Machine Learning on Biochemical Small Datasets: Strategies for Pursuing Predictive Analyses of Human Voltage Gated Sodium Ion Channel (hNa_Vs) Inhibitors

Vamsi Varanasi (vamsijv), Jay Liu (jliu99), JC Liang (liangjc)
Stanford University, Stanford, California 94305, USA
(Dated: June 13, 2019)

INTRODUCTION

Voltage-gated sodium channels (Na_Vs), which modulate membrane permeability to sodium ions, facilitate crucial intercellular communication. Their dysfunction is implicated in a variety of diseases, including epilepsy, cardiac arrhythmia, and chronic pain. So far, nine human isoforms of Na_Vs (hNa_Vs 1.1–1.9) have been distinguished, but as we currently lack the ability to image Na_Vs isoforms in live cells, many channelopathy studies have been tabled until a proper tracking agent can be developed. Luckily, some eukaryotic Na_Vs are susceptible to a class of highly potent mono- and bisguanidinium neurotoxins, including (+)-saxitoxin (STX) and tetrodotoxin (TTX), which represent scaffolds upon which chemists have attempted to synthesize small molecule Na_Vs imaging agents, mutant-selective ligands, and isoform-specific blockers.

Unfortunately, not all Na_Vs are susceptible to toxin inhibition; for instance, Na_Vs 1.5, 1.8, and 1.9 are TTX-resistant. Attempts to intelligently design around these limitations have been frustrated by lack of information pertaining to the binding pose of toxins in certain hNa_Vs (due to the difficulty of acquiring crystal structures of transmembrane receptors and the imprecision of homology modeling). Traditionally, binding models have been proposed from clues garnered through mutant cycle analysis of Na_Vs targets and modified toxin derivatives, but this is a labor-intensive endeavor in organic synthesis and site-directed mutagenesis.

In light of these difficulties, we chose to investigate whether machine learning techniques could be leveraged to better predict the binding affinity of candidate ligands to protein targets like human sodium channels. Although existing experimental binding affinity data on neurotoxin derivatives and their protein targets constitute a very small and limited dataset, we were able to find ways to supplement the existing data by training models on the Protein Data Bank (PDB), a large public dataset that contains binding affinity data derived from high-resolution cryo-EM and crystal structures of a wide range of protein target-ligand pairs. Using Random Forest, SVM, and KRR regression models trained on featurized and labeled data from PDB, we were able to set the foundation for evaluating this model's ability to make accurate, computationally affordable predictions in a specific biochemical subspace (namely, that of binding between the sodium channel and derivatives of STX).

RELATED WORK

Justin Du Bois is a faculty member in Stanford's Department of Chemistry that has done significant work on synthesizing toxin derivatives in order to assess their potency towards various Na_Vs isoforms and mutants. Thus far, his group has been able to identify key functional groups (chemical features) on saxitoxin that likely contribute to the toxin's high binding affinity, and they are in the process of selectively functionalizing (adding chemical features to) carbons on saxitoxin in order to 1) better understand the binding dynamics of STX to sodium channels and 2) develop STX derivatives that are selective for only one of two human sodium channel isoforms. [1] However, this iterative experimental synthesis process is laborious and intricate, motivating machine learning approaches to simulate toxin-Na_V interactions.

Unfortunately, available data on sodium channel/STX binding is rare and disproportionately weighted towards 1) STX derivatives with modifications on very well-studied carbon atoms and 2) STX derivatives that bind rat sodium channel 1.4, the protein most commonly used to determine ligand binding affinity. At first glance, this problem does not seem amendable to a traditional machine learning approach, a circumstance that confronts many researchers confronted with a challenging problem, limited information, and a small biochemical dataset.

Approaches to deal with this problem fall into two classes. The first is to leverage existing optimization techniques to run machine learning algorithms on the small dataset. Some of the techniques include multiple runs for model development, surrogate data analysis for model validation, and k-fold cross validation. These methods were successfully leveraged by Shaikhina et al. to design a neural network and decision tree for predicting the likelihood of antibody-mediated kidney transplant rejection from only 35 bone specimens and 80 kidney transplants. [2]

While the results of this study are promising, the techniques are not directly applicable to our project, as it would not have been feasible under our time constraints to gather, filter, and sort the data from the Du Bois lab prior to developing a machine learning model.

An alternative technique is to supplement a small dataset with a much larger dataset of protein-ligand interactions. The interatom interactions that govern binding between proteins and their small molecules are common across most protein-ligand binding pairs. Thus,

even though the larger dataset may not contain information specific to saxitoxin and sodium channels, machine learning models may still be able to learn enough about atomic interactions to predict accurate binding affinities of candidate STX derivatives to their protein targets. Such large target/ligand datasets exist and have been successfully curated by multiple sources. [3]

This approach has been utilized successfully by Pande et al. to quantitatively predict the binding affinities of inhibitors to human -secretase 1 (BACE-1). [4] Pande's group synthesized most of their project code into a library called DeepChem, which provides tools for featurizing protein-ligand interactions [5]. Using DeepChem, we attempted to extend Pande's method to our problem subspace.

DATASET AND FEATURES

The Protein Data Bank (PDB) (https://www.rcsb.org/) is a repository of over 152,000 known protein structures. PDBBind (http://www.pdbbind.org.cn/) is an expertly curated database of experimentally measured binding affinities from biomolecular complexes published to PDB [6]. As of 2018, it contains over 19,000 entries, of which approximately 16,000 are protein-ligand interactions. PDBBind is further divided into higher quality "core" (n=189) and "refined" (n=3,568) sets. We used the PDBBind dataset circa 2015, when it contained 11,303 high-resolution static images of proteins bound to ligands, to conduct our studies.

To characterize these interactions, we used the computational chemistry suite furnished by DeepChem [5] to featurize protein ligand interactions in PDBBind. DeepChem's featurizer translates the protein and its ligand into two (N, 3) arrays, where the rows represent each atom in the structure and the columns represent the atom coordinates in angstroms (with the origin is set as the center of the protein). Next, it identifies concave pockets on the protein to find potential binding sites, and passes the regions of interest to the RDKitGridFeaturizer. The grid featurizer voxelizes the regions of the interest, identifies atoms in close proximity based their coordinate arrays, and extracts information about interatomic and intermolecular interactions (hydrogen bonding, electrostatics, and pi-stacking). Finally, it computes a circular topological fingerprint of the molecular environment and concatenates the result with information from the grid featurizer on relevant intermolecular forces.

The result is an (N, 2052) array for the featurized dataset, where N is the number of datapoints, each of which is described by $d = 2052$ features. The corresponding labels are stored in an (N, 1) array, where the value of the label is the experimentally reported binding affinity of the protein-ligand complex, given as $pK_d - pK_i$.

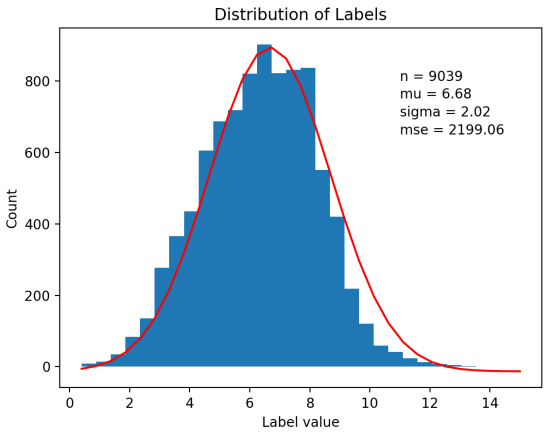

FIG. 1: **Binned Labels with Fitted Normal Distribution**

To account for training time, we worked with the core and refined subsets of PDBBind, with 189 and 3,568 datapoints, respectively. We visualized the labels for the full PDBBind dataset (11,000 data points) by plotting them as a histogram in Figure 1. We see that even with a larger dataset, the labels are slightly skewed from a fitted Gaussian. Finally, the core and refined data was randomly sorted into train/validation/test sets using an 80/10/10 split.

METHODS

We trained all of our models using Python packages from Scikit-learn. The predictive capacity and accuracy of our models was judged by qualitatively evaluating the parity plots generated from our training, validation, and testing sets, and quantitatively by comparing Pearson R^2 values for the predictions. To achieve reasonable training times for our SVM models, we conducted our training in Stanford's *rice* shared computing cluster, running on one 4GB core of a CPU for a maximum of 40 hours.

Ultimately, we analyzed our data using 4 types of models: Random Forest Regression, Support Vector Machine Models, and Kernel Ridge Regression. We also performed Principle Component Analysis (PCA) to combat perceived overfitting in our data.

A. Linear Regression

Linear regression models attempt to fit a given dataset to an equation of the form:

$$h_\theta(x) = \theta^T x$$

The parameters are derived by minimizing the squared error of the model, either by finding the global maxima of the loss function via gradient descent, or by solving

the normal equation:

$$\theta = \left(X^T X\right)^{-1} X^T y$$

B. Random Forest (RF)

We sampled, with replacement, data points to train random forests consisting of n decision trees, where $n = 10, 20, 50, 100, 200, 500$, and 1000 trees. Our predictions were generated by averaging the predictions made from n individual trees to reduce the penalty of overfitting by any particular tree.

$$\hat{y} = \frac{1}{n}\sum_{b=1}^{n} f_b(x)$$

Decision trees are generated by dividing a dataset into discrete regions within its feature space, then making a prediction by averaging all training labels in the region. The regions are refined by minimizing the residual sum of squares for datapoints within the region:

$$\sum_{j=1}^{J}\sum_{i\in R_j}^{m}\left(y^{(i)} - \hat{y}^{R_j}\right)^2$$

C. Support Vector Machines (SVM)

Kernelized SVMs map input data into a high-dimensional feature space using a non-linear kernel, then constructs a set of hyperplanes that maximizes the functional margins of the training data:

$$\begin{aligned}&\max_{\gamma,w,b}\ \gamma\\ &\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq \gamma, \quad i = 1,\ldots,m\\ &\qquad \|w\| = 1\end{aligned}$$

When cast back into the feature space of the original data, the hyperplane becomes a model capable of discerning nonlinear patterns in the dataset. Predictions are made by casting featurized data onto the hyperplane. Since kernelization allows the transformed features to be represented as an inner product of vectors in a high-dimensional space, it reduces training time and makes working in a high dimensional space feasible. We tried transforming the data using linear, polymonial, Gaussian RBF, and sigmoidal kernels.

D. Kernel Ridge Regression (KRR)

Kernel ridge regression is an extension of linear regression with techniques to reduce bias and variance. Bias is reduced by featurizing the dataset via the kernel trick (described above) prior to training the linear regressor. Variance is controlled by regularizaing: adding the L2-norm of the learned parameters to the squared error loss function when training.

$$J = \sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2 + \lambda\|\theta\|^2$$

Kernel	**Analytic form**
Linear	$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
Polynomial	$K(x,y) = \left(\alpha x^T y + c\right)^d$
Gaussian	$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
Sigmoidal	$K(x,y) = \tanh\left(\alpha x^T y + c\right)$
Laplacian	$k(x,y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right)$

TABLE I: All kernels used in kernel method models; SVM uses all but Laplacian, while KRR uses all.

Regularization prevents the model from overfitting by penalizing the model for weighing any feature too highly.

We used linear, polynomial, Gaussian RBF, sigmoidal, and Laplacian kernels to featurize our data.

E. Principal Component Analysis (PCA)

Principal Component Analysis is a technique to combat overfitting by dimensionally reducing the feature space. By computing an orthogonal basis set of principal components from the input data, where each principal component maximizes variability in the data set, PCA retains information about important trends in the data while reducing the model's propensity to learn about random noise. The principal component are derived by computing the eigenvectors of the covariance matrix of the data.

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1}\left\{\sum_i \left(\mathbf{x}_{(i)}\cdot\mathbf{w}\right)^2\right\}$$

RESULTS

Featurization

There were a number of issues that were encountered throughout the dataset cleaning and featurization process. First, given the size of the dataset, our attempts to featurize all of the compounds were limited by the long runtime of the processing. As a workaround, we attempted to run these scripts as Slurm jobs on Stanford's FarmShare cluster. Thus, our attempts to correctly featurize the data and train appropriate models were limited to a very few number caused by long queues and runtimes.

Further, we encountered problems with the featurization of target protein-ligand complexes that we added ourselves. By tracing through the deepchem's ligand featurization functions, we prepared SMILES strings that allowed us to stringify the composition of the complexes, as well as calculated the 3-dimensional positions of the molecules through the creation of an .SDF file, which provides the coordinates of each atom relative to the other compounds in the complex.

RF	10	20	50	100	200	500	1000
SVM	linear	cubic	RBF	sigmoid			
KRR	linear	cubic	RBF	Laplace	sigmoid		

TABLE II: Model parameters investigated. Entries for RF indicate number of trees in forest; entries for SVM and KRR indicate kernel used.

Baseline

To develop a rudimentary understanding of the feature and label space we used the core subset subset (189 points from the original 9039) to train an SVM regressor from the SciKitLearn library. The results were evaluated according to their mean squared error, which was 2.31 for the training set and 4.36 for the validation set. The results suggest that our current model is overfitting to the training set, and moreover, that a linear SVM is not a good descriptor for our data, which is what we expected.

Experiments

From here, we moved to run a battery of models to flesh out our sample space further. Since we knew very little about how the extremely high dimensional feature space actually correlated to the binding affinities, we aimed to mainly study first-order hyperparameters, i.e. the effect of the number of regression trees or the choice of kernel, rather than second order parameters such as the weighting of the regularization term in kernel ridge regression. We did not run cross validation, as our main object of study was the refined dataset, which is fairly large. As such, our studies on the smaller core dataset were not expected to be predictive, only informative about the behavior of the refined dataset.

The following table lists the models we studied, on both core and refined datasets, with and without PCA feature selection.

For this full battery on each data set, we generated parity plots and calculated Pearson R^2 coefficients for the train/dev/test split. From these, we chose the most effective models, as judged by test set prediction accuracy, to report. The parity plots and Pearson Pearson R^2 coefficients for the core dataset are shown in Figs. 2 and Fig. 3.

From these pictures, it is evident that for all models, the data overfits on the training set, with almost perfect correlation on the training set but poor predictive accuracy on the test set. The need for more data is evident– thus, we moved to run the same experiments on the larger refined set. These results are shown in Figs. 5 and 5.

Here, we see again that the SVM overfits (and performs poorly on the dev and test sets), but the RF50 and the KRR perform significantly better, reproducing a signifi-

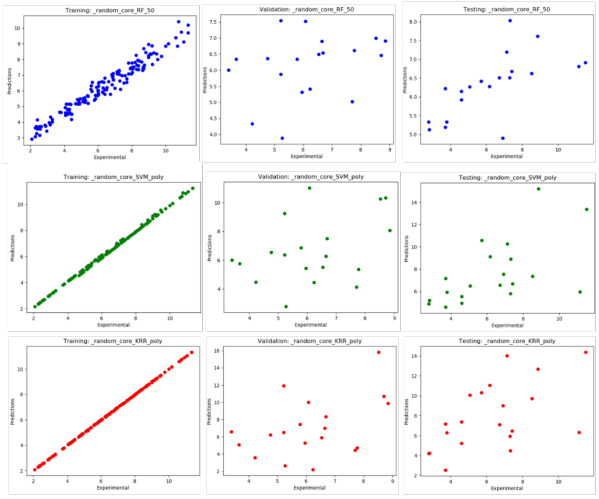

Fig 1: Parity plots for regression on core dataset *(n=189)*. Axes depict $pK_d - pK_i$.

FIG. 2: **Parity plots for the best performing of each category of model on the core dataset.**

Model	Train	Valid	Test
RF50	0.95583	0.22021	0.33663
SVM Poly_3	0.99883	0.05727	0.17337
KRR Poly_3	0.99999	0.09060	0.20296

FIG. 3: **Parity plots for the best performing of each category of model on the core dataset.**

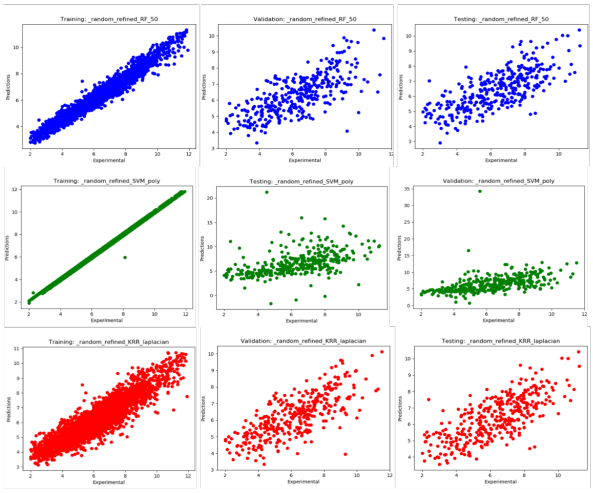

Fig 2: Parity plots for regression on refined dataset *(n=3,568)*. Axes depict $pK_d - pK_i$.

FIG. 4: **Parity plots for the best performing of each category of model on the refined dataset.**

Model	Train	Valid	Test
RF50	0.95598	0.52835	0.53343
SVM Poly_3	0.99738	0.25465	0.23003
KRR Laplacian	0.83470	0.55545	0.54080

FIG. 5: **Pearson R^2 scores for the best performing of each category of model on the refined dataset.**

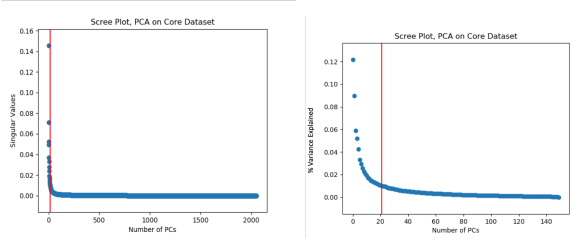

FIG. 6: **Scree plots for core (L) and refined (R) datasets. Cutoff is shown with a vertical red line**

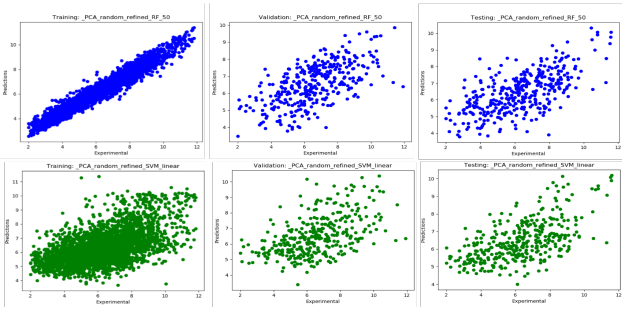

FIG. 7: **Parity plots for feature selected regression models on the refined dataset. KRR runtime exceeded cluster restrictions, and is thus not included.**

Model	Train	Valid	Test
RF50	0.94693	0.24687	0.28154
SVM Linear	0.31144	0.47570	0.27712
KRR Linear	0.34342	0.48877	0.33914

FIG. 8: **Pearson scores for feature selected regression models on the core dataset.**

cant positive correlation between experimental and predicted binding affinity values. In particular, we see that the ridge regression decreases the overall fit to the train set effectively, making for a more general model. However, the full-feature refined data set regression models took many hours to run on the cluster computers, rendering them unfeasible.

To combat overfitting and improve training time, we attempted to reduce our feature space down to a reasonable number of vectors by running PCA on the core and refined datasets prior to training on our battery of regression models. The results of dimensional reduction on the refined and core datasets are shown on the scree plots in Fig. 6

The plots show a definitive elbow in the amount of variance explained by the principal components. By filtering out all components that explain less than 1% of the variance in the datasets, we managed to reduce our core dataset to 22 features and the refined dataset to 16 features. Regression parity plots and Pearson scores for these feature-selected models are shown in Figs. 7 and 8. Unfortunately, the accuracy of the regression models we trained did not improve (and in fact worsened for several models). As the Pearson scores for training on the dimensionally reduced models demonstrate, the models are not only unable to accurately predict binding affinities for the core testing data; they also became unable to fit the training data very well. This is likely due our dimensional reduction. While many individual principal components explained less than 1% of the variance of our dataset, the composite of these principal components likely explain a significant amount of variation that was lost by our filtering process, thus lending less accuracy to our predictions.

FUTURE DIRECTIONS

The random forests turned out to be our best performing models, likely because the simpler models didn't overfit to noise in our datasets. In the future, we may try to train a convolutional neural network on this dataset in order to achieve a better binding affinity predictions. We could also perform more thorough error analysis on our model by analyzing the protein-ligand files that correspond to the worst predictions. Running featurized STX derivatives bound to human sodium channel 1.7 (pdb code 6J8H) through our model would generate binding affinity predictions. This can be done by generating hypothetical protein-ligand binding models using docking software (such as Schrodinger) for saxitoxin derivatives, and using a trained ML model to distinguish between candidate ligand binding affinities. Finally, we would chemically verify model predictions by synthesizing STX derivatives and experimentally measuring Ki, Kd.

CONTRIBUTIONS

All three authors contributed equally to this project, working together to understand and implement methods from the DeepChem library and open source machine

learning packages to run baseline tests. Jay ran the simulations on the cluster, wrote scripts for PCA, and interpreted graphical results. JC worked on featurizng and cleaning the dataset. Vamsi wrote scripts for training the regressor models. Jay and Vamsi assembled the report and the poster.

[1] J. R. Walker, P. A. Novick, W. H. Parsons, M. McGregor, J. Zablocki, V. S. Pande, and J. Du Bois, Proceedings of the National Academy of Sciences **109**, 18102 (2012).

[2] T. Shaikhina, D. Lowe, S. Daga, D. Briggs, R. Higgins, and N. Khovanova, IFAC-PapersOnLine **48**, 469 (2015).

[3] R. Wang, X. Fang, Y. Lu, and S. Wang, Journal of medicinal chemistry **47**, 2977 (2004).

[4] G. Subramanian, B. Ramsundar, V. Pande, and R. A. Denny, Journal of chemical information and modeling **56**, 1936 (2016).

[5] B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing, and Z. Wu, *Deep Learning for the Life Sciences* (O'Reilly Media, 2019) https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837.

[6] R. Wang, X. Fang, Y. Lu, C.-Y. Yang, and S. Wang, Journal of medicinal chemistry **48**, 4111 (2005).