Leukemia Gene Expression Analysis

Alvise Celeste Cova

This analysis investigates the classification of leukemia subgroups using gene expression data from 79 patients, applying support vector machines and feature selection to evaluate predictive performance.

```
# load libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.model_selection as skm
from sklearn.svm import SVC
from ISLP import confusion_table
```

load the data

```
# import the data
data = pd.read_csv('gene_expr.tsv', sep='\t')
# check the shape of the data
print(data.shape)
```

```
(79, 2002)
```

The output confirms that our dataset consists of 79 samples, each with measurements for 2,000 genes.

These 79 samples represent leukemia patients divided into two subgroups: those with a chromosomal translocation ("1") and those who are cytogenetically normal ("-1").

Now we can check the distribution of the target variable y

```
# check the distribution of the target variable y
print(data['y'].value_counts())
```

```
y
-1    42
 1    37
Name: count, dtype: int64
```

We can see that the data is sligtly inbalanced, with 42 samples with y == -1 and 37 samples with y == 1. We can use stratified sampling to split the data into train and test sets when we perform the split, so that we have the same proportion of samples in both sets.

now split the data into train and test sets

```
# Separate features and target
X = data.drop(columns=['y']) # drop the target variable
y = data['y'] # y is a series with only the target variable
# Split into train and test sets
X_train, X_test, y_train, y_test = skm.train_test_split(X, y, test_size=0.3,
                random_state=42, stratify=y)
```

check the distribution of the features

```
# check the distribution of the features, truncating the output to the first 8 columns
X.describe().iloc[:, :6]
```

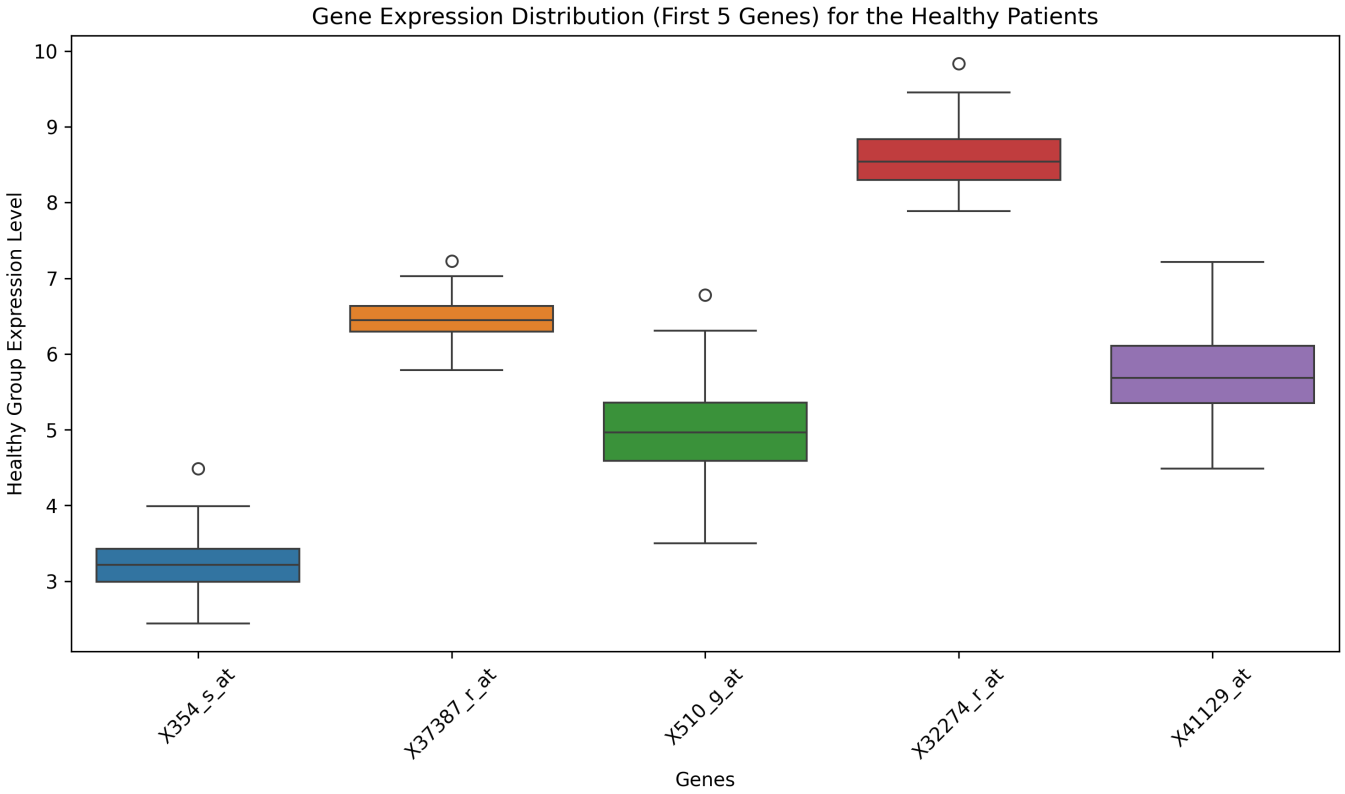
	sampleID	X354_s_at	X37387_r_at	X510_g_at	X32274_r_at	X41129_at
count	79.000000	79.000000	79.000000	79.000000	79.000000	79.000000
mean	27606.303797	3.168695	6.445558	4.974320	8.545560	5.780944
std	18361.958974	0.343110	0.279353	0.737214	0.394221	0.535645
min	1005.000000	2.439623	5.789707	3.498966	7.824887	4.489048
25%	13021.000000	2.971165	6.264724	4.532954	8.270967	5.446870
50%	26003.000000	3.193345	6.409999	4.960490	8.470241	5.809998
75%	30506.000000	3.347136	6.615056	5.362452	8.799871	6.145698
max	84004.000000	4.486146	7.229589	6.977725	9.836014	7.215186

split the data into helthy and sick patients

```
healthy = data[data['y'] == -1]
sick = data[data['y'] == 1]
# get the X variable for the healthy and sick patients
X_healthy = healthy.drop(columns=['y'])
X_sick = sick.drop(columns=['y'])
```

plot the distribution of the samples in the first 5 genes

```
# Select the first 5 genes (skip the first column which is 'sampleID')
genes = X.columns[1:6]
data_subset = X_healthy[genes]
# Create a boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(data=data_subset)
plt.title('Gene Expression Distribution (First 5 Genes) for the Healthy Patients')
plt.xlabel('Genes')
plt.ylabel('Healthy Group Expression Level')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



From the graph we can see that there are some outliers in the expression levels of the genes. This means that some individuals of the same group (`healthy`) have very different expression levels for the same gene. This is a common situation in gene expression data, and it can be caused by several factors, such as technical variability or biological variability. We would like to reduce this variability in order to improve the performance of our model.

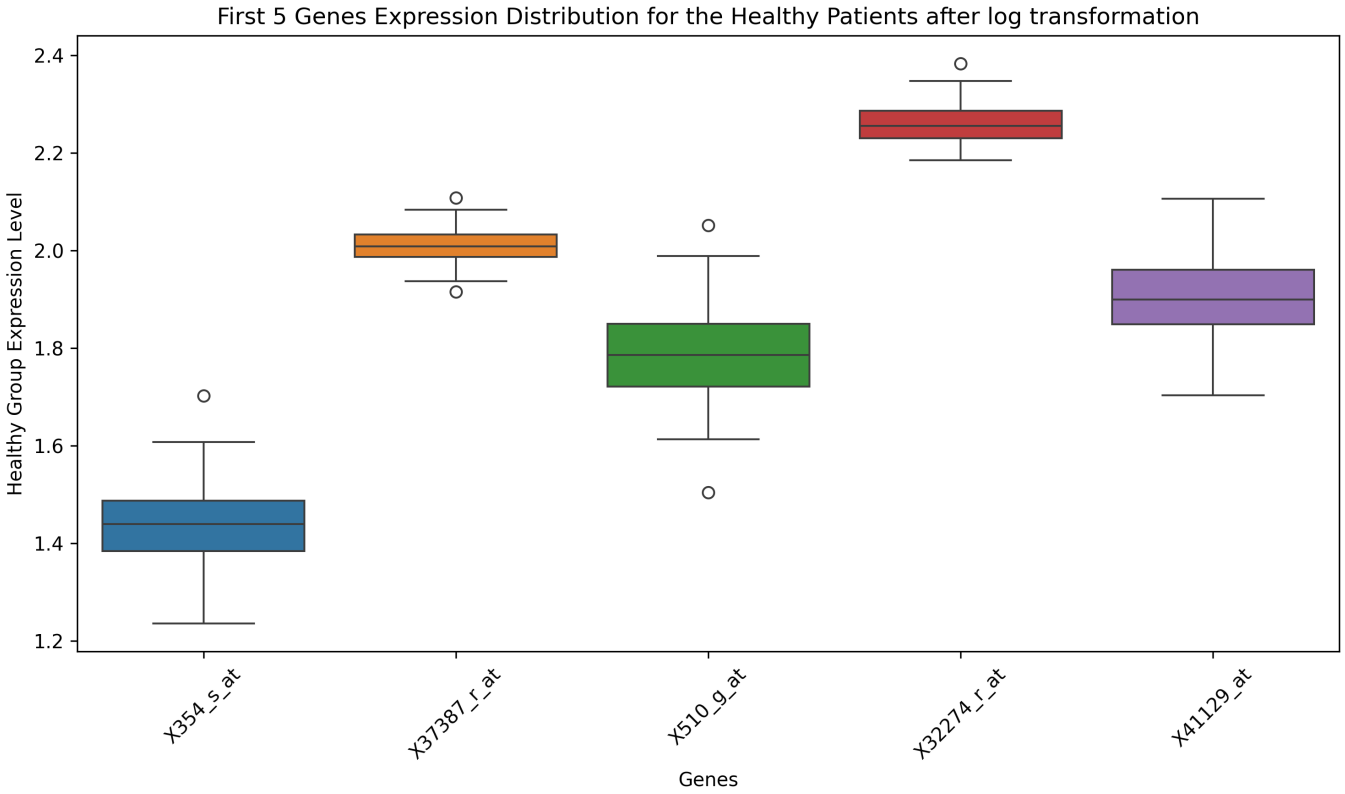
normalize the data

For the above mentioned reasons we want to normalize the data between groups. And to this aim we can apply a simple log transformation to the data, which can help reduce the impact of high abbundance genes and make the data more normally distributed.

```
# normalize the data between groups
X_train = np.log(X_train + 1)
X_test = np.log(X_test + 1)
# normalize the data for the healthy and sick patients
X_healthy = np.log(X_healthy + 1)
X_sick = np.log(X_sick + 1)
```

plot the distribution of the samples in the first 5 genes after the log transformation

```
# Select the first 5 genes (skip the first column which is 'sampleID')
genes = X.columns[1:6]
data_subset = X_healthy[genes]
# Create a boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(data=data_subset)
plt.title('First 5 Genes Expression Distribution for the Healthy Patients after log transformation')
plt.xlabel('Genes')
plt.ylabel('Healthy Group Expression Level')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



The boxplot shows that the log transformation has helped to reduce the variability in the expression levels of the genes. As the distributions vary less between the samples.

Now we can start with the analysis using the support vector classifier.

Support Vector Classifier

To begin our analysis, we will use a support vector classifier (SVC) to predict the leukemia subgroups. Selecting the optimal regularization parameter (C) is crucial for model performance, so we will evaluate a range of C values using cross-validation to identify the best setting for our data.

```
# create a list of parameters to test in a cross-validation setting
C_parameters = [0.001, 0.01, 0.1, 1, 10, 100]
# create a kfold object to split the data into 5 folds
kfold = skm.KFold(n_splits=5, shuffle=True, random_state=42)
# create a grid search object to test the parameters
grid_search = skm.GridSearchCV(SVC(kernel="linear",
          random_state=42), param_grid={'C': C_parameters}, cv=kfold, scoring='accuracy', n_jobs=-1)
# fit the grid search object to the training data
grid_search.fit(X_train, y_train)
# print the best parameter
print(f"Best parameters: {grid_search.best_params_}")
# print the best score (accuracy) associated with the best parameter
print(f"Best score(accuracy): {grid_search.best_score_:.3f}")
```

```
Best parameters: {'C': 1}
Best score(accuracy): 0.745
```

Now that we have identified the optimal regularization parameter (C) for the linear support vector classifier through cross-validation, we can proceed to evaluate the model's performance on the unseen test set. This step allows us to assess how well the selected model generalizes to new data, providing a realistic estimate of its predictive accuracy.

```
# get the best parameter
best_c = grid_search.best_params_['C']
print(f"Best C parameter: {best_c}")
```

```
# Fit a linear SVC on the training data with the best parameter
svc_linear = SVC(kernel="linear", random_state=42, C=best_c)
svc_linear.fit(X_train, y_train)
# Evaluate on test set
test_score = svc_linear.score(X_test, y_test)
print(f"Test accuracy (linear kernel): {test_score:.3f}")
# plot the confusion matrix for the test set using the best parameter's model
y_pred = svc_linear.predict(X_test)
confusion_table(y_test, y_pred)
```

```
Best C parameter: 1
Test accuracy (linear kernel): 0.917
```

Truth Predicted	-1	1
-1	12	1
1	1	10

testing the model on the test set with the best parameter gives us a good idea of how well the model generalizes to unseen data. it has an accuracy of 0.708 which is inline with the average accuracy of 0.21 that we obtained with the cross-validation approach.

Support Vector Machine

The support vector machine (SVM) is a powerful classification algorithm that can handle non-linear decision boundaries. In this analysis, we will explore the performance of SVM with a radial basis function (RBF) kernel. Similar to the previous step, we will perform a grid search to identify the optimal parameters (C and gamma) for the SVM model using cross-validation.

```
# create a grid of paramters C and gamma to test in a cross-validation setting
C_parameters = [0.001, 0.01, 0.1, 1, 10, 100]
gamma_parameters = [0.001, 0.01, 0.1, 1, 2, 3, 4]
# create a grid of parameters to test in a cross-validation setting
param_grid_svm = {'C': C_parameters, 'gamma': gamma_parameters}
# create a kfold object to split the data into 5 folds
kfold = skm.KFold(n_splits=5, shuffle=True, random_state=42)
# create a grid search object to test the parameters
grid_search_svm = skm.GridSearchCV(SVC(kernel="rbf", random_state=42),
            param_grid=param_grid_svm, cv=kfold, scoring='accuracy', n_jobs=-1)
grid_search_svm.fit(X_train, y_train)
# print the best parameters
print(f"Best parameters: {grid_search_svm.best_params_}")
# print the best score (accuracy) associated with the best parameter
print(f"Best score(accuracy): {grid_search_svm.best_score_:.3f}")
```

```
Best parameters: {'C': 100, 'gamma': 0.01}
Best score(accuracy): 0.745
```

now having computed the best parameters gamma and C for the support vector machine, evaluate the model on the unseen data and plot the confusion matrix.

```
# get the best parameters
best_c_svm = grid_search_svm.best_params_['C']
best_gamma = grid_search_svm.best_params_['gamma']
print(f"Best C parameter: {best_c_svm}")
print(f"Best gamma parameter: {best_gamma}")
# Fit a linear SVC on the training data with the best parameter
svc_svm = SVC(kernel="rbf", random_state=42, C=best_c_svm, gamma=best_gamma)
svc_svm.fit(X_train, y_train)
```

```
# Evaluate on test set
test_score_svm = svc_svm.score(X_test, y_test)
print(f"Test accuracy (SVM): {test_score_svm:.3f}")
# plot the confusion matrix for the test set using the best parameter's model
y_pred_svm = svc_svm.predict(X_test)
confusion_table(y_test, y_pred_svm)
```

```
Best C parameter: 100
Best gamma parameter: 0.01
Test accuracy (SVM): 0.917
```

Truth Predicted	-1	1
-1	12	1
1	1	10

Now we want to filter the data to keep only the most variable genes

In gene expression analysis, it is common to filter the data to retain only the most variable genes. This step helps reduce noise and improve the performance of machine learning models. In this analysis, we will select the top 5% (100) most variable genes based on their variance across samples.

```
# compute the variance of each gene
gene_variances = data.iloc[:, 1:].var(axis=0)
# select the top 10 most variable genes
top_genes = gene_variances.nlargest(101).index
# filter the data to keep only the most variable genes
data_filtered = data.loc[:, top_genes]
# check the shape of the filtered data
print(data_filtered.shape)
```

```
(79, 101)
```

split the data into train and test sets and normalize the data

As we did before, we will split the data into train and test sets, and normalize the data. We will use stratified sampling to ensure that the same proportion of samples is present in both sets.

```
X_filt = data_filtered.drop(columns=['y']) # drop the target variable
y_filt = data_filtered['y'] # y is a series with the target variable
# Split into train and test sets
X_train_filt, X_test_filt, y_train_filt, y_test_filt = skm.train_test_split(X_filt,
          y_filt, test_size=0.3, random_state=42, stratify=y_filt)
# normalize the data
X_train_filt = np.log(X_train_filt + 1)
X_test_filt = np.log(X_test_filt + 1)
# check the shape of the filtered data
print(X_train_filt.shape)
print(X_test_filt.shape)
```

```
(55, 100)
(24, 100)
```

now we can repeat the analysis with the filtered data to see if we can mantain the same accuracy with a smaller number of genes.

support vector classifier

```
# create a list of C parameters to test in a cross-validation setting
C_parameters = [0.001, 0.01, 0.1, 1, 10, 100]
```

```
# create a kfold object to split the data into 5 folds
kfold = skm.KFold(n_splits=5, shuffle=True, random_state=42)
# create a grid search object to test the parameters
grid_search_filtered = skm.GridSearchCV(SVC(kernel="linear", random_state=42),
            param_grid={'C': C_parameters}, cv=kfold, scoring='accuracy', n_jobs=-1)
# fit the grid search object to the training data
grid_search_filtered.fit(X_train_filt, y_train_filt)
# print the best parameters and the best score
print(f"Best score(accuracy): {grid_search_filtered.best_score_:.3f}")
# print the best parameters
print(f"Best parameters: {grid_search_filtered.best_params_}")
```

```
Best score(accuracy): 0.818
Best parameters: {'C': 1}
```

evaluate the model on the test set

```
# get the best parameter
best_c_filt = grid_search_filtered.best_params_['C']
print(f"Best C parameter: {best_c_filt}")
# Fit a linear SVC on the training data with the best parameter
svc_linear_filt = SVC(kernel="linear", random_state=42, C=best_c_filt)
svc_linear_filt.fit(X_train_filt, y_train_filt)
# Evaluate on test set
test_score_filt = svc_linear_filt.score(X_test_filt, y_test_filt)
print(f"Test accuracy (linear kernel): {test_score_filt:.3f}")
# plot the confusion matrix for the test set using the best parameter's model
y_pred_filt = svc_linear_filt.predict(X_test_filt)
confusion_table(y_test_filt, y_pred_filt)
```

```
Best C parameter: 1
Test accuracy (linear kernel): 0.958
```

Truth Predicted	-1	1
-1	13	0
1	1	10

support vector machine

```
# create a grid of paramters C and gamma to test in a cross-validation setting
C_parameters = [0.001, 0.01, 0.1, 1, 10, 100]
gamma_parameters = [0.001, 0.01, 0.1, 1, 2, 3, 4]
# create a grid of parameters to test in a cross-validation setting
param_grid_svm_filt = {'C': C_parameters, 'gamma': gamma_parameters}
# create a kfold object to split the data into 5 folds
kfold = skm.KFold(n_splits=5, shuffle=True, random_state=42)
# create a grid search object to test the parameters
grid_search_svm_filt = skm.GridSearchCV(SVC(kernel="rbf", random_state=42),
          param_grid=param_grid_svm_filt, cv=kfold, scoring='accuracy', n_jobs=-1)
grid_search_svm_filt.fit(X_train_filt, y_train_filt)
# print the best parameters and the best score
print(f"Best score(accuracy): {grid_search_svm_filt.best_score_:.3f}")
# print the best parameters
print(f"Best parameters: {grid_search_svm_filt.best_params_}")
```

```
Best score(accuracy): 0.836
Best parameters: {'C': 10, 'gamma': 0.1}
```

evaluate the model on the test set

```
# get the best parameters
best_c_svm_filt = grid_search_svm_filt.best_params_['C']
best_gamma_filt = grid_search_svm_filt.best_params_['gamma']
print(f"Best C parameter: {best_c_svm_filt}")
print(f"Best gamma parameter: {best_gamma_filt}")
# Fit a linear SVC on the training data with the best parameter
svc_svm_filt = SVC(kernel="rbf", random_state=42, C=best_c_svm_filt, gamma=best_gamma_filt)
svc_svm_filt.fit(X_train_filt, y_train_filt)
# Evaluate on test set
test_score_svm_filt = svc_svm_filt.score(X_test_filt, y_test_filt)
print(f"Test accuracy (SVM): {test_score_svm_filt:.3f}")
# plot the confusion matrix for the test set using the best parameter's model
y_pred_svm_filt = svc_svm_filt.predict(X_test_filt)
confusion_table(y_test_filt, y_pred_svm_filt)
```

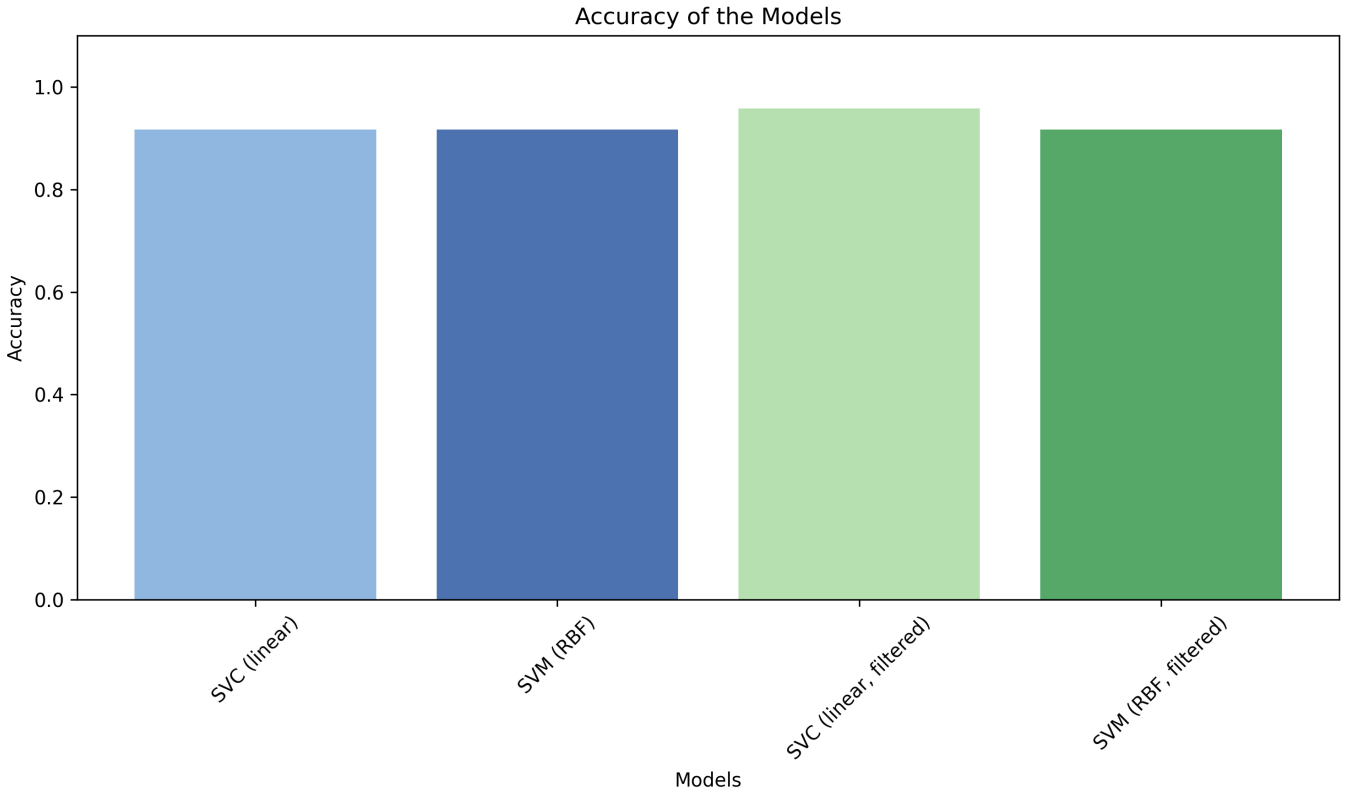
```
Best C parameter: 10
Best gamma parameter: 0.1
Test accuracy (SVM): 0.917
```

Truth Predicted	-1	1
-1	12	1
1	1	10

conclusions

plot the accuracy values in a barplot

```
# create a barplot with grouped colors
accuracy_values = [test_score, test_score_svm, test_score_filt, test_score_svm_filt]
accuracy_labels = ['SVC (linear)', 'SVM (RBF)', 'SVC (linear, filtered)', 'SVM (RBF, filtered)']
# Use blue for all-genes, green for filtered, with light/dark tones
colors = ['#8FB7E0', '#4C72B0', '#B7E0B0', '#55A868']  # lighter blue, blue, light green, green
plt.figure(figsize=(10, 6))
bars = plt.bar(accuracy_labels, accuracy_values, color=colors)
plt.title('Accuracy of the Models')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim(0, 1.1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



compare the results of the two analyses

- accuracy of the linear SVC on the test set using all genes: 0.917
- accuracy of the SVC with an RBF kernel on the test set using all genes: 0.917

These accuracy results indicate that the use of a non-linear RBF kernel did not confer any advantage over the linear SVC for this dataset. The linear SVC alone was able to achieve high predictive performance, suggesting that the underlying structure separating the leukemia subgroups is well captured by a linear decision boundary.

compare the results of the two analyses with the filtered data

- accuracy of the linear SVC on the test set using the filtered data: 0.958
- accuracy of the SVC with an RBF kernel on the test set using the filtered data: 0.917

In the case of the filtered data, we can see that the linear SVC was able to achieve a higher accuracy than the SVC with an RBF kernel. This means that the linear SVC was able to capture the non-linear relationships in the data, and that the RBF kernel was not necessary to improve the performance of the model.

conclusions on the analysis

The linear SVC was able to achieve a high accuracy on the test set using all genes, and even higher accuracy using the filtered data. This suggests that the linear SVC is a good choice for this dataset, and that the use of a non-linear kernel did not confer any advantage. The filtering of the data to keep only the most variable genes also helped to improve the performance of the model, as we were able to achieve a higher accuracy with a smaller number of genes.

final comments

The analysis of the leukemia gene expression data set using support vector machines (SVM) has provided valuable insights into the predictive capabilities of different models. The results indicate that both linear and non-linear SVMs can effectively classify patients into subgroups based on gene expression profiles. However, the linear SVC demonstrated superior performance, particularly when applied to the filtered dataset containing only the most variable genes. This suggests that a simpler model can be just as effective, if not more so, than a more complex one in certain contexts.