Applications of Machine Learning in Acute Myeloid Leukemia Prediction

Joanna Bitton and Erica Dominic

New York University
jtb470@nyu.edu, ehd255@nyu.edu

Abstract

Acute myeloid leukemia (AML) is a blood cancer correlated with certain genetic mutations. In this project, we implement six machine learning methods with the intent of detecting AML, given a patient's genetic profile and hematological data. The six models include support vector machines, classification trees, boosted trees, random forests, Naive Bayes, and logistic regression. Models were optimized using the metric area under the receiver operating characteristic (ROC) curve, and accuracy scores were also recorded. Our results indicate that the models with the highest accuracy (just over 70%) were random forests and logistic regression. The models with the lowest accuracy (around 60%) were support vector machines and Naive Bayes.

I. Introduction

Acute myeloid leukemia (AML) is a blood cancer that interferes with the function of normal blood cells. AML has a five-year survival rate of only 27.4%, and there are about 20,000 new cases in the United States every year [1].

Research has shown correlations between AML and certain genetic mutations; these genetic mutations appear in an individual's genome years before onset of the disease [2]. These genetic changes can serve as biomarkers for cancer cell proliferation, allowing for early detection of AML if an accurate screening test can be developed.

The goal of this project was to develop a method for the identification of individuals who are at risk for developing AML based on their blood and genetic profile. We applied several machine learning models that leverage a patient's genotype and hematological data in order to predict whether or not that individual has AML.

The benefits of creating such an accurate predictive model for AML are twofold. Firstly, generating a robust tool will greatly aid in early detection of AML. As with many types of cancer, early detection is key to successful treatment of patients and increased overall survival rate [3].

Secondly, the parameter vector produced by our model could potentially highlight other significant genetic changes/features that have been overlooked by the research community thus far.

II. The Data

1 Source

The data was provided by Dr. Duane Hassane at the Department of Medicine, Weill Cornell Medicine, New York.

2 Description

The dataset includes information from approximately 400 subjects, comprised of patients with and without AML. Patients without AML may have other diseases. For each subject, the following key attributes are recorded:

- allelic fractions for 45 distinct genes
- hematocrit
- platelet count
- white blood cell count
- hemoglobin
- age
- diagnosis of AML (yes/no)

3 Plots

The following plots show two-dimensional cross sections of the 50-dimensional input space. Each plot compares two of the following features: total allelic fraction (summed over all 45 genes), age, white blood cell count, platelet count, hemoglobin, and hematocrit.

The green plus signs signify AML patients and the blue points represent patients who have not been diagnosed with AML.

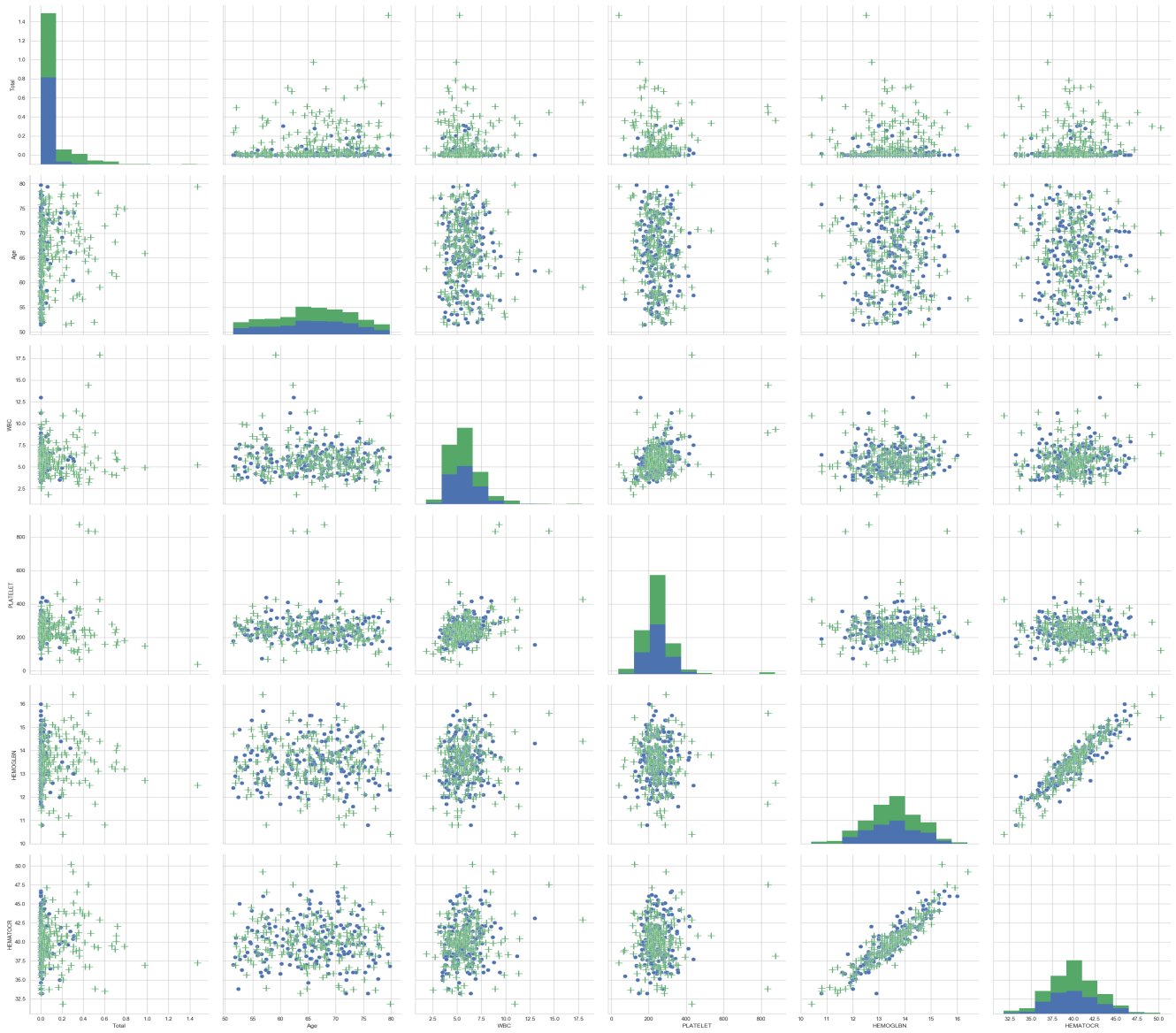

Figure 1: Green = diagnosed with AML; blue = not diagnosed with AML

4 Preprocessing

The following steps were taken in the data preprocessing phase:

1. Deleted irrelevant columns (such as IDs).
2. Converted all textual labels to numerical values.
3. Filled in missing values (approximately ten throughout the dataset) with column average.
4. Added a column that holds the total allelic fraction (summed over all 45 genes).
5. Added interaction features (optional, set by flag). The interaction features were created from the following given features: hematocrit, platelet count, white blood cell count, hemoglobin, age, total allelic fraction. These features were multiplied pairwise to create 21 new interaction features.
6. Partitioned the data into random train and test subsets using sklearn's `train_test_split()`. 65% of the input data is used for training; the remaining 35% is reserved for testing. See discussion below.
7. Standardized the data. (Deleted columns with variance zero; for the remaining columns, subtracted the column mean and divided by column variance.)
8. Added a column for the bias term.

5 Train-Test Split

In general, 20% of the dataset is reserved for testing purposes. However, due to our small dataset and the sparsity of the columns, we found through repeated experimentation that model performance was highly dependent on the specific train-test split of the data.

In order to temper that variability, we chose to use 35% of the available data as a test set. This necessarily led to a smaller training set, and possibly a tradeoff between performance and stability of results. We also set a random seed when calling `train_test_split()` for the purpose of reproducibility.

IV. Performance Evaluation

Area under the Receiver Operating Characteristic (ROC) curve was used to evaluate the performance of each model. The ROC curve plots true positive rate vs. false positive rate as the prediction threshold varies. The threshold value is the cut off point for predictions; if a prediction is at or above the specified threshold, we predict that data point is positive (the patient has AML).

Ideally, we want the ROC curve to pass through the upper left corner of the graph, where true positive rate is at a maximum and false positive rate is zero. In practice perfect prediction is unlikely, but a larger area under the curve indicates better performance. Note that the largest area under the curve we can attain is one.

ROC analysis is a performance measure is often used to assess diagnostic test accuracy. Variations of ROC analysis are commonly used in bioinformatics classification problems.

One added benefit of the ROC curve is that it allows us to find the optimal prediction threshold for probability models. After plotting the ROC curve for each tuned model, we found the threshold value that optimized accuracy, where accuracy is defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

where

TP = the number of true positives
TN = the number of true negatives
FP = the number of false positives
FN = the number of false negatives

Accuracy is a reliable performance metric because the dataset contains balanced class sizes.

V. Binary Classification Models

1 Support Vector Machines

The first model attempted was sklearn's Support Vector Classifier. With its default parameters, area under the ROC curve was 0.654 (without interaction features). After re-examining the data, we hypothesized that the low value was due to the data being linearly inseparable. Thus we decided to incorporate nonlinearity into the model via interaction features (discussed in section IV.4 above). The addition of interaction features decreased performance on the test set to 0.644, possibly due to overfitting.

To increase the area under the curve (AUC) and better assess the stability of the model, we used sklearn's `GridSearchCV` with 10-fold cross validation. We tuned the following hyperparameters:

- kernel function (e.g. RBF, sigmoid)
- regularization parameter
- kernel coefficient
- independent term in the kernel function (used in the polynomial and sigmoid kernels)
- degree of the polynomial kernel

See Appendix A for an example of hyperparameter tuning and grid search results.

Optimal models used a sigmoid kernel. After hyperparameter tuning (on the original data and interaction features), AUC was 0.659. After hyperparameter tuning (not including interaction features), AUC was 0.693.

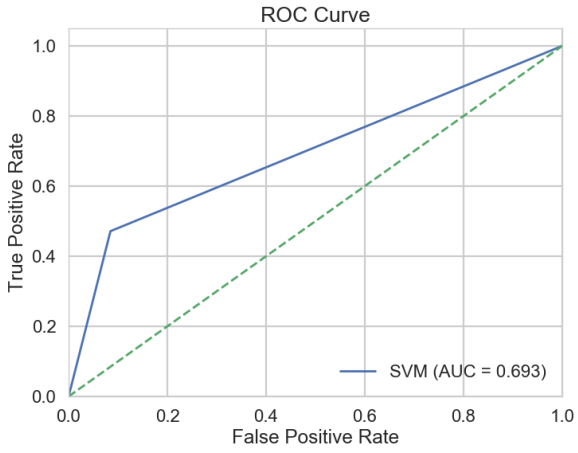

Figure 2: ROC curve for best SVM model

This AUC corresponded to an accuracy of **67.3%** on the test set. The results for SVM showed promise; our next step was to compare these results to that of a nonlinear binary classification model.

2 Classification Trees

Continuing our exploration of binary classifiers, we turned to sklearn's Decision Tree Classifier. With its default parameters, AUC was 0.589 (without interaction features) and 0.629 (with interaction features).

We once again used `GridSearchCV` with 10-fold cross validation to tune the classification tree hyperparameters:

- node impurity measure (i.e. entropy or Gini)
- strategy used to choose the split at each node (best or random)
- maximum depth of the tree
- minimum number of samples required to split a node
- minimum number of samples required to be present in each leaf
- minimum impurity decrease (each split must decrease impurity by this value or more)

After hyperparameter tuning, the resulting AUC was 0.713 (without interaction features) and 0.690 (with interaction features).

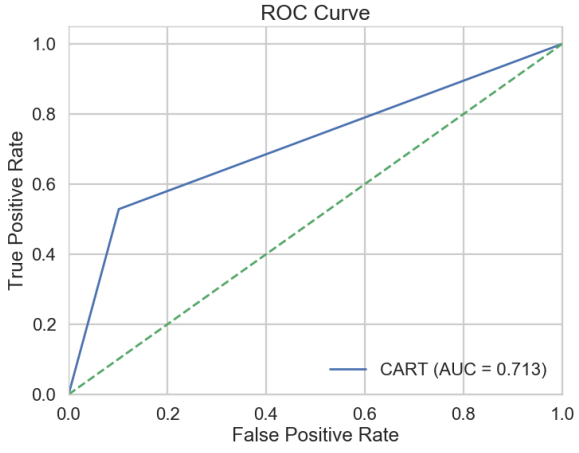

Figure 3: ROC curve for best CART model

The larger AUC measurement corresponded to an accuracy of **69.8%** on the test set.

Interestingly, the optimal model had a maximum depth of five. This made it easy to inspect the structure of the tree:

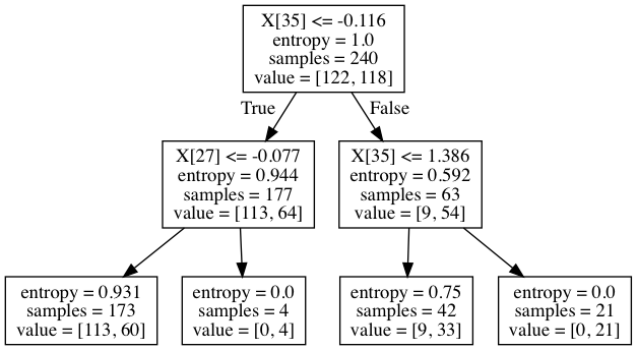

Figure 4: Best classification tree

Although unnecessary, features were centered and scaled, hence the negative splitting values. Note also that the samples in the diagram reference the samples in the training set.

For a tree of such limited depth, it attains a respectable accuracy of nearly 70%. The root node of the tree splits according to the total allelic fraction (which summed the allelic fractions over 45 genes). The tree contains two more splits; one uses total allelic fraction again, while the other (denoted X[27] in the graph) splits on gene 43.

It is tempting to attribute great importance to gene 43, but worthwhile to note that it pulls only four samples from a group of 177 to create a pure leaf node. (For reference, the dataset contains roughly 20 patients with a nonzero value for gene 43, only some of which are part of the training set.) A more in-depth discussion of feature importance can be found in section IV.

3 Boosted Trees

Given the improved accuracy of classification trees over SVM and the apparent nonlinearity in our data, our next step was to build on the success of CART with boosted trees. For this we used the XGBoost Classifier from the XGBoost library.

Under default parameters, AUC was 0.617 (without interaction features) and 0.665 (using interaction features).

We tuned the following parameters via `GridSearchCV` and 10-fold cross validation:

- number of boosted trees to fit
- maximum depth of the trees
- learning rate

- ℓ_2 regularization parameter
- ℓ_1 regularization parameter
- minimum loss reduction (each split must decrease the loss by this value or more)

After hyperparameter tuning, the resulting AUC was 0.705 (without interaction features) and 0.719 (with interaction features), which was an improvement over the classification tree AUC.

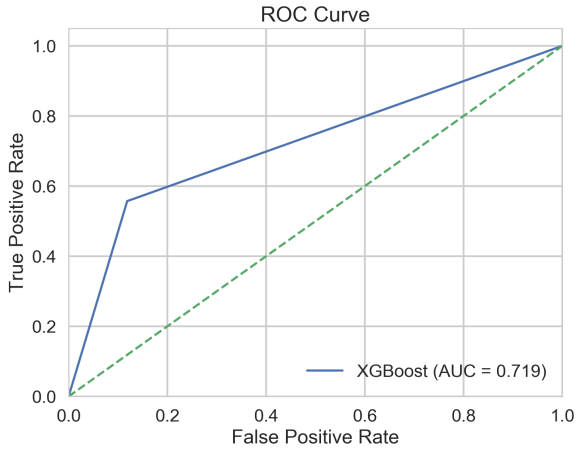

Figure 5: ROC curve for best XGBoost model

This AUC corresponded to an accuracy of **70.5%**.

With the slight increase in AUC, this model had a small boost in accuracy. The strongest XGBoost model set the maximum depth of the tree to ten, which again allowed us to easily display and interpret the tree:

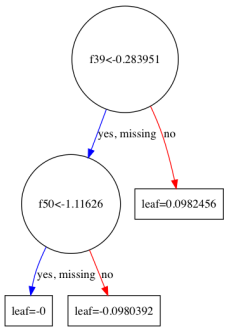

Figure 6: Best XGBoost tree

The first split uses an interaction feature. This feature is the total allelic fraction multiplied by white blood cell count. The second split also uses an interaction feature: platelet count times age. Thus far, XGBoost was the only model that had higher AUC measurements when interaction features were used, and the boosted trees clearly make use of the interaction features.

Given that there was not a significant increase in accuracy using boosted trees, we decided to try a different ensemble method for trees: random forests.

4 Random Forests

In an attempt to leverage the success of nonlinear classification trees, we then tried bagging trees using sklearn's Random Forest Classifier. With its default parameters, AUC was 0.688 (without interaction features) and 0.633 (with interaction features). Of the models explored up until this point, random forests boasted the highest default AUC.

Classification trees and random forests share many of the same hyperparameters:

- number of trees in the forest
- node impurity measure (i.e. entropy or Gini)
- maximum depth of the trees
- number of features to consider when looking for the best split
- minimum number of samples required to split a node
- minimum number of samples required to be present in each leaf
- whether or not bootstrap samples are used when building trees

After tuning the hyperparameters, the AUC was 0.741 (without interaction features) and 0.714 (with interaction features).

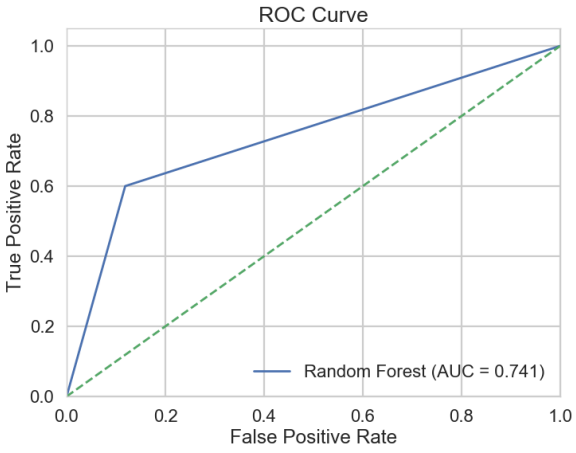

Figure 7: ROC curve for best random forest model

These were the highest AUC values achieved so far, and the accuracy, **72.9%**, was the highest accuracy attained on the test set.

Since random forests is a bagging ensemble method for trees, we could not easily examine an internal tree structure. However, it is interesting to note that as before, random forest's trees were aggressively curtailed in their complexity. The optimal hyperparameter settings allowed a maximum tree depth of only four, and 25 samples are required to split a node.

VI. Conditional Probability Models

1 Naive Bayes

Having tried several binary classification models, with the highest accuracy on test being 72.9%, we decided to approach the problem with a conditional probability model. We hypothesized that predicting probabilities rather than hard classifications would add flexibility to our model and increase accuracy.

We began with implementing sklearn's Gaussian Naive Bayes. Naive Bayes takes only one parameter: the prior probabilities of the classes. Both training and test sets were split fairly evenly (51% AML patients, 49% individuals without AML), so we set the prior distribution to (0.5, 0.5). Setting other prior distributions had no effect on the AUC measurements.

Naive Bayes achieved an AUC value of 0.662 (with and without interaction features):

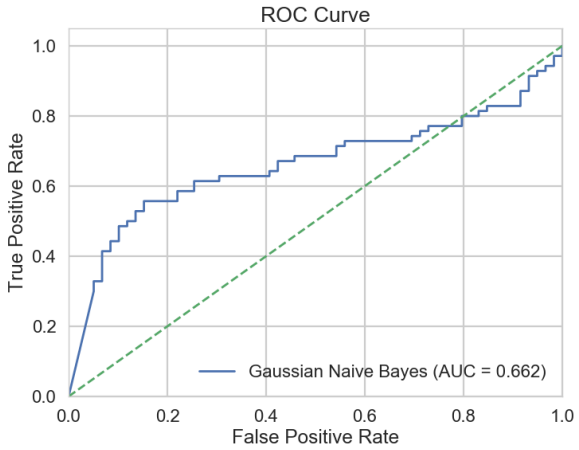

Figure 8: ROC curve for best Naive Bayes model

Because Naive Bayes is a conditional probability model, we used the ROC curve to choose a threshold value that would optimize accuracy. Several thresholds yielded the optimal accuracy of **60.5%** on the test set.

2 Logistic Regression

We next tried a different conditional probability model: sklearn's Logistic Regression. Using default parameters led to an AUC of 0.697 (without interaction features) and 0.682 (with interaction features).

We used grid search and 10-fold cross validation to test various hyperparameters pertinent to logistic regression:

- optimization solver (e.g. liblinear, saga, etc.)
- penalty (ℓ_1 or ℓ_2)
- regularization parameter
- whether or not a constant should be added to the decision function

After hyperparameter tuning, the resulting AUC was 0.743 (with interaction features) and 0.742 (without interaction features).

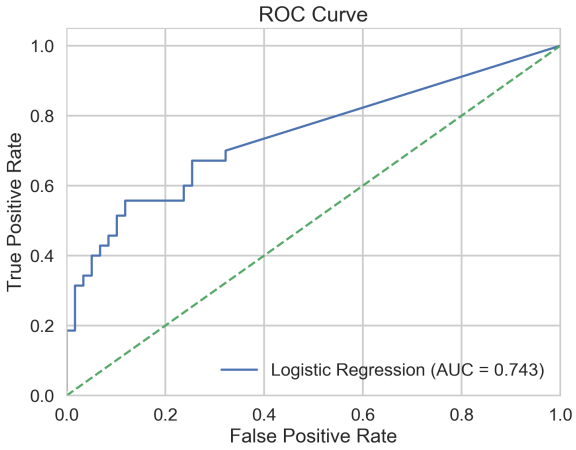

Figure 9: ROC curve for best logistic regression model

As with Naive Bayes, we used the ROC curve to find the prediction threshold that would yield the optimal accuracy. The optimal threshold was 0.436. The best accuracy attained on the test set was **71.3%**.

IV. Conclusions

In summary, the results from the six models were as follows:

Without interaction features:

	SVM	Classification Trees	XGBoost	Random Forests	Naive Bayes	Logistic Regression
AUC:	0.693	0.713	0.705	0.741	0.662	0.742
Accuracy:	67.4%	69.8%	69.8%	72.9%	60.5%	71.3%

With interaction features:

	SVM	Classification Trees	XGBoost	Random Forests	Naive Bayes	Logistic Regression
AUC:	0.659	0.690	0.719	0.714	0.662	0.743
Accuracy:	63.6%	68.2%	70.5%	70.5%	62.0%	69.0%

1 Model Analysis

From the data above, we can conclude that the Naive Bayes model was not well suited to our data. Naive Bayes relies on the assumption that features are pairwise independent. We chose to implement the method despite this since in practice, Naive Bayes works well on some datasets that do not uphold this assumption, and models typically do not need large training sets. Ultimately, Naive Bayes did not perform well.

SVM had the next lowest accuracy of the six models. From the pairwise plots in section II.3, we suspected the data was not linearly separable, but decided to experiment with various kernel functions. The sigmoid kernel performed best in both cases (with and without interaction features). It is interesting to note that the sigmoid function is a special case of the logistic function, and logistic regression performed surprising well (see discussion below). However, the overall performance of the SVM model was relatively low.

One possible explanation for this is "label noise" in the dataset. For example, the dataset labels each sample depending on whether or not the patient has AML. The dataset can contain two patients with similar input feature values, one of whom has AML (with no reported mutations) while the other does not. The first patient would be marked with a "yes" in the dataset, while the second patient would be categorized as "no" despite the two patients having similar input features values.

According to paper [4], certain models are more robust to label noise. The paper discusses how "the RF [random forest] classifier achieves high robustness to random and systematic label noise...whereas the SVM classifier is more sensitive to the kernel choice and to the input feature vectors." This offers one explanation of why the SVM model did not perform well while the random forest classifier had the highest accuracy.

Classification trees, boosted trees, and random forests attained similar accuracy scores. Because trees are nonlinear and well suited to medical screening tests, the relatively high accuracies of these models was not surprising. Of the tree-based models (and of all six models), random forests produced the highest accuracy: nearly 73%. Random forests generally reduce overfitting, and mitigate variance. In the course of hyperparameter tuning, all models needed significant regularization to prevent overfitting; this could explain the strength of the random forest model.

Finally, logistic regression performed better than expected on the dataset. The scientific community often uses logistic regression models on biomedical datasets with high-dimensionality, and continues to research how to tune and alter those models [5].

2 Feature Importance

To gain further insight into the data, we used our most successful models (random forests and logistic regression) to graph feature importance. These feature importances come from an attribute of sklearn's random forest classifier.

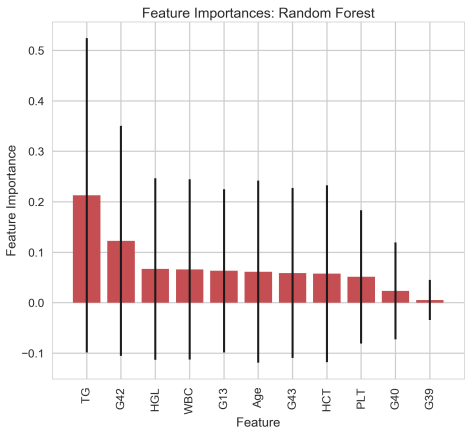

Figure 10: Random forest feature importance

The features that are not plotted have zero values. The top two features are total allelic fraction and gene 42. Many of the top ten features are blood attributes; only genes 42, 13, 43, 40, and 39 are mentioned explicitly. (Gene 43, which the optimal classification tree used as a splitting variable, ranks seventh.)

It is interesting to compare this with the top features ranked by sklearn's logistic regression model. The following is a plot of the coefficient values. Once again, the features that are not plotted have zero values.

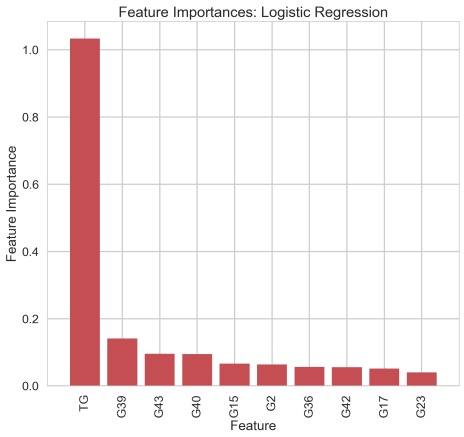

Figure 11: Logistic regression feature importance

Unsurprisingly, logistic regression also heavily emphasizes total allelic fraction. Unlike random forest feature importance, logistic regression does not use any other blood attribute. The favored genes in this model include 39, 43, 40, 15, 2, 36, 42, 17, and 23. This is almost a superset of the genes emphasized by random forest. Both models emphasize genes 39, 40, 42, and 43. Although this does not prove any correlation between these genes and AML, this observation could be compared to existing knowledge of those gene mutations in medical community.

3 Challenges

One major challenge we faced was choosing the train-test split of the dataset. We originally used the traditional 80%-20% train-test split. We then tuned a few models which performed well on validation sets (about 70% accuracy), but never exceeded 63% accuracy on the test set. We hypothesized that the 73 samples in our test set were not representative of the training data. To test this hypothesis, we tried several train-test splits (by changing the `random_state` parameter in `train_test_split()`), and saw widely varying accuracies. The accuracy depended more on the random seed than on the finely tuned hyperparameters.

Deciding how to remedy this problem was a challenge. Ultimately, we decided to increase the size of our test set to 35% of the original data to reduce variability. Of course this meant that our models could only train and cross validate on 65% of the data, which also led to a decrease in accuracy. However, we decided that the reduction in variability was worth the sacrifice in accuracy, and throughout the project we used a 65%-35% train-test split and a random seed of 8.

In general, using a small dataset (with many sparse columns) made it difficult to attain a high accuracy, but it also allowed us to examine the entire dataset manually which enhanced our understanding of the problem.

V. Future Work

The six models explored in this project certainly do not constitute an exhaustive list of models that could be applied to this problem. There are many other methods tailored to high-dimensionality datasets and bioinformatics/medical problems that we could test; see [4] and [5].

The techniques referenced in this project could also be more finely tuned. The train-test split of 65%-35% was chosen somewhat arbitrarily; in future work we could explore other means of choosing the train-test split proportion.

We could also further explore the potential of nonlinear features. The 21 interaction features referenced in section II.4 generally did not improve AUC or accuracy measures, and perhaps this was because they were also chosen arbitrarily. More scientific methods of nonlinear feature discovery exist, for example using trees or neural networks, and this would also be an interesting avenue to explore.

Aside from attempting other methods, it is important that we understand on a deeper level why some machine learning models had relatively high performance compared to others. Logistic regression's high performance was a surprise; understanding *why* it performed so well could provide more insight about our data and possible next steps.

References

[1] "Cancer Stat Facts: Leukemia - Acute Myeloid Leukemia (AML)." *National Cancer Institute*.

https://seer.cancer.gov/statfacts/html/amyl.html.

[2] Desai, Pinkal et al. "Acute Myeloid Leukemia (AML) Patients Demonstrate Increased Prevalence of AML-Defining Mutations in Peripheral Blood Years Prior to the Development of Overt Leukemia: A Case-Control Study." *Blood* 130.Suppl 1 (2017): 408. Web. 20 Mar. 2018.

[3] "Acute Myeloid Leukemia (AML)." *American Cancer Society*. https://www.cancer.org/cancer/acute-myeloid-leukemia.html.

[4] Charlotte Pelletier, et. al. Effect of Training Class Label Noise on Classification Performances for Land Cover Mapping with Satellite Image Time Series. *Remote Sensing* 9(173), February 2017.

[5] J.G. Liao, Khew-Voon Chin. Logistic Regression for Disease Classification Using Microarray Data: Model Selection in a Large p and Small n Case. *Bioinformatics* 23(15):1945-1951, August 2007.

Appendix A: Hyperparameter Tuning

When tuning the SVM model, we tested a wide range of hyperparameter values:

```
param_grid = {
    'kernel': ['rbf', 'poly', 'linear', 'sigmoid'],
    'degree': [2, 3, 4, 5, 6],
    'C': np.logspace(-4, 2, num=30),
    'gamma':np.logspace(-4, 2, num=30),
    'coef0':np.logspace(-4, 2, num=30),
    'random_state':[0]
}
```

The table below shows the results of grid search using 10-fold cross validation.

After iteratively narrowing the range of hyperparameters, we found that the optimal kernel was sigmoid, the optimal regularization parameter `C` was 87, and the optimal kernel coefficient `gamma` is 0.0212. These settings led to an AUC of 0.693.

	param_kernel	param_degree	param_C	param_gamma	param_coef0	param_random_state	mean_test_score	mean_train_score	std_test_score
2027	sigmoid	3	100.000000	0.003162	0.000100	0	0.691753	0.718912	0.070025
2047	sigmoid	4	100.000000	0.003162	0.000100	0	0.691753	0.718912	0.070025
2087	sigmoid	6	100.000000	0.003162	0.000100	0	0.691753	0.718912	0.070025
2067	sigmoid	5	100.000000	0.003162	0.000100	0	0.691753	0.718912	0.070025
2007	sigmoid	2	100.000000	0.003162	0.000100	0	0.691753	0.718912	0.070025
1333	poly	3	0.100000	3.162278	3.162278	0	0.690984	0.998585	0.062047
1233	poly	3	0.100000	3.162278	0.100000	0	0.690445	0.997646	0.071881
2401	poly	2	100.000000	0.000100	100.000000	0	0.687399	0.733732	0.072628
1033	poly	3	0.100000	3.162278	0.000100	0	0.686439	0.997646	0.069297
1133	poly	3	0.100000	3.162278	0.003162	0	0.686439	0.997646	0.069297
2127	sigmoid	3	100.000000	0.003162	0.003162	0	0.683246	0.716524	0.067079
2107	sigmoid	2	100.000000	0.003162	0.003162	0	0.683246	0.716524	0.067079
2187	sigmoid	6	100.000000	0.003162	0.003162	0	0.683246	0.716524	0.067079
2167	sigmoid	5	100.000000	0.003162	0.003162	0	0.683246	0.716524	0.067079
2147	sigmoid	4	100.000000	0.003162	0.003162	0	0.683246	0.716524	0.067079
2341	poly	4	100.000000	0.000100	3.162278	0	0.683072	0.731360	0.063013

Figure 12: Hyperparameter tuning for SVM via grid search