Statistical learning: Homework #3

Sofia Pietrini

18/05/2025

1 Introduction

In this homework we will analyze data coming from a gene expression dataset of 79 patients with leukemia belonging to two subgroups: patients with a chromosomal translocation ("1") and patients cytogenetically normal ("-1"). The *gene_expr.tsv* file contains expression for 2,000 genes and additional columns with patient labels and the outcome variable (y).
The aim is to perform a supervised analysis for prediction of the subgroups using support vector machines.

2 Data exploration and pre-processing

The analysis heavily relies on `NumPy` for numerical operations and `Pandas` for data manipulation. Core machine learning functionalities are provided by `scikit-learn`, including support vector machines (SVC), model selection tools like `GridSearchCV`, and metrics such as `RocCurveDisplay`. The `ISLP` library is utilized for specific tasks like data loading, generating confusion tables, and dedicated SVM plotting functions.

The dataset consists of 79 rows, each representing a patient, and 2002 columns, including expression measurements for 2000 genes along with two additional columns for patient labels and the outcome variable. The structure of the dataset remains unmodified, as there are no missing values that require imputation or removal.

[2]:
```
dataf = pd.read_csv('gene_expr.tsv', sep='\t') #read tsv file
dataf.shape #check data set dimension
```

[2]:
```
(79, 2002)
```

[3]:
```
dataf.isnull().values.any() #check for any missing value
```

[3]:
```
False
```

Given the nature of gene expression data, it is not unusual to observe a high dimensionality, with the number of features far exceeding the number of observations. While this characteristic can facilitate the identification of hyperplanes that effectively separate classes, it poses significant challenges for data visualization. Visualizing high-dimensional data typically necessitates dimensionality reduction techniques such as **Principal Component Analysis (PCA)**. However, by choosing not to apply these techniques, we will likely not be able to visually explore the data distribution and the hyperplanes derived from subsequent analyses.

3 Linear support vector machine

To predict the presence or absence of the specific chromosomal feature via supervised learning, we employ SVMs. A **key consideration** is the dataset's high dimensionality (79 samples, 2000 genes). In such high-dimensional spaces, data points are often more easily separated by a **linear boundary**. Consequently, a linear SVM kernel is hypothesized to be effective, potentially avoiding the need for more complex RBF or polynomial kernels. This section focuses on implementing and evaluating the linear SVM. Later sections will compare its performance against the non-linear alternatives.

As a first step, we partition the data into the feature matrix `X` (gene expressions) and the target vector `y`, which comprised 42 samples labeled *-1* and 37 samples labeled *1*, denoting the balance of the data set with respect to the outcome variable.

[4]:
```
y = dataf['y'] #select only the column of outcome variable
X = dataf.drop(columns=['y', 'sampleID'], axis=1) #drop the additional columns␣
 ↪of outcome variable and patient labels
X.shape
```

[4]:
```
(79, 2000)
```

[5]:
```
y.value_counts()
```

[5]:
```
y
-1    42
 1    37
Name: count, dtype: int64
```

Subsequently, the dataset (both features `X` and target `y`) was manually divided into training and testing sets using an approximate **50/50** split, this splitting process is required to build the confusion matrix and the ROC curve. This resulted in a training set (`X_train`, `y_train`) containing 39 observations, comprising 21 samples labeled *-1* and 18 samples labeled *1*. The remaining 40 observations were allocated to the test set.

[6]:
```
#manual split of test set and training set using skm
X_train, X_test, y_train, y_test = skm.train_test_split(X, y, test_size=0.5,␣
 ↪random_state=0)
X_train.shape
```

[6]:
```
(39, 2000)
```

[7]:
```
y_train.value_counts()
```

[7]:
```
y
 1    21
-1    18
Name: count, dtype: int64
```

We proceed by fitting the linear SVM model (`kernel='linear'`). For this initial fit, the regularization parameter C is set to a placeholder, as its optimal value will be determined later through a

dedicated hyperparameter tuning process (cross-validation).

[8]:
```
svm_linear = SVC(C=10, kernel="linear")
```

We applied **10-fold cross-validation** to tune the regularization parameter of the linear SVM. In the `scikit-learn` library, cross-validation is managed using splitter generators from the `model_selection` submodule, such as the `KFold(n_splits)` class, which implements K-Fold cross-validation.
To perform an exhaustive search over parameter values, we used the `GridSearchCV(model, param_grid, cv, ...)` function, which integrates cross-validation into the grid search process. By testing a pre-selected range of values, we were able to access the cross-validation metrics for each of the models stored in the attribute `grid.cv_results_`.

[ ]:
```
C_values = [1e-4, 1e-3, 0.01, 0.1, 1, 10, 100, 1e3, 1e4]
#create a ten folds cross validation object
kfold = skm.KFold(10, random_state=0, shuffle=True)
#create the gridsearch
grid = skm.GridSearchCV(svm_linear,
                        {"C": C_values},
                        cv=kfold,
                        scoring="accuracy") #use accuracy as scoring parameter
grid.fit(X,y)
print("Optimal linear model: ", grid.best_estimator_) #best par. combination
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

```
Optimal linear model:  SVC(C=0.1, kernel='linear')
Cross-validation accuracy of optimal model:  0.8625
```

As shown above, `C=0.1` results in the highest cross-validation accuracy of **0.8625**.
We proceeded by extracting the best model, corresponding to the optimal `C` value, stored in the attribute `best_estimator_`, and then fitting the optimal linear SVM on the training set.

[10]:
```
SVC(C=0.1, kernel='linear')
```

The cross-validation identified $\mathbf{C = 0.1}$ as the optimal choice for maximizing model performance on unseen data within the training set.
In the last part of this section, the best linear SVM is fitted on the training data set resulting from the previous manual split. We will show the confusion matrix and the accuracy value of this model coming from our 50/50 split.

[11]:
```
#prediction on test data
ypred_linear = best_linear_mod.predict(X_test)
#confusion matrix
confusion_table(ypred_linear, y_test)
```

[11]:
```
Truth      -1   1
Predicted
-1         17   1
 1          7  15
```

[12]:
```
(ypred_linear == y_test).mean() #accuracy
```

[12]:
```
0.8
```

The accuracy obtained on the manually separated test set is **0.8**, which is very close to the average accuracy of 0.8625 from cross-validation, confirming the model's good generalization performance.

4 Radial basis function and polynomial SVM

As previously mentioned, due to the nature of the dataset, a linear SVC is likely the most appropriate model for our classification problem.
To support this assumption, in this section we fit two additional Support Vector Machines, one using an **RBF kernel** and the other using a **polynomial kernel**, and compare their performance to that of the linear SVC.
Starting out with a radial kernel, we employed the `SVC` function with the parameter `kernel="rbf"`. Just like with the linear SVM, the two hyperparameters `C` and `gamma` were initially set to placeholder values. Then hyperparameter tuning by cross-validation was performed.

[40]:
```
svm_rbf = SVC(kernel="rbf", C=1, gamma=1)
```

[ ]:
```
C_range = [1e-4, 1e-3, 0.01, 0.1, 1, 10, 100, 1e3, 1e4]
gamma_range = [0.5, 1, 2, 3, 4]
params = {"C": C_range, "gamma":gamma_range}
kfold = skm.KFold(10, random_state=0, shuffle=True)
grid = skm.GridSearchCV(svm_rbf, param_grid=params, cv=kfold,␣
 ↪scoring="accuracy")
grid.fit(X_train, y_train)
print("Optimal rbf model: ", grid.best_estimator_)
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

```
Optimal rbf model:  SVC(C=0.0001, gamma=0.5)
Cross-validation accuracy of optimal model:  0.5416666666666667
```

[18]:
```
SVC(C=0.0001, gamma=0.5)
```

[19]:
```
Truth      -1   1
Predicted
-1          0   0
 1         24  16
```

[20]:
```
(ypred_rbf == y_test).mean() #accuracy
```

[20]:
```
0.4
```

Our evaluation of an SVM with an RBF kernel revealed significant performance issues. The model achieved a mere **0.4** accuracy on the test set, with a correspondingly low cross-validation accuracy of **0.54**. A key diagnostic insight was the model's consistent prediction of only class 1.
Given that prior analysis confirmed the dataset's balance, this biased output points decisively towards severe underfitting. This underfitting is attributed to the use of the smallest available

values for C (regularization parameter) and *gamma* (kernel coefficient) within our parameter grid. Low C values enforce high regularization, while low *gamma* values result in a very broad, less localized kernel influence. Together, these settings likely produced an excessively smooth and overly generalized decision boundary, preventing the model from capturing the nuanced patterns required for accurate class separation.
To address this, we transitioned to a **polynomial SVM**, implemented using `SVC(kernel="poly")`. The model's core hyperparameters, C and *degree*, were initially set to default/starting values. We then performed a comprehensive hyperparameter tuning process using cross-validation to identify the optimal combination for these parameters

[21]:
```
svm_poly = SVC(kernel="poly", C=1, degree=2)
```

[ ]:
```
C_range = [0.1, 1, 10, 100, 1000]
degree_range = [1, 2, 3, 4]
params = {"C": C_range, "degree":degree_range}
kfold = skm.KFold(10, random_state=0, shuffle=True)
grid = skm.GridSearchCV(svm_poly, param_grid=params, cv=kfold,␣
 ↪scoring="accuracy")
grid.fit(X_train, y_train)
grid.cv_results_
print("Optimal polynomial model: ", grid.best_estimator_)
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

```
Optimal polynomial model:  SVC(C=0.1, degree=4, kernel='poly')
Cross-validation accuracy of optimal model:  0.7
```

[23]:
```
SVC(C=0.1, degree=4, kernel='poly')
```

[24]:
```
Truth      -1   1
Predicted
-1         16   1
 1          8  15
```

[25]:
```
(ypred_poly == y_test).mean() #accuracy
```

[25]:
```
0.775
```

Given the high dimensionality of gene expression data, a linear decision boundary is generally expected to perform best, as complex kernels may introduce unnecessary variance. Nonetheless, the polynomial SVM achieved an accuracy of **0.775**, together with a cross-validation accuracy of **0.7**, which, although is lower than the 0.86 obtained with the linear kernel, is still relatively high. This suggests that the polynomial kernel was able to capture some underlying structure in the data, though it did not provide a significant advantage over the simpler linear model. These results reinforce the suitability of the linear kernel for this classification problem.

5 Gene selection

In gene expression analysis, it is common practice to **filter out** genes with low variability, as they often contribute little to the discrimination between classes and may introduce noise into the model.

To enhance signal-to-noise ratio and focus on the most informative features, we selected only the **top 5%** most variable genes based on their standard deviation across samples. This filtering step significantly reduces the dimensionality of the data while retaining the genes most likely to carry meaningful biological or predictive information.
We then repeated the analyses from the previous section using this reduced dataset to evaluate the impact of variability-based gene selection on model performance.

[ ]:
```
gene_std = X.std(axis=0) #calculate the s.d. of each gene (column-wise)
threshold = np.percentile(gene_std, 95) #compute the threshold corresponding to␣
 ↪the 95th percentile of gene variability
top_genes = gene_std[gene_std >= threshold].index #select the genes whose s.d.␣
 ↪is greater than or equal to the threshold
X_filtered = X[top_genes] #filter the original dataset to keep only the most␣
 ↪variable genes (top 5%)
X_filtered.shape
```

[ ]:
```
(79, 100)
```

We proceeded exactly as before: first, we performed a **manual split** of the dataset into training and test sets. Then, we initialized the SVM models with placeholder hyperparameter values before applying cross-validation to identify the optimal parameters for each kernel.

[27]:
```
#manual split of filtered training and test set using skm
X_train_f, X_test_f, y_train_f, y_test_f = skm.train_test_split(X_filtered, y,␣
 ↪test_size=0.5, random_state=0)
y_train.value_counts()
```

[27]:
```
y
 1    21
-1    18
Name: count, dtype: int64
```

5.1 Linear SVM

[28]:
```
svm_linear_filtered = SVC(C=10, kernel="linear")
svm_linear_filtered.fit(X_train_f,y_train_f)
C_values = [0.001, 0.01, 0.1, 1, 5, 10, 100]
kfold = skm.KFold(10, random_state=0, shuffle=True)
grid = skm.GridSearchCV(svm_linear_filtered,
                        {"C": C_values},
                        cv=kfold,
                        scoring="accuracy")
grid.fit(X_filtered,y)
#access the best parameter combination
print("Optimal linear model: ", grid.best_estimator_)
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

```
Optimal linear model:  SVC(C=0.01, kernel='linear')
Cross-validation accuracy of optimal model:  0.8339285714285714
```

After filtering the dataset, we observe a slight decrease in classification accuracy (from 0.86 to **0.83**). This suggests that while filtering helps reduce noise and dimensionality, it may also exclude some informative features.
However, the optimal regularization parameter C decreased from 0.1 to **0.01**, indicating that the reduced gene set led to a cleaner, more easily separable feature space.

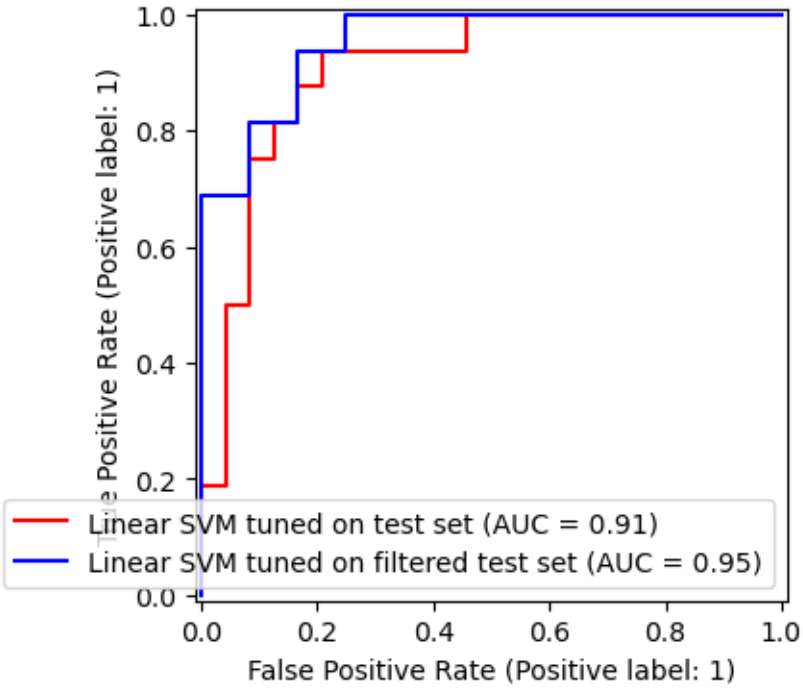
[ ]:
```
best_linear_mod_filtered = grid.best_estimator_
best_linear_mod_filtered.fit(X_train_f,y_train_f)
ypred_linear_filtered = best_linear_mod_filtered.predict(X_test_f)
ypred_linear_filtered
confusion_table(ypred_linear_filtered, y_test_f)
```

[ ]:
```
Truth       -1   1
Predicted
-1          17   0
 1           7  16
```

[30]:
```
(ypred_linear_filtered == y_test_f).mean() # accuracy
```

[30]:
```
0.825
```

While cross-validation (CV) favored the full feature set, a manual train-test split showed the filtered dataset achieved **higher accuracy** (0.8 to 0.825). This implies linear SVM's robustness in high dimensions benefits CV, but with fewer training samples in a single split, the reduced feature space likely prevented overfitting and improved generalization. This is reinforced by **ROC** analysis, where the filtered dataset achieved a superior AUC (0.95 vs. 0.91), indicating better discriminative power on the manually split data.



5.2 RBF and Polynomial SVM

Re-evaluating the RBF kernel to the filtered dataset confirmed its prior underperformance. Achieving a low accuracy (0.4) and CV accuracy (0.455833), it remained the least effective method.

[ ]:
```
svm_rbf_filtered = SVC(C=100, gamma=0.5, kernel="rbf")
svm_rbf_filtered.fit(X_train_f,y_train_f)
C_values = [0.001, 0.01, 0.1, 1, 5, 10, 100]
gamma_range = [0.5, 1, 2, 3, 4]
kfold = skm.KFold(5, random_state=0, shuffle=True)
grid = skm.GridSearchCV(svm_rbf_filtered,
                        {"C": C_values, "gamma": gamma_range},
                        cv=kfold,
                        scoring="accuracy")
grid.fit(X_filtered,y)
print("Optimal rbf model: ", grid.best_estimator_)
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

```
Optimal rbf model:  SVC(C=0.001, gamma=0.5)
Cross-validation accuracy of optimal model:  0.45583333333333337
```

[ ]:
```
Truth      -1   1
Predicted
-1          0   0
 1         24  16
```

[34]:
```
(ypred_rbf_filtered == y_test_f).mean() #accuracy
```

[34]:
```
0.4
```

In contrast, the Polynomial SVM showed a clear enhancement in this analysis. Its performance improved, achieving an accuracy of 0.8208, marking it as a much stronger method.

[36]:
```
svm_poly_filtered = SVC(C=100, degree=2, kernel="poly")
svm_poly_filtered.fit(X_train_f,y_train_f)
C_values = [0.001, 0.01, 0.1, 1, 5, 10, 100]
degree_range = [1, 2, 3, 4]
kfold = skm.KFold(5, random_state=0, shuffle=True)
grid = skm.GridSearchCV(svm_poly_filtered,
                        {"C": C_values, "degree": degree_range},
                        cv=kfold,
                        scoring="accuracy")
grid.fit(X_filtered,y)
print("Optimal polynomial model: ", grid.best_estimator_)
print("Cross-validation accuracy of optimal model: ", grid.best_score_)
```

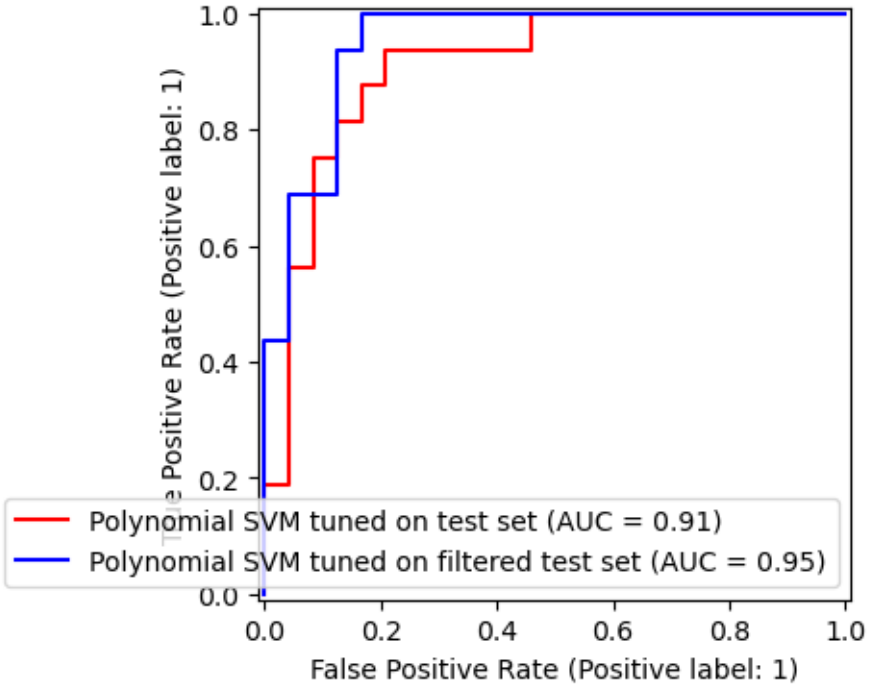
```
Optimal polynomial model:  SVC(C=0.1, degree=2, kernel='poly')
Cross-validation accuracy of optimal model:  0.8208333333333334
```

[ ]:
```
Truth      -1   1
Predicted
-1         17   0
 1          7  16
```

[38]:
```
(ypred_poly_filtered == y_test_f).mean() #accuracy
```

[38]:
```
0.825
```

Polynomial SVM performs nearly identically to Linear SVM, achieving the same test accuracy. Both models also show identical AUC values across original and filtered datasets, indicating similar sample ranking ability.



6 Conclusion

Theoretically, for gene expression data, characterized by high dimensionality and often a relatively small number of samples, **linear models** are frequently hypothesized to perform well. This is because they are less prone to overfitting in such scenarios compared to more complex models like RBF or high-degree Polynomial kernels, and can often find effective separating hyperplanes even when many features are present. Regularization (like the **C parameter** in SVMs) becomes critical to manage the high dimensionality and potential multicollinearity among genes. While non-linear kernels can capture more intricate relationships, they require careful tuning and are

more susceptible to fitting noise, especially without robust feature selection that isolates genuinely interacting features. This comparative analysis of SVM kernels on both the full and a 5% filtered gene expression dataset revealed distinct performance characteristics. The **Linear SVM** applied to the full dataset emerged as the optimal model, achieving the highest cross-validation accuracy of 0.8625 on the entirety of the dataset and 0.839 to the filtered one. In contrast, the **RBF kernel** consistently underperformed, yielding low accuracies of 0.5416 on the full dataset and dropping further to 0.4558 on the filtered set, indicating persistent issues likely related to underfitting. The **Polynomial SVM** presented a notable improvement with feature selection, increasing its accuracy from 0.7 on the full dataset to 0.8208 on the filtered data. Regarding the decrease in accuracy for the Linear SVM (and RBF) on the filtered dataset, this is somewhat counter-intuitive as feature selection is often beneficial. A possible explanation is that the 5% filtering process, while intended to reduce noise or dimensionality, may have inadvertently removed genes that, while perhaps not top-ranked individually by the filtering criterion, were collectively important for the linear model's discriminative power.

For the Polynomial SVM, the improvement suggests it was likely overfitting on the high-dimensional full dataset, and feature reduction helped by simplifying the problem space, allowing its non-linear capabilities to find a better fit on the reduced, potentially more focused, feature set. Overall, these results reinforce the idea that a **linear decision boundary** is sufficient for this classification task, and adding polynomial complexity does not substantially improve discrimination. The optimal regularization parameter C for this best-performing Linear SVM (on the full dataset) was found to be 0.1. When the Linear SVM was applied to the filtered dataset, the optimal C value shifted to 0.01. This decrease in C for the filtered data suggests that a stronger regularization (larger margin, more tolerance for misclassifications) was favored when working with a reduced, potentially less noisy, feature set. Conversely, the slightly higher C on the full, high-dimensional dataset allowed the model more flexibility to find a separating hyperplane among a larger number of features.