.* New York, NY, USA: Springer.

[25] Jolliffe, I. T. (2002). *Principal Component Analysis.* New York, NY, USA: Springer, 2 edition.

[26] Kellogg, E. H., Leaver-Fay, A., & Baker, D. (2011). Role of conformational sampling in computing mutation-induced changes in protein structure and stability. *Proteins: Structure, Function, and Bioinformatics*, 79(3), 830–838.

[27] Kriesel, D. (2007). A brief introduction to neural networks. available at http://www.dkriesel.com.

[28] Leaver-Fay, A., Tyka, M., Lewis, S. M., Lange, O. F., Thompson, J., Jacak, R., Kaufman, K., Renfrew, P. D., Smith, C. A., Sheffler, W., Davis, I. W., Cooper, S., Treuille, A., Mandell, D. J., Richter, F., Ban, Y.-E. A., Fleishman, S. J., Corn, J. E., Kim, D. E., Lyskov, S., Berrondo, M., Mentzer, S., Popović, Z., Havranek, J. J., Karanicolas, J., Das, R., Meiler, J., Kortemme, T., Gray, J. J., Kuhlman, B., Baker, D., & Bradley, P. (2011). Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods in Enzymology*, 487, 545–574.

[29] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of Machine Learning.* Cambridge, MA, USA: MIT Press, 1 edition.

[30] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press.

[31] Ng, A. Y. (1998). On Feature Selection: Learning with Exponentially many Irrelevant Features as Training Examples. In *Proceedings of the 15th International Conference on Machine Learning*: Morgan Kauffman.

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

[33] Pieper, U., Webb, B. M., Dong, G. Q., Schneidman-Duhovny, D., Fan, H., Kim, S. J., Khuri, N., Spill, Y. G., Weinkam, P., Hammel, M., Tainer, J. A., Nilges, M., & Sali, A. (2014). MODBASE, a database of annotated comparative protein structure models and associated resources. *Nucleic Acids Research*, 42, D336–346.

[34] Reece, J. B., Wasserman, S. A., Urry, L. A., Minorsky, P. V., Cain, M. L., & Jackson, R. B. (2014). *Campbell Biology, 10th Ed.* Boston, MA, USA: Pearson, 10 edition.

[35] Russell, S. & Norvig, P. (2010). *AI: A Modern Approach, 3rd Ed.* Upper Saddle River, NJ, USA: Pearson, 3 edition.

[36] Samek, W., Wiegand, T., & Müller, K.-R. (2017). Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. arXiv:1708.08296.

[37] Shalev-Shwartz, S. & Ben-David, S. (2014). *Understanding Machine Learning from Theory to Algorithms*. New York, NY, USA: Cambridge University Press.

[38] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27, 623–656.

[39] UniProt Consortium (2007). The Universal Protein Resource (UniProt). *Nucleic Acids Research*, 35, D193–197.

[40] Waterhouse, A., Bertoni, M., Bienert, S., Studer, G., Tauriello, G., Gumienny, R., Heer, F. T., de Beer, T. A. P., Rempfer, C., Bordoli, L., Lepore, R., & Schwede, T. (2018). SWISS-MODEL: homology modelling of protein structures and complexes. *Nucleic Acids Research*, 46(W1), W296–W303.

[41] Word, J. M., C.Lovell, S., LaBeana, T. H., C.Taylor, H., E.Zalisa, M., K.Presley, B., S.Richardson, J., & C.Richardson, D. (1999). Visualizing and quantifying molecular goodness-of-fit: small-probe contact dots with explicit hydrogen atoms. *Journal of Molecular Biology*, 285(4), 1711–1733.

[42] Zou, H. & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 67, 301–320.