What Are The Machines Actually Learning?
02-620: Machine Learning For Scientists Report

Siddharth Reed
Department of Computational Biology
Carnegie Mellon University
Pittsburgh, PA 15213
slreed@andrew.cmu.edu

1 Introduction

1.1 Problem & Motivation

It goes without saying that cancer is an incredibly complex, variable disease affecting the expression of many, many genes. The catalysts of malignancy can also be quite variable, resulting from certain behavioural, environmental or genetic factors. Even though researchers and physicians have catalogued many oncogenes that are frequently the primary drivers of cancer, there is still significant heterogeneity among gene expression in tumors [1]. This heterogeneity, as well as the rise of next generation sequencing technology, is what is driving researchers to develop treatment strategies personalized to individual patients and even individual tumors. Having a better understanding of the genetic landscape and architecture of various cancers is critical to effective, timely and personalized treatments.

Scientists are now able to accurately capture much of this information with genomic and transcriptomic data for an individual patient at a relatively low and relatively high throughput. However this is often too much information to be parsed in a timely manner, even by an experienced bioinformatician, let alone a physician. This is part of why machine learning methods have gained such popularity for biological applications, they are made to work with large volumes of data to produce instant predictions. Even if they only help guide the intuition of doctors they can be tremendously helpful, especially as we continue to generate more data and develop more robust, computationally efficient methods. But unlike a doctor machines are not able to show their work per se, a machine learning model cannot really tell you *why* it says a patient has cancer.

Much of machine learning work, especially when applied to medical diagnosis, if often training a model to preform some kind of classification task (e.g. diseased/non diseased). Most of the most successful machine learning methods applied to medical diagnosis problems are deep learning methods [2]. Such methods, while often capable of highly accurate and robust predictions offer little more that the actual diagnosis result. Consider a multilayer perceptron trained to diagnose cancer from gene expression data, the parameters learned for the intermediate layer nodes have no clear correspondence with which genes are actually important in making the prediction. If this model is wrong, not matter how unlikely, there is really no way to know without some further kind of testing or treatment which may have negative heath effects. Either someone who is diseased may not be recommended treatment it or someone who is not diseased undergoes a potentially harmful treatment. Being able to "debug" treatments and why they did/did not work is a crucial part of modern medicine, as the saying goes "it's called the medical practice for a reason".

However not all machine learning approaches are as opaque as a neural network, many methods exists that have learned parameters that correspond to the actual features the model is trained on. In the case of transcriptomic data these features are genes, so getting some clue as to what the model is prioritizing can greatly help one understand where and why a model's prediction is right *or* wrong. Instead of relying on black-box solvers perhaps we can build less accurate but more transparent models I want to show that using transparent models we can try and extract some relevant biological information about our problem, in this case lung cancer classification from transcriptomic data. Using the feature (gene) importances built into some of these simple, transparent methods we can hopefully understand more about lung cancer. As well I want to see how variable machine learning models are in terms of what is learned, even when applied to the same data for the same problem

1.2 Approach

Consider the following thought experiment, John goes to a job interview feeling woefully unprepared and sees another man, Smith, waiting outside the interviewer's office. Smith starts boasting about shows John the lucky coin in his pocket, does his interview and finishes, confiding in John that he is sure he will get the job. John believes that the man with a coin in his pocket will get the job and does his interview. During the interview John in offered the job over Smith and accepts, upon leaving he notices that he in fact has a coin in his pocket. John is right that the man with a coin in his pocket will get the job but for completely incorrect reasons. In our cancer problem a model we don't want to use is "black box solver" specifically for the cases where it can end up like John. He may get it right but why he get's it right is just as important if not more for these kinds of problems. It is able to accurately predict a pateint's disease given the gene expression data but provides no interpretable information about how the features (sequencing results) relate to the diagnosis. Since I am working with biological data for diagnosis I want to use methods that are both accurate and informative. Thankfully many classifiers are much more transparent than about what they are learning from the data and are much better for humans to interpret and dissect. For this goal I chose to use the following supervised learning methods

- Random Forest
- Support Vector Machine
- Naive Bayes

Random forests provide decision paths for each tree and you can look at how informative certain splits or conditions in the data are. They also provide the relative importance or features (genes) which can be hugely beneficial for follow experiments and treatments (especially has gene therapies become usable). Similarly SVMs provide support vectors, points (patients) that are nearest to the decision boundaries and can help physicians come up with decision rules for diagnoses. They can also provide confidence levels for their classifications, as a function of the distance of a newly classified patient to the decision boundary. Naive Bayes is generative and builds a distribution of the underlying data which is used for classification. The naive assumption can also be relaxed for known clusters of genes to potentially improve performance and the method scales well to having many (in the case $\approx 20,000$) features.

1.3 Data

The data I am using is transcriptomic data from human lung samples. Specifically I am using 374 lung samples from the GTEx project and 601 Lung Adenocarcinoma samples from the TCGA project. The GTEx project was meant to collect "normal" gene expression data, normal in the sense that the individuals sampled did not have any specific health condition. This helps establish a baseline of what normal gene expression looks like for various tissues, in our case it shows us what genes we expect to be expressed in the lung. TCGA (The Cancer Genome Atlas) was another large project meant to collect various forms of data from (you guessed it) cancer tissues to see what drives cancer growth, both generally and in specific cancer types. In our case we use it to see what expression patterns are specific to lung cancer, such that they may help with diagnosis (classification) and shed light onto important processes that lung tumors accelerate or halt.

In order to avoid processing the raw data myself I use data from the recount2 project, which has processed all of the raw GTEx and TCGA sequence data to generate gene counts for each sample [3]. They do this using a robust, standard pipeline to ensure that all of the count data is comparable. I specifically used the gene-level counts, so any read from the raw data that mapped anywhere thing the boundary of a gene (as defined by ensembl biomart) counts towards that gene, even if the reads represent two distinct transcripts. For more details about the specifics please refer to the original paper [3]. I further filtered this data to only include counts for protein coding genes as defined by ensembl biomart ($\sim$ 20,000 genes). The gene counts and files specifying which samples were lung samples were downloaded from the recount website here. I wrote custom bash and python scripts to clean and join all of the data for training the models.

Due to various logistical issues with actually installing the recount package I could not use their provided normalization functions or access sequencing depth values. I elected to use quantile normalization since it does not require sample/gene specific information and is already implemented in `scikit-learn`.

The GO terms associated with each gene were also taken from ensembl biomart provided in the file `./data/downloaded/gene_metadata.txt`. All other downloaded files (gene counts, lung sample ids, coding gene list) are in the `./data/downloaded/` directory as well.

2 Methods

2.1 Machine Learning Methods

SVM A support vector machine (SVM) is a machine learning model best suited for binary classification tasks. An SVM when trained provides a decision boundary in sample space that best separates the samples by their label. So ideally only samples from the same class are on a given side of our learned boundary, however this may not be possible for non-linearly separable data, at least without some tricks. In fact we want to ensure our boundary separates the classes optimally, so we wan to maximize the dot product (distance) between and sample point and the boundary line itself. This is called the maximum-margin principle, the margin referring to the smallest distance between two samples with different labels. Maximizing this distance ensures that even our most ambiguous samples are made to be as distinct as possible in terms of how the decision boundary would classify them.

This boundary is actually just a line in vector space so it can be represented by this simple equation

$$w'x + b = 0$$

where w defines the line in space, b is the bias and x is any sample. Similarly you can define the lines parallel to the boundary that define the margin with

$$w'x + b = 1 \tag{1}$$

$$w'x + b = -1 \tag{2}$$

which also defines restrictions for our model scheme i.e. for any sample x in class 1 then $w'x + b \geq 1$ and for any sample in class 2 then $w'x + b \leq -1$. This is also how we classify new points with a learned SVM, we simply evolute the sign of $w'x_{new} + b$ to determine our class label. Note that no sample can be inside the margin due to how its defined, as even the samples closest to the boundary line are at best on one of the margin lines.

Since we are still solving for something we can define the width of the margin $M = \frac{2}{\sqrt{w'w}}$ so now we have an objective function to maximize (or minimize the inverse). However if our data is not linearly separable then we need to enforce some kind of penalty for misclassified samples, our misclassified points should be close to the boundary. We can use the distance (dot product) of the misclassified point to the margin boundary that is should be closest too. So our optimization is now to minimize the penalty and the margin width inverse i.e. $min\frac{w'w}{2} + \sum_i^n \epsilon_i$ where ϵ_i is our misclassification error for x_i, further constrained by $w'x + b \geq 1 - \epsilon_i$ and $w'x + b \leq -1 + \epsilon_i$ for our classifications.

There are more mathematical tricks to simplify/speed up the computation of and SVM, such as the Dual formulation. Here we express the SVM optimization as a Lagrangian with non-zero coefficients only for those samples that are on the margin boundaries (support vectors). All others points do not influence the margin and can be ignored during our optimization as we only care when our Lagrangian parameters are not zero (support vectors). There are also kernel tricks, where projecting non-separable data into a higher dimension using a kernel function that then allow a linear boundary to perfectly separate the labelled data. SVM have also been adapted for multi class classification and regression problems

Naive Bayes Naive Bayes is a generative classifier, often used for predicting a categorical variable from a set of categorical observations. Specifically we want to learn the parameters θ for a distribution

$P(Y|X,\theta)$ to predict some categorical Y outcome. As soon as we start to grow the number of features n, even if each feature has only 2 possible vales we will need to learn 2^n parameters to define our joint distribution $P(Y|X_1, \ldots X_n) = \frac{P(X_1,\ldots X_n|Y)P(Y)}{P(X_1,\ldots X_n)}$. Not only does this quickly becomes intractable (especially for genomic data) but there are likely many pairs of features that can be treated as independent without losing any significant predictive power in the model.

This is the Naivete of the Naive Bayes model, assuming all features are independent so that we can be ~~lazy~~ efficient. So we can redefine our condition distribution $P(X_1, \ldots X_n|Y) = \prod_i P(X_i|Y)$, brining us from 2^n parameters (most of which were probably useless) to $4n$ parameters. Note that thus far our $P()$ function is completely arbitrary still, it can be defined as a Bernoulli or Multinoulli or Gaussian distribution depending on the case. So whatever parameters we need to learn are dependant on which probability distribution we decide to use, which is in turn likely dependant on our data and goal.

Since Naive Bayes is simply computing and multiplying a bunch of different conditional probability distributions actually training the model is simply a counting exercise given the data, assuming a Bernoulli or Multinoulli distribution. Even in the case of a Gaussian one simply defines a normal distribution with a μ, σ that are computed from subsets of the data for each output class. These are formalized with closed form MLE and MAP definitions that will depend on your probability distribution. One key issues is that if any distribution $P(X_i|Y) = 0$ then the entire model is ruined by only a single poor parameter, but this can be avoided by sampling adding pseudocounts to avoid a zero and perturb the data as minimally as possible.

Once we have learned our distribution $P(Y, X_1, \ldots, X_n)$ we can easily classify a new point X_{new} by simply evaluating $P(Y = i|X_1, \ldots, X_n)$ for all i output categories and picking the most probably one.

Random Forest A random forest is a ensemble classification model, which aggregates the predictions of many distinct decisions trees to predict (often) some kind of categorical output. This definition likely raises the question "what is a decision tree?". A decision tree is essentially a flowchart, where you start at the root and at each node you pick a branch contingent on the data for your sample until you reach a leaf node which reprints the classification label. So the problem for a decision tree now becomes "how do I decide what conditions lead to what nodes?". This is where information theory comes in, helping to quantify how much information we can gain by splitting the data at a given threshold for a given feature. The ideal case is that some split segregates all the labels perfectly (no label mixing in any partion after a split), which we can quantify using the entropy H, where a perfect split would correspond to an entropy value of 0. This is very rare, but the closer our split is to perfect, the more informative it is and the fewer splits we will likely need to end up with a correct prediction. So if we can find splits that minimize entropy we can be sure we are finding the most informative splits possible. There are also techniques to prune or modify existing tree to ensure they remain simple and accurate.

It can often be difficult to build a really accurate, robust, *single* decision tree, especially as more continuous features are added. But if we can take advantage of the fact that decision trees are very simple and cheap, we can try and aggregate our classification over many different decision trees. If we train each decision tree on a random subset of our data we can 1) avoid overfitting of any single tree and 2) ideally grow enough trees that most of our trees are capturing something about the structure of our data. We can compensate for the high-variance of the classification by simple building more decision trees so that no individual bad tree can ruin our classification.

Random forests also have the advantage of being incredibly transparent, as each set of decisions leading to a classifications are directly encoded as thresholds among the features. Even when looking at a forest you can see which trees voted for which class and what splits they came to that drove that classification.

2.2 Feature Importance

Here I sought to examine which features (genes) each of these models prioritized or deemed important for distinguishing deathly and cancerous lung cells. I chose the methods explained because they are, at least somewhat, transparent, some connection can be drawn between the parameters learned in eahc model and [1]

3 Results

I trained each mode on the 347 GTEx + 601 TCGA samples with $\sim 20,000$ genes as features. I trained each model on a random split of 77% of the data, I then tested the accuracy of the trained models on the test data with the following results (Table 1). Clearly the models preformed well so I had some confidence in moving forward, but this is likely some kind of overfitting due to how many features were used.

Model	Test Accuracy
SVM	0.9937888198757764
Naive Bayes	0.9503105590062112
Random Forest	0.9968944099378882

Table 1: Model accuracy on 33% test set of transcriptomic data for healthy/cancerous lung cells.

	Method			
Rank	SVM	Naive Bayes	Random Forest	log_2 Fold Change
1	ENSG00000198938.2	ENSG00000198804.2	ENSG00000188269.8	ENSG00000134193.14
2	ENSG00000115414.18	ENSG00000168878.16	ENSG00000160013.8	ENSG00000150244.11
3	ENSG00000019582.14	ENSG00000198886.2	ENSG00000047648.21	ENSG00000147381.11
4	ENSG00000254709.7	ENSG00000075624.13	ENSG00000018280.16	ENSG00000221867.8
5	ENSG00000086548.8	ENSG00000185303.15	ENSG00000206047.2	ENSG00000170373.8

Table 2: Most important genes (features) for each model as well as the most differentially expressed genes (largest log_2 fold change and adjusted p-value < 0.05)

Here Table 2 shows the genes that had the largest importance values for each model as well as the most differentially expressed genes. I keep the DE genes here as they provide an intuitive reference for what genes should be good for training a binary classifier on this data. It is interesting to note that there is no overlap among any of the models or even the DE genes, indicating that they all found different ways to accurately solve the classification problem. After looking up online it appears that many of these genes are related to generating energy (cytochromes and cytoskeletal rearrangement (cell adhesion, fibronectin) which may be related to tumor growth and expansion. However outliers like an olfactory receptor and prostoglandin are also present, so perhaps there is some issues with my pre-processing of the data.

[1]Note that due to logistical issues all data presented in the results uses the scikit-learn implementations of the models described but I still have implemented SVM and Naive Bayes myself.

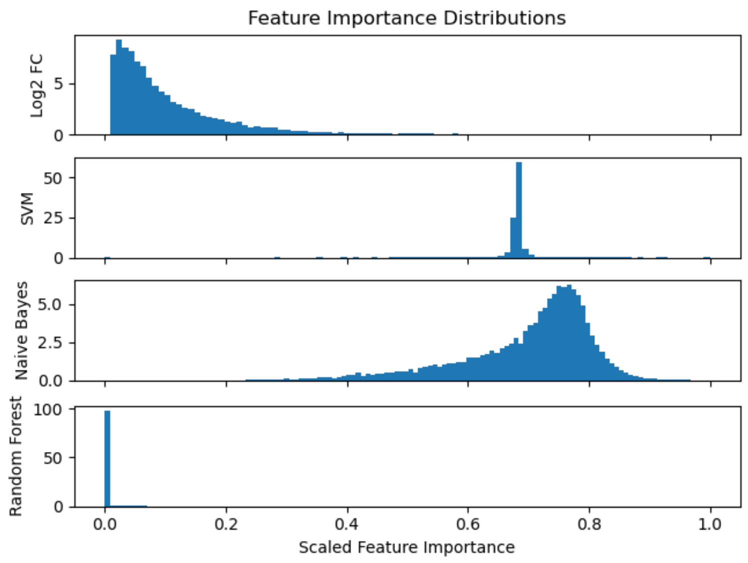

Figure 1: Distribution of feature importances scaled to [0,1] for each model. For Log2 FC the absolute log fold change is show for genes that are differential expressed in cancer (adjusted p-value < 0.05)

I find Figure 1 very interesting as even how many important genes or the average importance of genes are starkly different between the models. It makes sense for Naive Bayes as a distribution is estimated for each gene, so the genes all posses some level of importance, but clearly there are some outliers of both highly influential and highly inflectional genes The random forest distribution is also somewhat intuitive as in many cancer there are only a handful of genes that actually drive the growth of cancer and undergo large changes in expression. This also lines up with the DE results since only about ~ 4000 genes were significantly differential expressed with a fold change > 2.

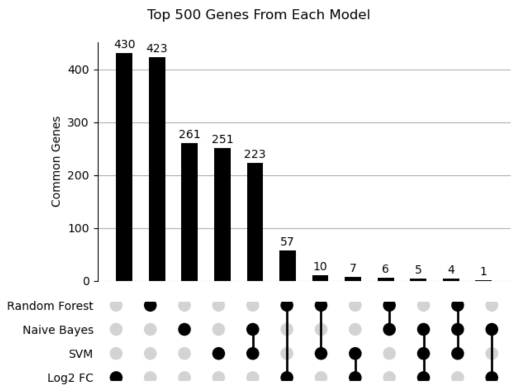

Figure 2: Upset plot of the top 500 most important genes for each model. For Log2 FC the absolute log fold change is show for genes that are differential expressed in cancer (adjusted p-value < 0.05)

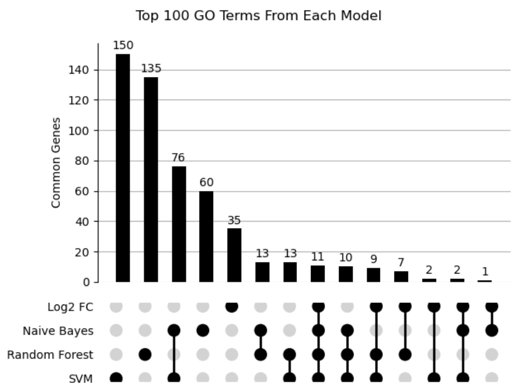

Figure 3: Upset plot of all GO terms associated with the top 100 most important genes for each model. For Log2 FC the absolute log fold change is show for genes that are differential expressed in cancer (adjusted p-value < 0.05)

Again in Figure 2 and3 both show that the models have very little in common in terms of what genes they deemed important, but there are at least a few genes that all the models found.[2] It is especially curios to see that 4 genes were prioritized by all 3 models but were not among the most differentially expressed genes.

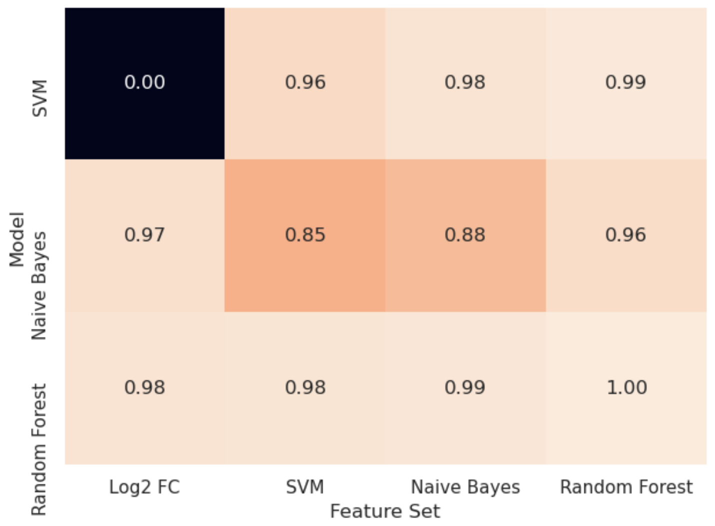

Figure 4: Accuracy on the test data (n=322) for each model trained on the top 50 most important genes as determined by the trained model for the feature set. So the top 50 most important genes for SVM, when used to train a Naive Bayes Classifier, has an accuracy of 0.85. The top 50 absolute log2 fold change genes with adjusted p-value < 0.05 were also used as a feature set.

Finally I tried to train each model on the top 50 most important genes found by each other model or by DE. And despite the minimal overlap in what genes the models prioritize it seems like they are all capable of learning well from each other, save for Naive Bayes which appears to be lagging a little. I would expect that this is because Naive Bayes is less robust to noise due to considering all features in classification, which is not necessarily true for the SVM or random forest. Note that the SVM cell is 0 because the model failed to converge. It is also possible that 50 genes is too many for 947 samples and may still be overfitting.

[2]Note that this is an upset plot, an alternative to a venn diagram, all categories are mutually exclusive, if you wish to know more here is link to the publication.

4 Conclusion

Findings I find it interesting how all of these methods found clearly distinct patterns of genes to distinguish cancer an healthy tissue. Among the most important genes found very few were common to even 2 of the methods and the vast majority were method specific. Despite this these genes all seem to be in some way informative enough to allow for accurate classification from all other methods. It makes me curious to further investigate what about the structure of these models (or quirks in the data or flaws in my work) lead to such stark distinctions. Cancer is also known to be very complex so the idea that there may be distinct by specific genetic signatures for it is perhaps not too surprising.

Future Directions I feel that there is perhaps a way to adapt feature importances as a method of feature engineering/dimensionality reduction. In a sense we let the machines learn what is important and report back to us, instead of just using them for classification. It would also be interesting to further investigate the actual genes and pathways discovered here and look for functional consequences or interactions between differently important genes. There is also the effort by some to develop deep learning methods that still retain some kind of interpretability in thier latent variables [4]. Associating the latent variables with known functional groups of genes (GO, MSigSB etc) allows research to take advantage of the power of deep learning while still having some sense of what their machine is actually learning.

References

1. Croce, C. M. Oncogenes and Cancer. en. *New England Journal of Medicine* **358,** 502–511. ISSN: 0028-4793, 1533-4406. http://www.nejm.org/doi/abs/10.1056/NEJMra072367 (2021) (Jan. 2008).
2. Xu, C. & Jackson, S. A. Machine learning and complex biological data. en. *Genome Biology* **20,** 76, s13059–019–1689–. ISSN: 1474-760X. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1689-0 (2021) (Dec. 2019).
3. Collado-Torres, L. *et al.* Reproducible RNA-seq analysis using recount2. en. *Nature Biotechnology* **35,** 319–321. ISSN: 1087-0156, 1546-1696. http://www.nature.com/articles/nbt.3838 (2021) (Apr. 2017).
4. Rybakov, S., Lotfollahi, M., Theis, F. J. & Wold, F. A. Learning interpretable latent autoencoder representations with annotations offeature sets. *arXiv.* https://www.biorxiv.org/content/biorxiv/early/2020/12/03/2020.12.02.401182.full.pdf (2021) (2019).