Human Gene Function Prediction Challenge

Anuraag Bukkuri
bukku001
University of Minnesota
Minneapolis, USA

Linus Hennessee
henne519
University of Minnesota
Minneapolis, USA

Sydney Matthys
matth416
University of Minnesota
Minneapolis, USA

Kia Vang
vang0951
University of Minnesota
Minneapolis, USA

***Abstract*—Discovering the function of genes is critical to determining the importance of a gene to cell function. A machine learning algorithm that could predict the Gene Ontology (GO) term associated with a given gene would be an extremely valuable tool for exploring different genes and their functions. Machine learning has a variety of algorithms for prediction; in this project, the authors explore the performance of some of these algorithms on GO term performance. Algorithms explored are the following: k-Nearest Neighbors (kNN), Regularized Linear Regression (RLR), Random Forest (RF), and Feedforward Neural Networks (FNN). Algorithms are evaluated and compared using an area under the curve (AUC) score, averaged among 200 GO terms.**

I. Introduction

The GO (Gene Ontology) knowledgebase is the world's largest source of information about genes, and GO annotations on genes are extremely valuable to researchers working with said genes. A machine learning algorithm that could predict whether a GO term would be assigned to a given gene would be an extremely valuable tool for exploring different genes and what their functions may be. In the case of cancer, from a patient-derived biopsy, the physician could perform a gene screening and determine which drugs the patient would most likely respond to, improving patient outcomes and leading to a more personalized medicine approach.

This data set was derived from the Dependency Map Project and included genome-wide screening across 625 human cancer cell lines (DepMap, 2020). They were collected from a CRISPR-Cas9 screening. Each sample or data point corresponded to one gene across all cancer samples. The training data contained 6,154 genes and the test data contained 3077 genes to predict their respective GO term annotations. The GO terms were provided as a large matrix each column representing a GO term and rows corresponding to the gene. For each respective GO term there was a binary classification, 1 representing being annotated with the respective GO term and 0 means that the gene is not annotated with the GO term.

II. Methods

Four supervised machine learning approaches were used to predict GO term association with the given set of genes: kNN, RLR, RF, and FNN. We decided to pick a range of machine learning approaches to evaluate as this prediction problem involved 200 different GO terms. The wide range of options allows the different methods to exceed with AUC scores for different terms.

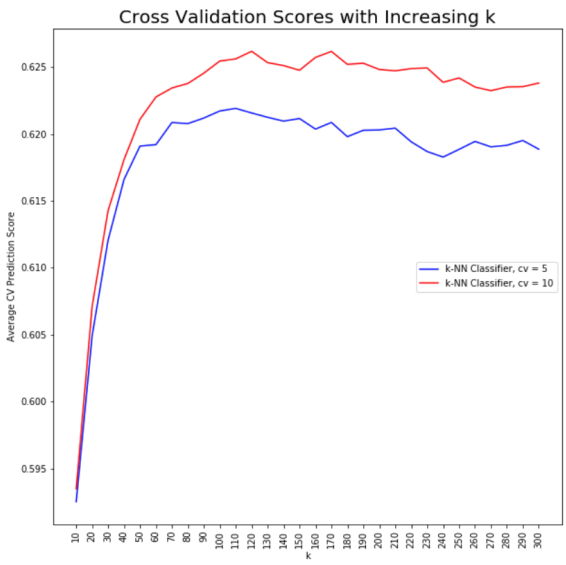

Fig. 1: Displays the cross validation scores as k increases. Respectively for ten fold and five fold cross validation.

A. k Nearest Neighbors (kNN)

One approach used to make GO term association predictions was kNN. It is a simple algorithm that implements learning based on the k nearest neighbors at a point of query, where k is defined by the user. Before implementing this algorithm, we first standardized the training and test data set and performed a 5-fold and 10-fold cross validation to avoid the methodological mistake of overfitting and to gauge kNN's estimating power. Cross validation and its procedure can be described as follows: given all data, a test set is held out for final evaluation and not used in the cross validation process. The training set is split into k smaller sets (known as folds) and for each of the folds, a model is trained using k-1 of the folds as training data. The resulting model is evaluated on the held out test set in which a performance measure is obtained. This measure is the average of the values computed in the iteration.

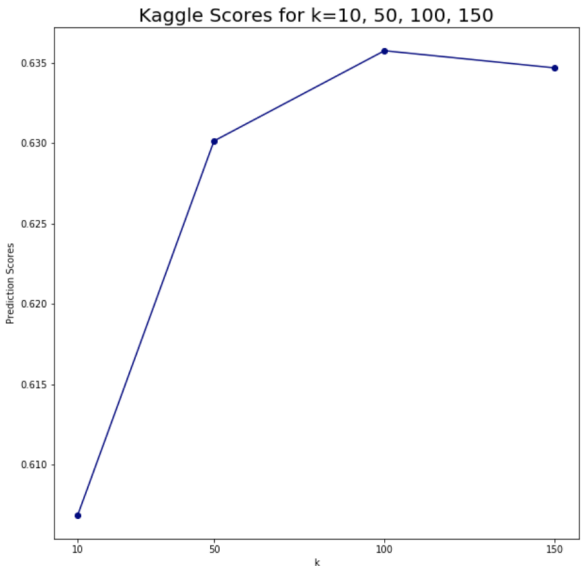

Fig. 2: Plots the Kaggle public prediction score as k increases from 10 to 150.

All average cross-validation scores were plotted with increasing k to determine k for our kNN model. We did this to see how increasing k can impact its estimating performance. It is observed that selecting k greater than 150 did not perform any better than k less than 100. We tested k = 10, 50, 100, 150. Kaggle's public scores confirmed our observation that k =100 showed the best estimating performance.

B. Regularized Logistic Regression (RLR)

A regularized logistic regression approach was used on the gene interaction data. Logistic regression uses a logistic function to model a dependent binary variable, in this case, whether a given GO term is associated with a gene or not. A regularized approach was used to avoid overfitting by penalizing high-valued regression coefficients (Glen, 2016). This is necessary because standard logistic regression and least squares approaches can be unstable, especially with the presence of multicollinearity. It does this through introducing penalty terms. In our model, we test the following four penalty terms: none, l1, l2, and elastic-net (with an l1 ratio of 0.5). The l1 penalty, as used in lasso regression, adds a penalty equal to the absolute value of the magnitude of the coefficients, often leading to sparse models. L2 regularization, as used in ridge regression, adds a penalty equal to the square of the magnitude of the coefficients, and does not result in sparse models. Finally, elastic-net penalties combine the l1 and l2 penalties with an indicated weighting.

For numerical purposes, different optimization algorithms were used for different penalty terms. For the none and l2 penalties, the limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) optimization algorithm, a member of the quasi-Newton method family, was used (Malouf 2002; Andrew Gao 2007). For numerical and data storage purposes, this algorithm was chosen over the BFGS algorithm because LBFGS uses an inverse Hessian estimate to search the variable space, while BFGS stores a dense nxn approximation to the inverse Hessian. Since LBFGS has only linear memory requirements, it works especially well with optimization problems with many variables, like the one we're dealing with here (Byrd et al. 1995; Zhu et al. 1997; Rafati Marcia 2019).

For the l1 and elastic-net penalties, the SAGA solver was used, a variant of SAG (stochastic average gradient) which supports the l1 penalty in sklearn, and is especially useful for sparse multinomial logistic regression and large data sets, like the one we have here; it also typically trains faster than SAG (Defazio et al. 2014). SAGA also has better theoretical convergence rates and has support for composite objectives with proximal operators on the regularizers (Defazio et al. 2014).

C. Random Forest (RF)

Another approach used on the gene interaction data to infer GO term association was random forest. Random forests are a class of ensemble learning methods which construct many decision trees while training and outputs a class that is the mode of the mean prediction (or classes in classification algorithms) of the trees (Ho 1995; Ho 1998). This method was chosen over basic decision trees (DT), since DT tends to overfit to their training data. It is worth pointing out the close relation between RF and kNNs: both are actually just different versions of a weighted neighborhood scheme (Lin Jeon 2002).

In our model, we used 100 trees in the forest. This number was used to strike a balance between computation time and ROC AUC. We also required only two samples for the algorithm to split a node. Finally, to measure the quality of a split, we tested two different criteria: gini impurity (intended for continuous attributes) and information gain (used for attributes which occur in classes). Though these metrics are quite similar, there are some key differences between them.

Gini impurity measures how likely a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the label distribution in the subset. The Gini impurity is calculated by summing the probability of an item with label i being chosen times the probability of a mistake in categorizing it. Intuitively, it thus reaches its minimum value of 0 when all cases in the node fall into a single category. Information gain is derived from the concept of entropy in information theory (hence the alternative name). Entropy is defined as -1 times the summing of probabilities of an item with label i being chosen times the log (base 2) of this probability. The information gain is then calculated by subtracting the sum of entropy of

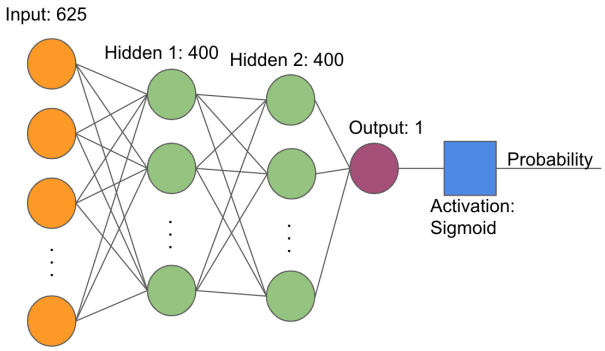

Fig. 3: Displays the structure of model two neural network for applied to this prediction challenge.

the children from the entropy of the parent. The goal at each step is to choose the split which produces the purest children nodes (i.e. finding the attribute which returns the highest information gain/the most homogenous branches).

Overall, note that entropy is slightly more computationally costly, due to the need to compute logarithms in its computation. Furthermore, due to the use of logarithms, lower probability values in entropy tend to be scaled up.

D. Feedforward Neural Networks

The original thought with the neural networks was to create one multi-label neural network. The general idea was to avoid creating multiple models for each of the respective 200 models. This proved to be a difficult task with neural networks. This plan was changed to creating a feedforward neural network for each respective GO term as a binary label classification problem. Feedforward neural networks were chosen due to their simple structure, as it was thought that it could be more generalizable across multiple GO terms in the neural network space. Feedforward networks are a type of model that only moves information forward to classify a sample. These networks take in samples as an input layer, and apply weights. The corresponding output after applying weights will become input for the next layer; this process continues until the output layer is reached. Weights are set randomly to 0 before training begins, and are adjusted and optimized using backpropagation.

There are two major python packages that implement neural networks, Tensorflow from Google and Pytorch from Facebook (torch.nn, 2020). Pytorch was chosen to build our models for its more python style structure, and for its easy installation. Pytorch is a powerful package that has multiple types of layers, loss functions and activation functions.

Neural networks contain many parameters, including but not limited to: number of layers, activation function, learning rate, loss functions, and optimizers. Each plays a role, our complete models contain the similar loss functions, learning rate, and optimizer to save from computation time. The neural networks all contain linear layers and intermittent activation functions applied. Two models were analyzed due to computation time and power. Model one contained an input layer which took in the gene sample size of 625, one linear hidden layer of size 300, and an output layer of size 1. The output layer is of size one because this is a binary problem. Once linear layers were applied the activation layer was used to give probabilities of being in class 1 or associated with the go term. The sigmoid activation function was applied as all values are in the range [0,1]. The binary cross entropy function was used to calculate loss between each iteration. The Adam optimizer with a learning rate of 0.001 was used as it is faster at converging to a minimum than the typical scholastic gradient descent which in turn saves on computation time as well as memory (Kingma Ba, 2015). Model one was trained for 2000 iterations each. Model two contains a different combination of layers, this model contains an input layer, two hidden layers of size 400, following the hidden layer and input layer a ReLU activation layer was applied, followed lastly by the output layer and sigmoid activation function. Figure 3. shows the structure of the neural network model. All other parameters are consistent with model one. Though each model was only trained until their loss value reached below 0.05 to help prevent overfitting from the first model. Each GO term was trained with the same structure of model, which means that 200 neural networks were trained for each evaluation submitted to kaggle.

III. Results

Overall results on the performance of each model was used by averaging the AUC scores from across 200 GO terms. This problem is tough for the fact that some of the GO terms are easily predicted while others are more complex. We evaluated our models with five-fold cross validation, to determine parameters and to judge the overall performance on the test data set. The test dataset provided was then split into two data set to be evaluated on the Kaggle challenge, a public and private group. Table 1 displays results of average AUC scores for the five-fold cross validation, public and private kaggle results.

Based on the cross validation scores we can see that we predicted all of our models to perform better than a random classifier on the test data. Though when looking at the scores most of the regularized linear regression models with the exception of the elastic net had Kaggle scores less than or approximately a random classifier. The feedforward neural network has a wide range of AUC scores for each of the individual GO terms. The range of scores found in the five fold cross validation were from 0.30 to 0.99. Meaning that on several of the go terms this classifier was near perfect but others it proved to be a harder challenge. While we originally thought kNN would perform the worst as a model, our cross

TABLE I: Results for all models used on the human gene prediction challenge. Each model displays a five fold cross validation score, along with its corresponding Kaggle private and public scores.

Model	Cross Validation Score	Public Kaggle Score	Private Kaggle Score
RLR - No Penalty	0.6035	0.4822	0.4928
RLR - Elastic Net	0.6318	0.6305	0.6358
RLR - L1	0.6276	0.4803	0.5010
RLR - L2	0.6233	0.4824	0.4979
RF - Entropy	0.5918	0.6058	0.6070
RF - Gini	0.5868	0.5159	0.5173
NN - 1 hidden layer	0.5972	0.6111	0.6124
NN - 2 hidden layers	0.6096	0.6134	0.6184
kNN - k = 10	0.5935	0.6068	0.6068
kNN - k = 50	0.6211	0.6301	0.6219
kNN - k = 100	0.6255	0.6358	0.6314
kNN - k = 150	0.6247	0.6347	0.6323

validation and Kaggle scores proved this not to be the case. In its simplicity, the kNN model outperformed other models we tested. Comparing cross-validation scores to Kaggle with increasing k, Kaggle public scores were consistently higher with an offset of 0.01.

IV. Discussion

A. kNN

kNN as a model was not computationally expensive to generate and performed the best overall. The default distance metric used in our kNN model is euclidean (unweighted). If given the same challenge in the future with more timing, we would be interested in trying different distance metrics and observe how they would impact kNN's performance. For example, other metrics such as manhattan (weighted and unweighted), and euclidean (weighted).

B. Regularized Linear Regression

When using the Regularized Logistic Regression model with an elastic-net penalty, our model performed fairly well, and had the second-best performance of all of our models when tested on the data in Kaggle. The penalties for the RLR model penalize high-valued regression coefficients, and are used to avoid overfitting. The elastic-net penalty combines the L1 and L2 penalties in a weighted manner (both were weighted evenly in our case), and it appears that combining the L1 and L2 penalties with elastic-net produced the best results. Based on the results of our 5-fold cross-validation, the models with the L1 and L2 penalties performed very similarly, with the model with no penalty performed by far the worst, likely because without a penalty, the model became overfit to the data.

One aspect of our results that jumps out is the discrepancy that exists between the results of our five-fold cross-validation and the results of testing on the Kaggle data for RLR models with L1, L2, and no penalty. One possible source for this discrepancy is that our models ended up overfitting to the data the models were trained and tested on. Overfitting occurs in a function such as the one in our RLR model when the function is fit too closely to the data the model is trained/tested on. This causes the model to perform very well on said test/train data, but makes the logistic function too complex, and not generalized enough to make accurate predictions outside of the test/train data. Normally, one would expect cross-validation to prevent an issue such as overfitting, since it changes where the test/train split is made in the data, preventing the model from reflecting irregularities in any one part of the test/train data, however it is possible that irregularities in the entire test/train data set caused overfitting that allowed for high 5-fold cross-validation scores, but ultimately performed poorly on the Kaggle data.It is possible that only the elastic-net penalty was for our RLR model was able to effectively prevent overfitting, causing it to be the only one of our RLR model of ours with a Kaggle score that lived up to its 5-fold cross-validation score. The other possibility to consider is that 5-fold cross-validation was simply inadequate for preventing overfitting, and that the more-commonly used 10-fold cross-validation would have produced scores closer to the ones produced by running our model on the Kaggle dataset.

C. Random Forest

In comparison with our other models, the Random Forest model performed poorly on the Kaggle dataset. Versions of our Neural Network, kNN, and Regularized Logistic Regression models all performed better than our highest-performing Random Forest model. Narrowing our view to just the Random Forest model, we used two different methods of measuring the quality of a split within the RF algorithm: gini impurity (the Gini model), and information gain (the Entropy model). Both performed decently when running 5-fold cross-validation; however, only the Entropy model continued this performance when used on the Kaggle data. The information gain method of measuring quality of a split involves logarithms, and naturally causes lower probability values to be scaled up. It's possible that the data we are working with naturally lends itself to this use of logarithms, causing the entropy model to perform better on the Kaggle data. It is also certainly possible that the Entropy model did a good job of preventing overfitting, whereas the Gini model did not, causing the discrepancy we see between the Gini

model's 5-fold cross-validation and Kaggle scores.

It is also worth noting that Random Forest models in general are very similar to kNN models, as both are simply different versions of a neighborhood scheme (Lin Jeon 2002).This is particularly interesting considering that our Random Forest model performed so poorly, while our most successful model was k-Nearest-Neighbor.

D. Feedforward Neural Network

There is much to improve on with the feedforward neural network, as there are many parameters to consider when training a FNN. Our analysis concluded that increasing the number of hidden layers from one to two brought some improvement in predictive power. We would consider adding more layers to the network. This could provide more accurate results but overfitting should be accounted for. Dropout layers would be beneficial to avoid overfitting the model when additional hidden layers are added. Other ideas to adjust for possible improvement are to adjust the layer sizes. Currently our models tested hidden layers of size 300 and 400. Reducing or expanding those layer sizes may improve additional prediction accuracy.

Another option to consider is to change the optimizer and learning rate of the optimizer. Adam was used to train the FNN for the memory and execution time benefits, though to make the adjustments more stable if time allowed, using scholastic gradient descent with a larger or smaller learning rate may improve the accuracy of the network.

Overall, the FNN needs additional time and computation power to explore the correct configuration to achieve better accuracy, to outperform the simpler model such as kNN.

V. Conclusion

The human gene prediction challenge has proven difficult to solve. If given the opportunity again, we would rerun our models and explore the option of "stacking," which is an ensemble machine learning algorithm that learns how to best combine the predictions from multiple well-performing models, and produces greater prediction results. By doing this, we would be able to combine the best aspects (i.e. which algorithms predict which GO term/gene associations best) from several machine learning models into a single one. Though training a neural network is no trivial task, we certainly expected better from this method. Given more time for training, perhaps its performance would have been improved. Finally, as was unexpected at the start of our project, kNN performed the best out of our 4 models overall. This may be due to the more complicated models overfitting the data and not capturing biologically relevant features in its predictions of GO term association.

VI. References

Andrew, G.; Gao, J. (2007). "Scalable training of L1-regularized log-linear models". Proceedings of the 24th International Conference on Machine Learning.

Byrd, R.H.; Lu, P; Nocedal, J. (1995). "A Limited Memory Algorithm for Bound Constrained Optimization". SIAM Journal on Scientific and Statistical Computing, 16, 5, pp. 1190-1208.

Defazio, A.; Bach, F.; Lacoste-Julien, S. (2014). "SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives". arXiv:1407.0202v3

Glen, S. (2016). "Regularization: Simple Definition, L1 L2 Penalties". Statistics How To. https://www.statisticshowto.com/regularization/

Ho, T.K. (1995). "Random Decision Forests". Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

Ho T.K. (1998). "The Random Subspace Method for Constructing Decision Forests". IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844.

Kingma, D. P., Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations (ICLR).

Lin, Y.; Jeon, Y. (2002). ?Random forests and adaptive nearest neighbors? (Technical report). Technical Report No. 1055. University of Wisconsin.

Malouf, R. (2002). "A comparison of algorithms for maximum entropy parameter estimation". Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002). pp. 49–55.

Rafati, J.; Marcia, R.F. (2019). "Deep Reinforcement Learning via L-BFGS Optimization". arXiv: 1811.02693v2.

torch.nn — PyTorch master documentation. (2020). Pytorch.org. Retrieved 8 May 2020, from https://pytorch.org/docs/stable/nn.htmllinear-layers

Zhu, C; Byrd, R.H.; Nocedal, J. (1997). "L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization". ACM Transactions on Mathematical Software, 23, 4, pp. 550 - 560.