Overall Survival Prediction for Patients Diagnosed with Myeloid Leukemia

Gaëtan Ecrepont

March 2025

Abstract

Predicting overall survival for patients diagnosed with myeloid leukemia is crucial for personalized treatment. This report presents our machine learning approach to predict risk scores for adult patients with myeloid leukemia[1]. Our model is trained on a dataset containing clinical and genetic information, and we evaluate its performance using the IPCW-C-index. The results indicate that our model's predicted risk score is informative for flagging high-risk and low-risk patients, but it struggles with intermediate-risk predictions and recall for high-risk patients.

Contents

[1] We will mostly focus on Acute Myeloid Leukemia (AML) because it has the richest literature available.

This paper is a report of our work on the 2025 *Challenge Data* competition "Overall Survival Prediction for Patients Diagnosed with Myeloid Leukemia" proposed by the *QRT* and held in partnership with the *Institut Gustave Roussy*. The goal of this competition was to predict the risk score of adult patients diagnosed with myeloid leukemia using clinical and genetic data.

The code is available on GitHub at `https://github.com/gaetanX21/QRT2025`, and the official challenge page is available at `https://challengedata.ens.fr/participants/challenges/162`.

We first introduce the dataset and the models considered. We then describe our experimental setup and present our results. Finally, we discuss our model's limitations and ideas for improvement.

1 Dataset

The dataset consists of clinical and genetic information, with the target variable being overall survival. The main challenges we faced were:

- **Right-censored data** means we cannot use standard regression techniques. Instead, we must resort to *survival analysis* models.
- **Highly non-Gaussian data** (e.g., blood counts with large right skew) and complex categorical variables (e.g., cytogenetics) require careful preprocessing.
- **Lack of expert knowledge** on AML means we cannot easily understand interactions between features or engineer new features. We will need to leverage the existing medical literature to create informative features.
- **No clear data-generating process**, implying that we must consider the data as a *black box*. Besides, we don't really know how the data was collected and therefore we have no idea of the potential biases created by the data collection process.
- **Missing data** (7% in clinical, 9% in genetic data) requires imputation.

Note that the number of patients is moderate (around 3000) compared to the number of features (89 after feature engineering). This is a common situation in medical datasets, and it means that robustness will be a significant concern. We will need to be careful with hyperparameter tuning and model selection to avoid overfitting the data. In addition, 3000 samples is *a priori* not enough to train a robust deep learning model, so we won't consider this approach.

2 Models considered

We evaluated multiple models for survival prediction:[2]

- **Penalized Cox Model**: Simple and interpretable but lacks non-linearity.
- **Random Survival Forest**: Good interpretability but at risk of overfitting. Slow fit means cross-validation and hyperparameter tuning are expensive.
- **Survival XGBoost**: Gradient-Boosted Trees: high performance and fast training, but high risk of overfit unless carefully cross-validated and tuned.

	Penalized Cox Model	Random Survival Forest	Survival XGBoost
Nonlinear	no	yes	yes
High dimensional	mid	yes	yes
Little overfit	yes	no	no
Superior performance	no	yes	yes++
Fast fit	no	no	yes

Table 1: Comparison of survival models

The penalized Cox model and the Random Survival Forest were implemented using the `scikit-survival` whereas XGBoost was implemented using the `xgboost` library.

[2]Note that we won't describe the functioning of each model in detail, as it is not the focus of this report. For more information, please refer to the references at the end of this report.

After some testing, XGBoost was selected due to its superior performance and computational efficiency, though we had to carefully tweak the hyperparameters to avoid overfitting. Note that although ensembling models often leads to a "free lunch" increase in performance, we did not pursue this approach as the risk scores outputted by the different models were not directly comparable. Indeed, the risk scores from different models are relative, not absolute, and thus ensembling them is nontrivial.[3]

3 Feature Engineering

3.1 Clinical Features

We first extracted information for the `CYTOGENETICS` column:

- chromosomal anomalies: chromosome loss, gain, deletion, derivative, marker, short arm, long arm, translocation, XX, XY.
- missing cytogenetics.
- `n_+-`: total count of chromosome gains and losses.
- specific risk factors: loss of chromosome 7, deletion of chromosome 5q, translocation between chromosomes 9 and 22.
- `cyto_complexity`: measure of karyotype complexity, defined as the count of commas in the cytogenetics.
- `n_high_risk_markers`: count of high-risk markers (found in the literature).

We also defined the following ratios:

- `nlr`: neutrophil-to-lymphocyte ratio.
- `plr`: platelet-to-lymphocyte ratio.
- `blast_wbc_ratio`: ratio of blast cells to white blood cells.

Finally we added the `ipss_r_score` (Revised International Prognostic Scoring System) as a feature, it is a well-known prognostic factor in myeloid leukemia, defined as the following linear combination:

$$\texttt{ipss\_r\_score} = 0.4 \cdot \texttt{BM\_BLAST} - 0.3 \cdot \texttt{HB} - 0.2 \cdot \texttt{PLT} + 0.5 \cdot \texttt{cyto\_complexity} \tag{1}$$

3.2 Genetic Features

We first grouped the genetic features by patients and created aggregates and counts. We then created the following features (one per patient):

- **Gene statistics**: number of mutations, number of chromosome affected, number of genes affected, number of effects, and VAF (variant allele frequency) sum and max across the mutations.
- **Key genes**: flag for the presence of mutations in key genes (e.g., FLT3, NPM1, IDH1, IDH2, TP53, TET2, ASXL1, RUNX1, and DNMT3A).
- **Gene counts**: number of mutations in each gene.
- **Chromosome counts**: number of mutations in each chromosome.

4 Experimental Setup

4.1 Data Transformation

Although tree-based models do not care for data normalization *in theory*, we still applied a transformation in the hope that it would help XGBoost find its splits. Besides, having Gaussian features was needed to test the penalized Cox model which is a linear model. We only transformed clinical features because they exhibited significant right skewness. We applied the Yeo-Johnson transformation to normalize the data. The Box-Johnson transformation was also considered, but since some variables could equal zero, it was not applicable.

[3]We actually tried creating a metamodel using XGBoost and Random Survival Forest as base models, using a log transform and then a min-max transform to make the risk scores comparable. However, this approach did not yield significant performance improvements, so we dropped it in favor of a simpler model.

4.2 Data Imputation

Missing values were imputed using two methods:

- **Clinical Dataset**: 7% missing values. All real positive variables (blood counts and bone marrow blast percentage) were imputed using the mode. Missing cytogenetics were imputed with the string "missing" so that the model could know when a patient had no cytogenetics data available.
- **Genetic Dataset**: 9% missing values. Missing values were first ignored when grouping by patients and creating aggregates and counts. Then those newly created variables were imputed using -1. This makes sense as almost all these variables are (nonnegative integer) counts or indicators, i.e. ordinal variables. For example, the number of mutations in a gene is a count variable, and the presence of a mutation is an indicator variable. The categorical -1 value indicates to the model that the patient has no data available for this variable.

Empirically, other imputation methods (e.g., mean instead of mode, 0 instead of -1) did not yield significant differences in performance. We chose the above two approaches as they seemed most intuitive.

4.3 Evaluation Metric

Performance was assessed using IPCW-C-index, which is similar to the classic Concordance Index for Right Censored Data (C-index) but using inverse probability weighting to account for the right-censored nature of the data. A score of 0.5 indicates random predictions, while a score of 1.0 indicates perfect predictions (and a score of 0.0 indicates perfectly wrong predictions). Although complicated to compute, The IPCW-C-index is a robust metric for survival analysis.

4.4 Cross-Validation

We used 5-fold cross-validation to evaluate model performance. Note that to avoid data leakage, we performed the imputation the data transformation and imputation during the cross-validation process. We saw no use of Stratified K-Folds as the dataset was already balanced between dead and censored patients. Likewise, each patient had only one row so we did not need Group K-Folds.

4.5 Hyperparameter Tuning

XGBoost tuning is a delicate exercise. The large number of hyperparameters means that we can easily overfit the data, and yet one can obtain significant performance improvements by tuning them carefully. Since a full grid search would be too expensive and might lead to overfitting, we used a **Bayesian optimization approach** to find the best hyperparameters. We used the `optuna` library, which is a powerful and efficient hyperparameter optimization framework. The search space included:

- `n_estimators` $\in [100, 500]$: Number of trees to be built.
- `learning_rate` $\in [0.001, 0.1]$: Step size shrinkage used in update to prevent overfitting.
- `max_depth` $\in [2, 6]$: Maximum depth of a tree.
- `subsample` $\in [0.4, 0.8]$: Fraction of samples to be used for each tree.
- `colsample_bytree` $\in [0.4, 1.0]$: Fraction of features to be used for each tree.
- `colsample_bylevel` $\in [0.4, 1.0]$: Fraction of features to be used for each level.
- `colsample_bynode` $\in [0.4, 1.0]$: Fraction of features to be used for each node.
- `min_child_weight` $\in [1, 10]$: Minimum sum of instance weight (hessian) needed in a child.
- `gamma` $\in [0, 5]$: Minimum loss reduction required to make a further partition on a leaf node.

Note that except for `n_estimators`, all the other hyperparameters were added to combat overfit. Also, to have an idea of the degree of overfit for each hyperparameter configuration, we recorded the performance of the model on the training set while our search was optimizing performance on the test set.

- `n_estimators`: 369
- `max_depth`: 3

- `learning_rate`: 0.0197
- `subsample`: 0.432
- `colsample_bytree`: 0.661
- `colsample_bylevel`: 0.819
- `colsample_bynode`: 0.413
- `min_child_weight`: 9
- `gamma`: 1.392

Note in particular the high `n_estimators` to improve performance combined a small `max_depth` to limit overfitting. We used 369 estimators in the end, but in practice we ventured into the 500-1000 range during the cross-validation phase and obtained similar performance. We decided to keep the 369 estimators to decrease the risk of overfitting.

5 Results

5.1 C-IPCW scores

Our XGBoost model had the following C-IPCW scores:

- **Local cross-validation score**: 0.726 test & 0.763 train, indicating a moederate amount of overfitting.
- **Public leaderboard score**: 0.7268, placing us 23rd. The closeness between our local score and the public leaderboard score seems to confirm that we didn't overfit the training data. In fact, the public leaderboard score is even slightly better than our local score, which is probably due to luck.
- **Private leaderboard score**: 0.7076, placing us 13th. Note that the 0.0192 drop between the public and private leaderboard scores is the second lowest and much lower than the competition average drop of 0.0476. This is probably due to the fact that we only did a single submission to the public leaderboard, meaning that we couldn't overfit to it, unlike participants who presumably picked among all their submissions the one having the highest public score, thereby overfitting the public dataset *by definition*.

5.2 Feature Importance

For more robustness, we used four different feature importance methods:

- **Weight**: Total number of time a feature is used to split the data.
- **Total gain**: Total improvement in the loss function brought by a feature.
- **Total cover**: Total number of samples impacted by a feature across all splits.
- **Permutation importance**: Model-agnostic method that measures the increase in prediction error when a feature is randomly shuffled. This method is more robust to overfitting and provides a better estimate of feature importance, but it is also more computationally expensive.

For each feature importance method, we plotted the top 10 feature importance scores in Figure 1. The first three plots show the built-in feature importance scores for the XGBoost model using different criteria (weight, gain, and cover). The last plot shows the permutation importance scores, which provide a more robust estimate of feature importance. Note that in all plots, the most importance feature is `HB`. Then, in changing orders depending on the method, we have the base blood counts (`PLT`, `ANC`, `WBC`, `MONOCYTES`) and their interactions (`plr`, `nlr`, `ipss_r_score`) alongside `cyto_complexity`. The only genetic features in the top 10 are `vaf_sum` and `vaf_max`.

The other features have much lower importance scores, indicating that they are not as informative for the model, though they presumably do help the model gain a little bit of extra performance. In particular, it is remarkable that genetic features seem not be very informative for the risk score. We must be careful with this quick conclusion though, as it may be because genetic features are mostly highly specific indicator variables (e.g. you have gene X or not) and as such they aren't used a lot to discriminate patients *overall*, but they can still be very informative *for a given patient*.

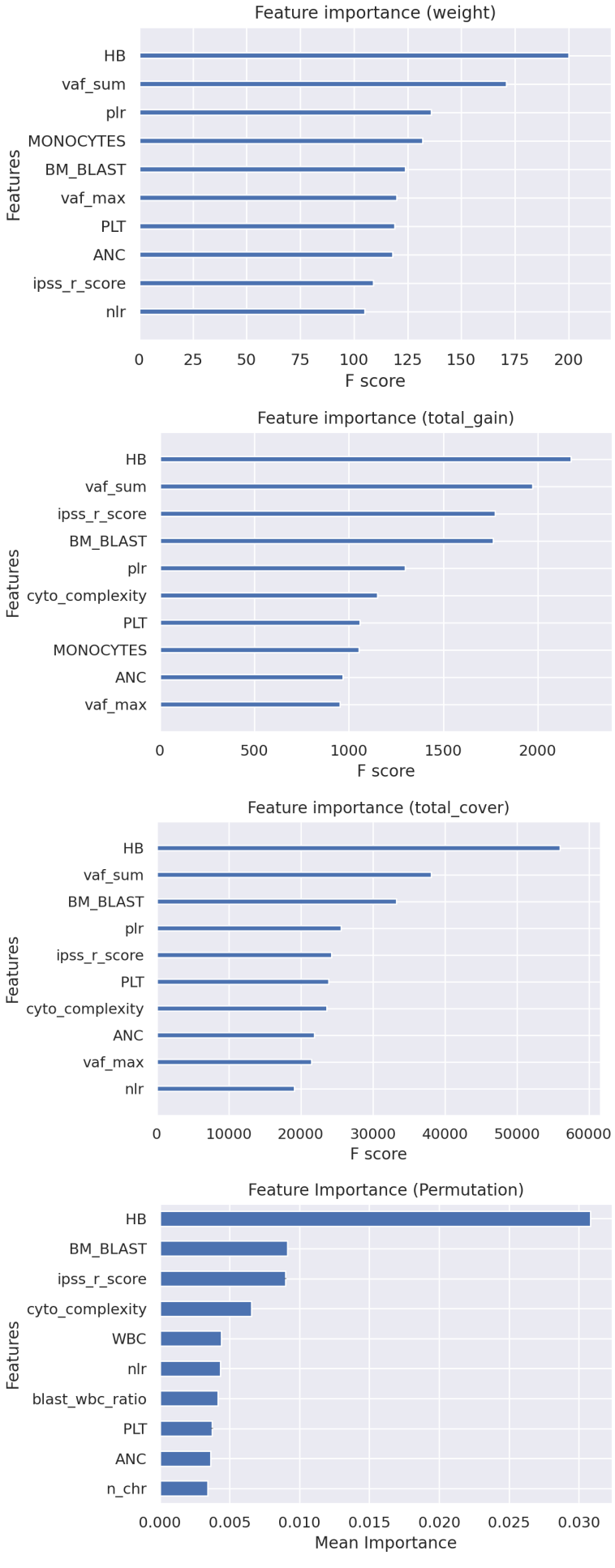

Figure 1: Feature importance scores for XGBoost model.

5.3 Analysis of model performance

The analysis of our model performance is not straightforward because the IPCW-C-index is complex. Since we are trying to predict the (relative) risk score of patients, one natural way to analyze the performance of our model is to look at how informative the risk score is. We can view this question using notions from classification, as if we were trying to predict whether the patient will die within the next 7 years[4] or not. We can thus define the following notions:

- **Precision**: When our model flags a patient as high-risk (resp. low-risk) does the patient indeed die within 7 years (resp. not die within 7 years)?
- **Recall**: When a patient dies (resp. doesn't die) within 7 years, does our model flag them as high-risk (resp. low-risk)?
- What happens for mid-risk patients?

To answer these questions, we can look at the scatter plot of the risk score against the actual survival time and survival class. The results are shown in Figure 2. We find that the model has *good precision for low-risk patients* (blue oval) and *high precision for high-risk patients* (yellow oval). However, the model has *medium recall for high-risk patients* (purple oval). Finally, the model is globally *uncertain for mid-risk patients* (green oval). This is not surprising as the model is trying to predict a continuous variable (the risk score) and the outcome is binary (death of not).

To summarize our analysis: our model is good at flagging low-risk patients and high-risk patients, but it is moderately good at catching high-risk patients. Besides, our model globally behaves poorly outside the extremes: that is, our risk score is not very informative for mid-risk patients.

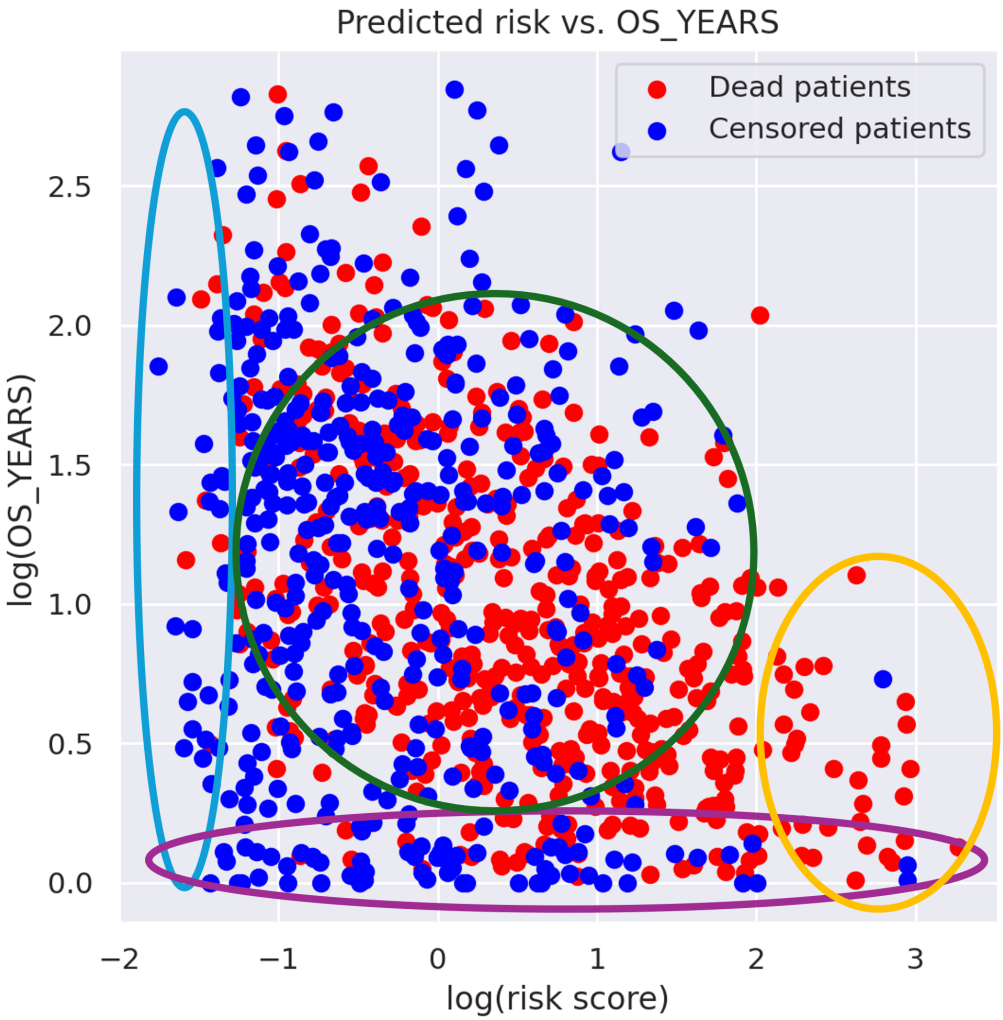

Figure 2: Scatter plot of risk score against actual survival time, colored by survival class.

[4]We censored observations at 7 years to compute C-IPCW.

6 Conclusion and Future Work

While XGBoost performs well at flagging high and low-risk patients, intermediate-risk predictions remain uncertain, and recall is medium on high-risk patients. This suggests that the model is not very informative for mid-risk patients, which is a significant limitation.

Several approaches can be explored to improve our model's performance:

- **Feature Engineering**: We could leverage medical literature on AML to create more informative features, especially for genetic data and cytogenetics. For example, we could create very specific features based on the presence of specific mutations or chromosomal abnormalities that are known to be associated with poor prognosis.
- **Ensembling**: Combining different models would almost certainly improve performance and robustness, but it is nontrivial since we first need to somehow make the risk scores comparable. We could also simply use a stacking metamodel to combine the predictions of different models without needing to make the risk scores comparable, but then the risk of overfit is very high, so we would probably need to use a very simple model as a metamodel, or gather more data...
- **Hyperparameter Tuning**: Further tuning of hyperparameters could yield better performance, especially for the XGBoost model. However we run the risk of overfitting the data, so we should be careful and use a validation set to evaluate the performance of the model.
- **Model Selection**: Exploring other models could lead to better performance, for instance deep learning models, which are known to perform well on high-dimensional data. However, they are also more prone to overfitting and require more data to train properly.
- **More Data**: Gathering more data could help improve the model's performance. This could be done by collaborating with other hospitals or institutions to gather more data on AML patients.

References

[1] Pourrajab F, Zare-Khormizi MR, Hashemi AS, Hekmatimoghaddam S. Genetic Characterization and Risk Stratification of Acute Myeloid Leukemia. Cancer Manag Res. 2020 Mar 25;12:2231-2253. doi: 10.2147/CMAR.S242479. PMID: 32273762; PMCID: PMC7104087. https://pubmed.ncbi.nlm.nih.gov/32273762/

[2] Revised international prognostic scoring system for myelodysplastic syndromes. Blood. 2012 Sep 20;120(12):2454-65. doi: 10.1182/blood-2012-03-420489. Epub 2012 Jun 27. PMID: 22740453; PMCID: PMC4425443. https://pubmed.ncbi.nlm.nih.gov/22740453/

[3] Scikit-survival: A library for survival analysis in Python. https://scikit-survival.readthedocs.io/en/stable/

[4] XGBoost: A scalable tree boosting system. https://xgboost.readthedocs.io/en/latest/

[5] Optuna: A Bayesian hyperparameter optimization framework. https://optuna.org/