Leukemia Classification Based on Gene Expression

By: Dhruv Doshi, Lindsey Alexanian, & Shital Waters

Abstract:

In the realm of biomedical research, the integration of machine learning has catalyzed a transformative shift, enabling profound insights in cancer research. This project focuses on the molecular classification of leukemia based on gene expression patterns, striving to create a robust predictive tool for accurate leukemia type diagnosis. Leveraging an ensemble classification model, we initially explored diverse individual models, including KNN, Decision Tree, Logistic Regression, and Random Forest, leading to valuable insights and promising results. Notably, our final ensemble model achieved both an accuracy and f1-score of 94%, surpassing individual models and demonstrating the power of model fusion. The f1-score is as significant as precision and recall, both critical in this medical context. This work carries substantial medical implications, offering potential enhancements in timely treatment strategies. By synergizing biomedical research and machine learning, we contribute to the broader mission of advancing leukemia understanding and treatment at the molecular level.

Introduction:

In the evolving domain of biomedical research, the application of machine learning has ushered in a paradigm shift. Leveraging computational techniques, researchers are now able to analyze and interpret vast volumes of data to draw significant insights, especially in cancer research. This research specifically delves into the molecular classification of leukemia based on gene expression patterns. The intricate nature of genes and their manifestation in various diseases, particularly leukemia, necessitates a comprehensive and precise analytical approach. Our project undertakes this challenge by employing an ensemble classification model. While our initial inclination was towards a singular kNN classification model, our explorative approach encouraged us to expand our scope to include multiple classification models. This ensured an extensive assessment, fostering optimal results. The intent was not only to achieve accuracy but also to understand the nuanced behavior of various models in predicting gene expressions linked to leukemia. Through this research, we aim to furnish a robust predictive tool that can aid in the timely and accurate diagnosis of leukemia types, potentially informing treatment strategies and enhancing patient outcomes.

Dataset:

https://www.kaggle.com/datasets/crawford/gene-expression

Problem Statement:

The objective of this research is to construct an ensemble machine learning model that can accurately classify and distinguish between acute myeloid leukemia (AML) and acute lymphoblastic leukemia (ALL) based on their gene expression patterns. This classification aims to provide a more timely and precise diagnostic tool that can potentially inform appropriate treatment strategies, thereby improving patient outcomes in the realm of leukemia care.

Original Plan of Action:

We will be using the Golub et al. “Molecular Classification of Cancer: Class Discovery and Class” dataset from 1999, which is publicly available on Kaggle and several other sites. This dataset was used to classify new cases of cancer based on genetic expression; in this specific data, each sample has been classified as either acute myeloid leukemia (AML) or acute lymphoblastic leukemia (ALL).

For our project, we are planning to use the gene expression data to build a classification model that classifies each patient/sample as AML or ALL. Initially, we plan to build a kNN classification model, in which we will find the ideal value for *k*, and then we will take the average over bootstrap-resampled training samples and try to maximize the model’s accuracy. Moreover, if the results are unsatisfactory, we plan to implement other classification models as well.

Deliverables:

- Classification of each sample as AML or ALL, followed by a comparison with the actual classification.
- Evaluation results of the models using diverse scoring metrics.
- Visualization of the results to identify underlying patterns.
- Presentation of our observations and findings in the form of a comprehensive report.

Methodology:

We began by downloading the dataset, then formatting and cleaning it for use in training and testing models. This mainly involved eliminating unnecessary columns (not actual data points, but rather extra columns that did not contain useful data for our purposes) and switching the rows and columns so that each column represented a feature and each row represented a sample. There was no missing data, so there was no need to eliminate any rows or impute any data. We then split the data into training and testing data; we initially did an 80/20 split but that resulted in the size of the testing data being too small (especially when the dataset is already fairly small and it is harder to get good and reliable test results when the testing data size is so small), so we switched to a 75/25 split.

For the models themselves, we initially planned on just doing K-Nearest Neighbors (kNN) classification, but we ended up developing 4 separate classification models and then a final ensemble model that combines the lot of them. The 4 separate classification models we implemented were the following: kNN, Decision Tree, Logistic Regression, and Random Forest. We also did some parameter tuning on each individual model. We opted to not do feature selection, because there were quite a lot of features and each represented a gene, so it would be hard to choose which ones to include vs. not include. Instead, we adjusted other settings (i.e., adjusting maximum length for the Decision Tree and setting the maximum features to square-root, but not manually selecting features ourselves, or setting cross-validation for Logistic Regression, etc.).

At the end, we combined the four individual models into one final ensemble model. The ensemble model did classification based on the result of the majority of the individual models. For any ties, the final selection was randomized.

Code (brief explanation):

Formatting and cleaning the data:

We dropped the extra columns, switched rows and columns, and adjusted headers and indices to clean up the data for use in building and testing the models.

```
# drop extra columns
cols_drop = [col for col in train_data.columns if "call" in col]
cols_drop.append("Gene Description")
golub.drop(columns=cols_drop, inplace=True)

# make Gene Accession Number the index of the df
golub.set_index("Gene Accession Number", inplace=False)
# change shape (labels on y instead of x)
golub = golub.T
# change header
golub.columns = golub.iloc[0]
# drop first row
golub = golub.drop(golub.index[0])
```

Building the models:

All of the individual models utilized the scikit-learn library for Python.

kNN:

We built a kNN classifier using scikit-learn. We began with k = 3, but after doing some cross-validation testing and looking at the elbow curve, we determined that the most accurate k value would be around 9. Please see images A and B for this.

```
# initialize kNN where k=3
knn = KNeighborsClassifier(n_neighbors=3)
# fit to train data
knn.fit(X_train, y_train)

# make predictions on test data
test_pred = knn.predict(X_test)

# find accuracy on test data
test_accuracy = accuracy_score(y_test, test_pred)
print(f"Accuracy: {test_accuracy}")

%%time

# let's try some cross validation

# we'll do 20 reps
k_values = [i for i in range (1,20)]
scores = []

# scale data
scaler = StandardScaler()
X = scaler.fit_transform(golub)

# run tests for cross val scores
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    score = cross_val_score(knn, X, labels.cancer, cv=5)
    scores.append(np.mean(score))
```

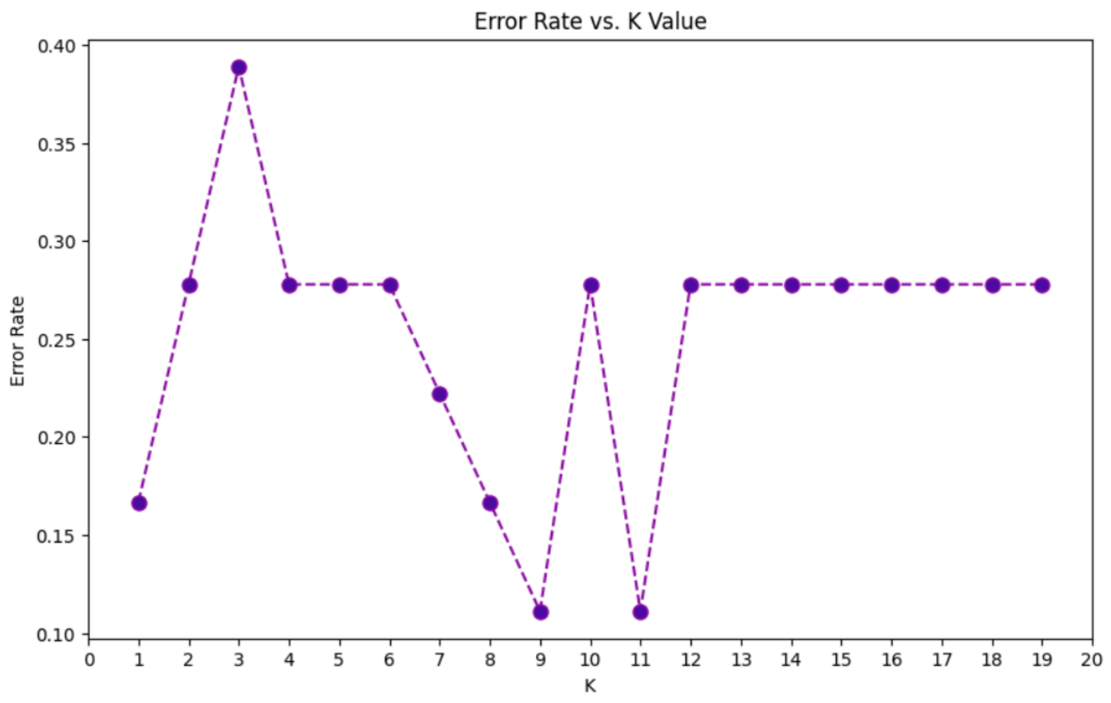

Image A: Elbow Curve for kNN classifier

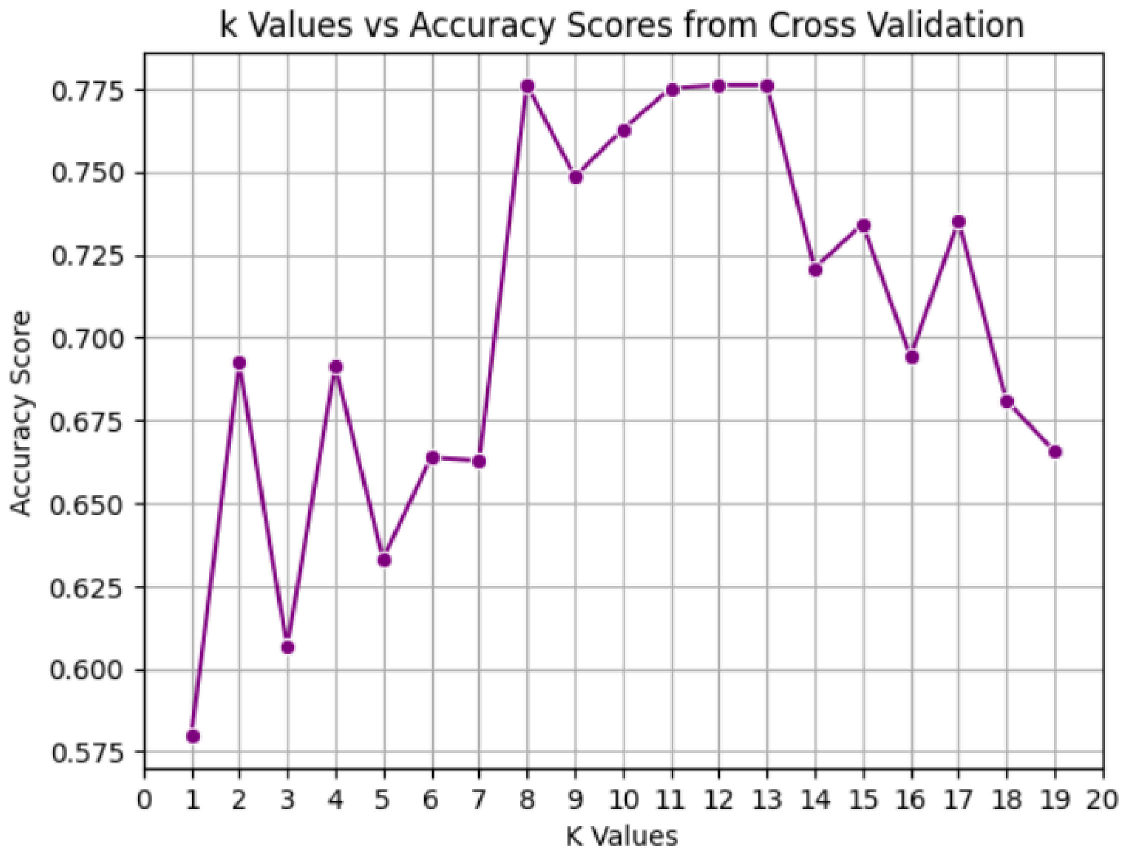

Image B: kNN Accuracy Scores (by k) with Cross Validation

We then tested for accuracy, precision, and recall with the new model.

```
# initialize kNN where k=9
knn2 = KNeighborsClassifier(n_neighbors=9)
# fit to train data
knn2.fit(X_train, y_train)

# make predictions on the test data
y_pred_knn = knn2.predict(X_test)

# find accuracy! (again)
test_accuracy2 = accuracy_score(y_test, y_pred_knn)
print(f"Accuracy: {test_accuracy2}")
```

Decision Tree:

We set the Decision Tree model to use the square root function for the maximum number of features, as this proved more accurate than other settings. We did not set a maximum length for the Decision Tree, so the model ended up having a length of 4. Please see Image C for a visualization of the Decision Tree model.

```
# create tree model
tree_model = tree.DecisionTreeClassifier(random_state=1, max_features="sqrt")

# fit model on training data
tree_model.fit(X_train, y_train)

# predictions on test data!
y_pred_dt = tree_model.predict(X_test)

# find accuracy
tree_score = accuracy_score(y_test, y_pred_dt)

# print
print(f"Test accuracy score: {tree_score}")
```

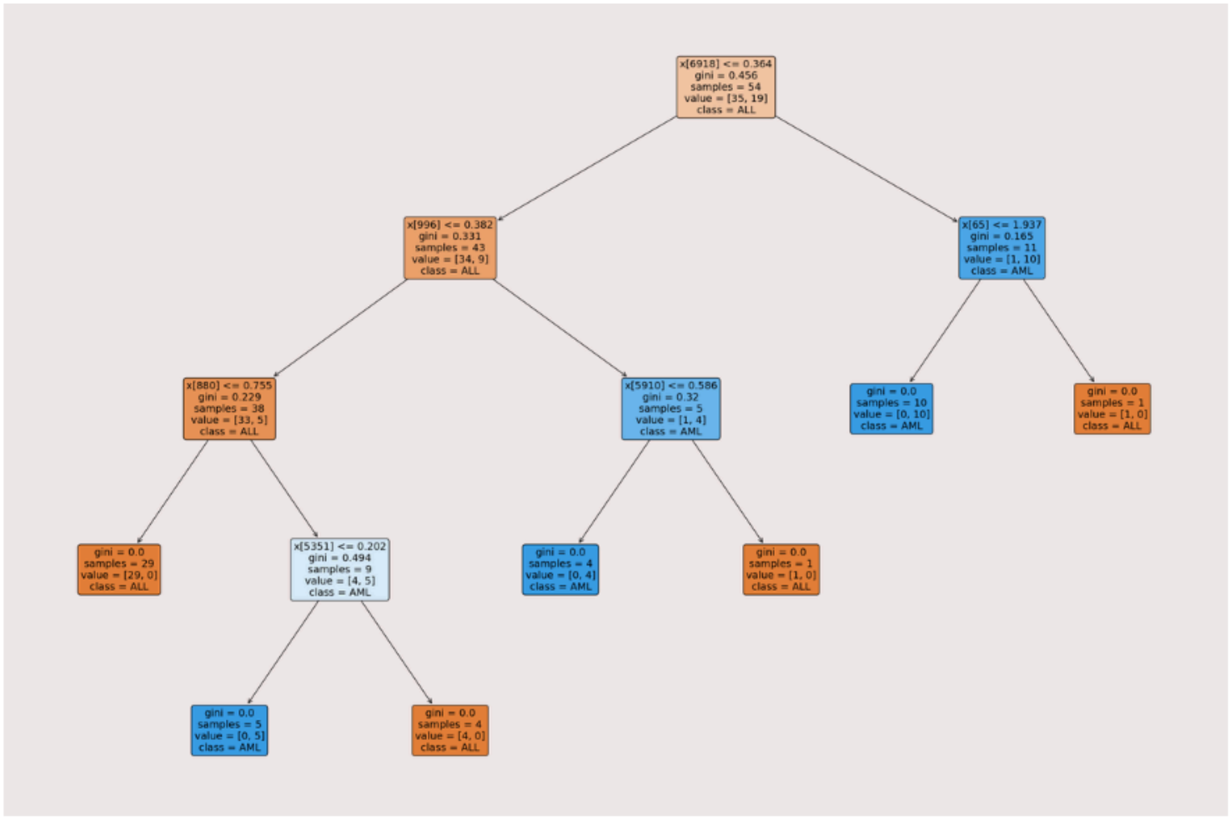

Image C: Decision Tree model

Logistic Regression:

We chose logistic regression specifically because it can work very well for datasets that have smaller sample sizes. We also did 5-fold cross-validation with a liblinear solver. This was used to train the model, the L1 Lasso and L2 Ridge were used for tuning purposes. Whichever one performed better during training is the one that is ultimately used for testing.

```
warnings.filterwarnings('ignore')

# Using grid search to find the best combination of hyperparameters
# for the logistic regression model.

# 1. Using L1 Lasso as the penalty for tuning

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1']
}

grid_search = GridSearchCV(
    LogisticRegression(random_state = 10, solver='liblinear'),
    param_grid,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy'
)
grid_search.fit(X_train, y_train)

best_lr_1 = grid_search.best_estimator_

y_pred_lr_tuned_1 = best_lr_1.predict(X_test)
accuracy_lr_tuned_1 = accuracy_score(y_test, y_pred_lr_tuned_1)
print("Tuned Logistic Regression Accuracy with L1 Lasso:", accuracy_lr_tuned_1)

# 2. Using L2 Ridge as the penalty for tuning

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': [None, 'l2']
}

grid_search = GridSearchCV(
    LogisticRegression(random_state = 10),
    param_grid,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy'
)
grid_search.fit(X_train, y_train)

best_lr_2 = grid_search.best_estimator_

y_pred_lr_tuned_2 = best_lr_2.predict(X_test)
accuracy_lr_tuned_2 = accuracy_score(y_test, y_pred_lr_tuned_2)
print("Tuned Logistic Regression Accuracy with L2 Ridge:", accuracy_lr_tuned_2)

# Compare both the accuracies. Store the predictions of the LR model which
# gives better predictions.
if accuracy_lr_tuned_1 > accuracy_lr_tuned_2:
  y_pred_lr = y_pred_lr_tuned_1
else:
  y_pred_lr = y_pred_lr_tuned_2
```

Random Forest:

We decided to go ahead with the Random Forest model because based on the evaluation metrics of the decision tree, it seemed that the model was overfitting the training data. Hence, a bagging approach could give better results. To tune this model, we used grid search and performed tuning based on number of trees, maximum depth of the trees and minimum sample split. `Grid_search.best_params_` gives us the best parameters it found by fitting each combination of the values given. Then we fit the model using the best parameters and evaluate the model using the test set. Finally, using the `sklearn.metrics` library, we get the classification report.

```
# defining param_grid
param_grid = {
   'n_estimators': [50, 100, 200],
   'max_depth': [None, 10, 20, 30],
   'min_samples_split': [2, 5, 10]
}

# performing grid search with cross-validation, to find best params
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state = 23),
                          param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# store the best params
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# build model using the best params
best_rf = RandomForestClassifier(**best_params)
best_rf.fit(X_train, y_train)

# make predictions
y_pred_rf = best_rf.predict(X_test)
accuracy_best = accuracy_score(y_test, y_pred_rf)
print(f'Best model accuracy on test data: {accuracy_best}')

# Classification report
print(sklearn.metrics.classification_report(y_test, y_pred_rf))
```

Ensemble Model:

As previously mentioned, the ensemble model utilizes the results from all four individual models. It does classification based on the result of the majority of the individual models. For any ties, the final selection was randomized.

```
# take results from all the models and make final prediction based on the
# most voted class. Breaking ties by randomizing.
```

```
def mostFreqElement(my_list):
  element_counts = Counter(my_list)
  max_frequency = max(element_counts.values())
  return random.choice([element for element, count in element_counts.items()
  if count == max_frequency])

y_pred_final = []
for i in range(0, len(y_pred_knn)):
  y_pred_final.append(mostFreqElement([y_pred_knn[i], y_pred_dt[i],  y_pred_lr[i],
y_pred_rf[i]]))
```

Analysis:

To get error metrics for any of the individual models as well as the ensemble model, we got the classification report. In scikit-learn, the classification_report() function returns the overall accuracy, as well as the precision, recall, f1-score, and support for each of the result classes. This was generated for all models, but we will only include the code for one here so as to not be redundant.

```
# Classification report
print(sklearn.metrics.classification_report(y_test, y_pred_final))
```

Results:

We implemented and tuned 4 classification models: K-Nearest Neighbors, Decision Tree, Logistic Regression and Random Forest. Here are the results obtained for each model (displaying the weighted averages):

	Accuracy	**Precision**	**Recall**	**f1-score**
KNN	0.89	0.90	0.89	0.88
Decision Tree	0.72	0.78	0.72	0.73
Logistic Regression	0.89	0.89	0.89	0.89
Random Forest	0.78	0.77	0.78	0.76
Ensemble	**0.94**	**0.95**	**0.94**	**0.94**

We can observe that the ensemble of models gives the best results when compared to each of the models individually.

Discussion:

Preparing the gene generation data for using it as an input for the machine learning models was one of the important parts of this project. This included feature selection, reindexing and transposing to name a few steps. Performing these steps took time and proved to be fruitful as we started implementing models. We did not face many errors while inputting the data into the models.

K-nearest neighbors (KNN) was the first model we implemented. By directly inputting the training data, setting k = 3 as the default value, we achieved an accuracy of 61% which was not satisfactory. This means that the default parameters of the model are not giving a good accuracy score and we need to tune the model. After using cross-validation and trying out different values for k, using the elbow curve, we observed that we are getting a good accuracy and lowest error rate for k = 9. Hence, we got an accuracy = 89%, precision = 90% and recall = 89% which looks good but if we check the recall for 'AML' class, it is just 67%. This made us try other models for classifying a gene expression as 'ALL' or 'AML'.

Decision tree gave a training accuracy of 100% which is a good start. But when we evaluated the model using test data, we got accuracy = 72%, precision = 78% and recall = 72%. Based on these metrics, we observed that the decision tree model is overfitting the training data and hence we implemented a random forest model to try and eliminate the overfitting issue.

Random forest also gave a training accuracy of 100%. After using test data to evaluate the model, we got accuracy = 78%, precision = 77% and recall = 78% after hypertuning, which is an improvement over the decision tree model but yet not better than KNN model.

Next, we tried the Logistic Regression classifier. Upon evaluation using the test set, we got results which were similar to the KNN model. We got accuracy = 89%, precision = 89% and recall = 89%. Finally, we decided to ensemble the predictions of all the models and evaluate it against the true labels. Using the ensemble predictions, we got accuracy = 94%, precision = 95% and recall = 94% which are the best scores so far. For the 'ALL' class, we got precision = 92% and recall = 92% and for the 'AML' class, we got precision = 83% and recall = 83%. This is an improvement over the low recall for 'AML' class when using only the KNN model.

For our case of classification in the medical domain, it is important to analyze the consequences of false positives and false negatives. If the consequences of false positives are significant then precision is the metric to prioritize. But if the consequences of false negatives are significant then recall is the metric to prioritize. But in our case, we believe that both precision and recall are equally important. This is because if a patient is predicted to have ‘AML’ when they actually have ‘ALL’ or if a patient is predicted to have ‘ALL’ when they actually have ‘AML’, it could lead to unnecessary medical procedures, treatments, and emotional distress for patients and their families. It will also lead to delay of the correct medical procedure for the actual disease. In these cases, timing of the treatment is usually critical. As in many medical scenarios, there's a trade-off between precision and recall. Increasing one often results in a decrease in the other. Hence, it is important to give equal importance to both and to do so we can focus more on the f1-score, which is the harmonic mean of precision and recall. The f1-score is useful when we want to consider both false positives and false negatives.

Future work:

The promising results achieved in our current study pave the way for several further avenues of exploration. As we navigate the intricate landscape of biomedical research and machine learning integration, the following represent our envisioned directions for future work:

Exploration of Other Models: While our ensemble model proved effective, the rapidly evolving nature of machine learning introduces new algorithms and techniques regularly. We believe it would be beneficial to investigate other classification models, such as Support Vector Machines, Neural Networks, or Gradient Boosted Trees, to evaluate their efficacy against the leukemia classification problem.

Bagging and Boosting: These ensemble techniques hold the potential to further refine our results. Bagging could help reduce variance by introducing random subsets of the data, whereas boosting could increase our model's accuracy by adjusting for previous mistakes. An application of techniques like AdaBoost or XGBoost might yield even better predictive results.

Collecting More Data: Our current dataset, while comprehensive, can always benefit from augmentation. With more data samples, the model is likely to be more robust, reducing potential biases and enhancing its generalizability. Partnering with medical institutions to acquire more patient gene expression data can be a strategic step.

Deep Learning Approaches: Given the complexity of gene expression patterns, deep learning models, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), can capture more intricate patterns and dependencies in the data. Exploring these architectures might offer better insights into the intricate relationships of gene expressions.

Conclusion:

The field of biomedical research is rapidly expanding, and our endeavor into the classification of leukemia based on gene expression patterns has presented both challenges and promising outcomes. Utilizing machine learning techniques, we were able to sift through vast gene expression data to discern patterns and classify samples into acute myeloid leukemia (AML) or acute lymphoblastic leukemia (ALL) with high accuracy.

Our initial approach with K-Nearest Neighbors (KNN) provided valuable insights, but it was the ensemble model that truly demonstrated the capability of merging multiple models for enhanced results. With an impressive accuracy of 94%, the ensemble model outperformed each of the individual models, underlining the potential of combining the strengths of diverse algorithms.

The medical implications of our work are profound. Accurate and timely diagnosis is paramount in cancer treatment, and the predictive tool we've developed holds the potential to significantly influence patient outcomes. While our results are promising, they also underscore the importance of precision and recall in the medical domain, where both false positives and false negatives can have consequential implications on treatment strategies and patient well-being.

In the broader context, this research exemplifies the potent union of biomedical research and machine learning. As technology continues to advance, it is increasingly evident that the integration of these two domains can usher in revolutionary tools and methodologies that can redefine our understanding and treatment of complex diseases. Looking ahead, we are optimistic about further refining our models and exploring even more sophisticated algorithms to contribute to the larger mission of combating and understanding leukemia at a molecular level.

References:

Bansal, I. (2022). K-Nearest Neighbors (KNN) in Python. *DigitalOcean*. https://www.digitalocean.com/community/tutorials/k-nearest-neighbors-knn-in-python

Dinger, S.C., Van Wyk, M.A., Carmona, S. *et al.* (2012). Clustering gene expression data using a diffraction‐inspired framework. *BioMed Eng OnLine* 11, 85. https://doi.org/10.1186/1475-925X-11-85

Galarnyk, M. (2022, April 27). Logistic Regression using Python (scikit-learn) - Towards Data Science. *Medium*.

https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

Golub et al. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science*, Vol. 286:531-537. https://www.kaggle.com/datasets/crawford/gene-expression

Navlani, A. (2023). *Decision Tree Classification in Python Tutorial*. https://www.datacamp.com/tutorial/decision-tree-classification-python

Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python, *JMLR* Vol. 12, pp. 2825-2830.

Shafi, A. (2023). *Random Forest Classification with Scikit-Learn*. https://www.datacamp.com/tutorial/random-forests-classifier-python

Annex-A:

This Annex has a list of details regarding the individual contribution of each member of the project group.

Team Member	**Contributions**
Lindsey	Collected, cleaned, and formatted the dataset. Wrote, tested, and debugged the code. Trained and tested all the individual models, then did hyperparameter tuning and overall analysis. Wrote the Methods and Code sections, and contributed to the Original Plan of Action as well as the Deliverables. Wrote and formatted the citations.
Shital	Helped develop and debug code for Random Forest, Logistic Regression, and Ensemble models. Authored the Introduction, Problem Statement, Future Work, and Conclusion sections, and contributed to both the Original Plan of Action and Deliverables.

Dhruv	Wrote, tested, and debugged the code. Trained and tested Random Forest, Logistic Regression and Ensemble models, followed by parameter tuning and overall analysis. Added formatting and comments in the code file. Wrote the Abstract, Results and Discussion sections, and contributed to the Original Plan of Action as well as the Deliverables.