Predicting Survival in Pediatric Bone Marrow Transplant Recipients

Ryan Jun Wang

Introduction

Why do we need bone marrow transplants? What are some complications that may occur?

Bone marrow transplants are an effective source of hematopoeitic stem cells, whose transplants constitutes important cell and gene therapies, treating a variety of disease, primarily hematological [1]. One of the main challenges with these therapeutics are the occurrence of Graft vs. Host disease (GVHD), in which the recipient immune system rejects the transplant due to the recipient immune complex recognizing the graft as a foreign entity [1, 2]. This complication can lead to worse survival rates in transplant recipients, specifically in the non-disease relapse case [2]. This is primarily prevented through donor matching via MHC/HLA (immune recognition complexes) which hopefully reduces the chances of a recipient detecting the donor as *foreign* [2]. In addition to GVHD, another major complication is relapse after transplant [3]. Relapse can result in high mortality post-transplant, especially in the case of highly-malignant disease [3]. Children provide a unique target for these transplants as they do not have fully developed immune system and so they both require less cells via the transplant and have lower rates of GVHD [1, 4]. As such, it is important to not only verify these claims, but effectively model the chance of success for a given transplant in order to make better medical decisions in the future.

An initial exploration of pediatric bone marrow transplants

A dataset of children who underwent bone marrow transplants will be used to build prediction models for survival outcome and time prediction [5]. There were 39 attributes accounting for donor age, ABO blood group, cytomegaloviral presence, recipient age, gender, body mass, ABO and RH groups, disease type, disease group, donor-recipient matches on gender, ABO, cytomegalaovirus, HLA, antigens, alleles, risk groups, stem cell sources, CD34 transmembrane expression, neutrophil recovery status/time, platelet recovery status/time, acute GVHD presence and time, extensive chronic GVHD development, disease relapse, survival time, and survival status. Initially, we performed a correlation analysis of numerical attributes to explore relationships between the attributes of the dataset (**Fig. 1**). Interestingly, we find that mortality is positively associated with disease relapse and longer platelet recovery time. This agrees with the previous literature review as previously noted, and the longer platelet recovery time may cause increased risk of death due to worsening bleeding (where platelets function is blood clotting). We also explored missingness, and found that we have 5-10% missingness in many attributes, so we attempt to perform a multiple imputation, combining the resulting predictive performance with a complete-case analysis. We also show that disease type seems to be associated with survival through an initial Kaplan-Meier estimate stratified by disease type (Acute Lymphocytic Leukemia - ALL, Acute Myeloid Leukemia - AML, chronic, non-malignant, and lymphoma diseases), where lymphoma seems be associated with lower survival time and non-malignant disease is associated with a higher survival time (**Fig. 2**).

How do we make more informed medical decisions?

We aim to use machine learning models in order to predict survival outcome and model survival time, in order to provide a way of making better and more informed medical decisions before actually applying a treatment. We will do this by comparing parametric and non-parametric models, that is, the parametric Regularized

Logistic Regression (RLR) versus the non-parametric Gradient-Boosted Classification Trees (GBCT) for modeling death outcome, and the semi-parametric Regularized Cox Proportional Hazards (RCPH) Regression versus the non-parametric Random Survival Forests for modeling survival time.

Data processing was performed in *R* (*tidyverse*) and *Python* (*pandas, numpy*), and all exploratory analysis, imputation, and modeling was performed in *Python* (*sklearn, xgboost, sksurv*).

Results

Feature selection

We first wanted to investigate what features should be used in order to achieve best predictive performance. Previously in the correlation analysis, we found some multicollinearity between recipient ages and HLA features, potentially because features are split into subsets. So in an attempt to get decorrelated features, we performed a principal components analysis and analyzed the variance explained by each principal component (**Fig. 3**). We find that the final ~5-10 principal components do not have as much information, however in general, we do not see a step drop in information provided, along with a relatively small dataset for prediction. Hence, we justify the use of the entire set of features as covariates for the classification and survival analysis task.

Multiple imputation results in better predictions than a complete-case approach

As previously noted, we performed two modifications on the pediatric dataset, multiple imputation and a complete case analysis (removing missing observations). To analyze the predictive performance, specifically for classification of patient death, we modeled two logistic regressions under a 5-fold cross validation to choose the optimal L2 regularizer penalty. We evaluated the model using F-1, AUC, and accuracy to effectively determine the discriminative capability of this initial model (**Table 1**, **Fig. 4**). The MICE imputed data resulted in better predictions in all three metrics, however with logistic regression we still have relatively poor performance for this task. This model boasts an especially poor 0.3846 F-1 score suggesting a very poor discriminative capability, in fact, the confusion matrix was only 5-7% better than a 50% true positive and true negative rate. Hence, we would definitely not employ this model for production.

Gradient-Boosted Classification Trees outperform Penalized Logistic Regression for classification

Due to the poor performance of the logistic regression, even with L2 penalty hyperparameter tuning, we attempt to use a most flexible non-parametric model, the gradient-boosted classification tree (GBCT). We performed a grid search cross validation using 5 folds to pick the best of a set of hyperparameters, resulting in an estimator that outperforms logistic regression by a wide margin in all three previously specified metrics (**Table 2**, **Fig. 5**). Similarly, this model approach improves on the true positive and true negative rates by around 20%. Together, this shows how powerful these more flexible modeling techniques can be, which is especially important in the context of medical diagnosis and prediction.

Relapse, chronic GVHD, and platelet recovery important for determining survival outcome

Using the previously specified GBCT, we performed a feature importance analysis in order to learn more about what was contributing to this predictive power. Here, we measured importance by *XGBoost*'s **Gain** metric, that is, the accuracy gained by splitting on a feature (**Fig. 6**). Interesting, like we initially saw in our exploratory correlation plot, we get the most accuracy gain from the *Relapse* feature, indicating whether or not an individual gets a hematological disease again. This makes sense, as a disease reoccuring might suggest something faulty with the immune system, or that the disease may have overcome some resistence and so it may be more lethal. This highlights the previously stated importance of relapse in transplant complications [3]. The second most important predictor was the development of Extensive Chronic Graft vs. Host Disease. This again, agrees with previous literature, and especially highlighting the **chronic** GVHD because unlike the acute version, chronic GVHD has been shown to be the most significant complication (apart from relaps) in child recipients [4]. Furthermore, the third most important predictor, *time until platelet recovery* also resulted in a large gain of accuracy. This makes sense as platelets are an important part of the immune system and so a longer period of time until full recovery would increase the change of death. Together, these data show the interpretability and performance of GBCTs, highlighting the support they provide for claims shown in literature and our correlation analysis.

Random Survival Forests outperform Penalized Cox Proportional Hazards for survival analysis

Finally, we investigated the actual survival times rather than purely the death outcome. We initially showed non-parametric Kaplan-Meier estimates showing a relatively unbalanced set of survival proportions, where lymphoma prescence was the only population that resulted in 0% survival (**Fig. 2**). In order to use these data for prediction, we first use a Penalized Cox Proportional Hazards model in order to take a semi-parametric appraoch, rather than going straight into a flexible model choice. We find that the survival curves are relatively parallel (except the lymphoma population), and so we take this approach. We perform a 5-fold cross validation in order to choose an Elastic Net linear scalar parameter to weight L1 and L2 penalization terms. Following this, we used a Random Survival Forest (RSF) in order to flexibly model time to death without hyperparmeter tuning. Even without tuning, the RSF outperforms the Cox model using *c*-index as a metric, essentially discriminating risk incorporating survival time over between pairs of individuals (**Table 3**, **Fig. 7**). We notice that RSF predictions tend to have lower survival on average, but spread the probabilities out, while the Cox model has a relatively imbalanced distribution. This suggests, even in terms of survival analysis, for the prediction task, non-parametric tree-based models like RSFs are able to effectively model the survival time over regression.

One limitation we see however, is that RSFs can be a bit more difficult to work with as they lack immediately interpretable features unlike coefficients from a regression model. However, purely for predictive performance, RSFs are favoured in these data. Furthermore, we did not perform hyperparameter tuning for RSFs (while we did on the Cox model), and doing so may further improve on these results. In addition, the previously noted imbalanced Cox distribution could be potentially due to the imbalanced dataset stratified on malignancy/disease, and so, having a more balanced dataset or performing weighting/subsampling might change these results.

Conclusion

Overall, we highlight the effectiveness of non-parametric modeling approaches in predicting both death outcome and predicting survival times. While more difficult than in regression, we provide support for empirical evidence on predictors of death post-bone marrow transplant in children including *chronic GVHD*, *platelet recovery time*, and *disease relapse* [2, 3, 4]. We show that non-parametric models outperform standard regression models even with penalization in terms of accuracy and discriminative accuracy. By being able to better predict the outcome of a patient given the potential bone marrow transplant approach, we will be able to make a more informed decision about whether the given approach is likely to work or fail. Furthermore, if a transplant was *absolutely necessary*, we could model the survival curves of the patients in order to predict in advance, where in time a patient might need more care (if they had a low predicted survival probability at the timepoint). In doing so, we may be able to make better medical procedure decisions resulting in an overall better outcome for patients on average.

References

1. Simpson, E. & Dazzi, F. Bone Marrow Transplantation 1957-2019. Frontiers in Immunology 10, (2019).
2. Sung, A. D. & Chao, N. J. Concise review: Acute graft-versus-host disease: Immunobiology, prevention, and treatment. Stem Cells Translational Medicine 2, 25–32 (2012).
3. Barrett, A. J. & Battiwalla, M. Relapse after Allogeneic Stem Cell transplantation. Expert Review of Hematology 3, 429–441 (2010).
4. Baird, K., Cooke, K. & Schultz, K. R. Chronic graft-versus-host disease (GVHD) in children. Pediatric Clinics of North America 57, 297–322 (2010).
5. Sikora M., Wrobel L., & Gudys A. GuideR: A guided separate-and-conquer rule learning in classification, regression, and survival settings. Knowledge-Based Systems (2019). Accessed from: https://archive.ics.uci.edu/ml/datasets/Bone+marrow+transplant%3A+children

Appendix

Figures

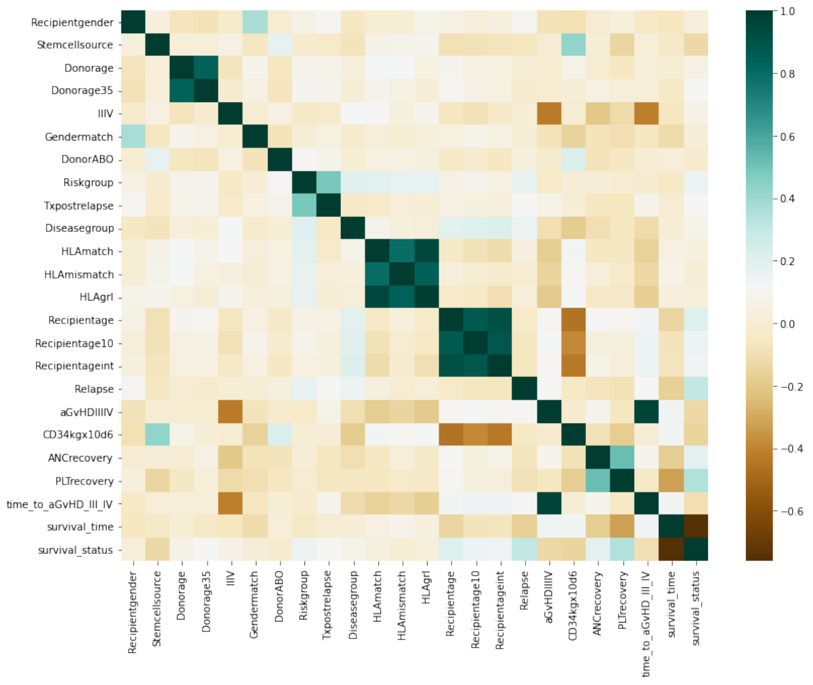

Figure 1: Correlation between covariates of the data.

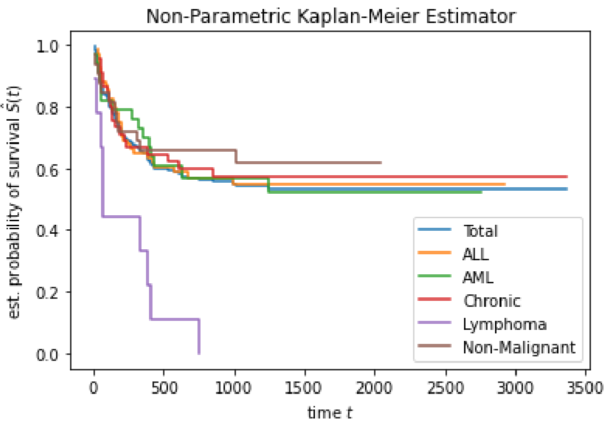

Figure 2: Kaplan Meier estimates of survival probability stratified by disease cateogory.

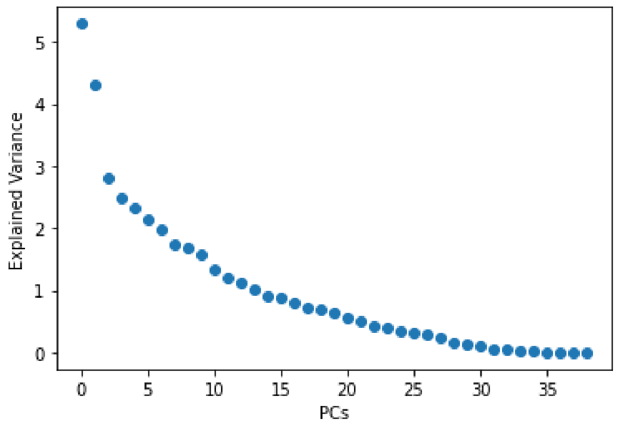

Figure 3: Variance explained by each principal component.

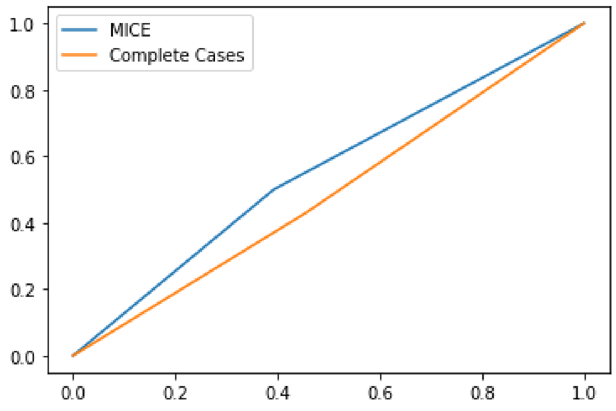

Figure 4: Predictive performance using multiple imputation versus complete case analysis.

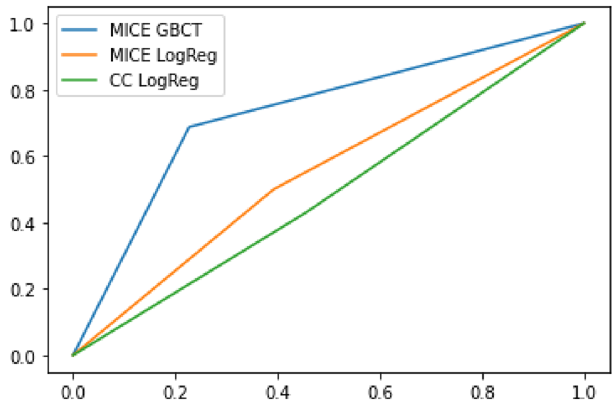

Figure 5: Comparison of prediction models.

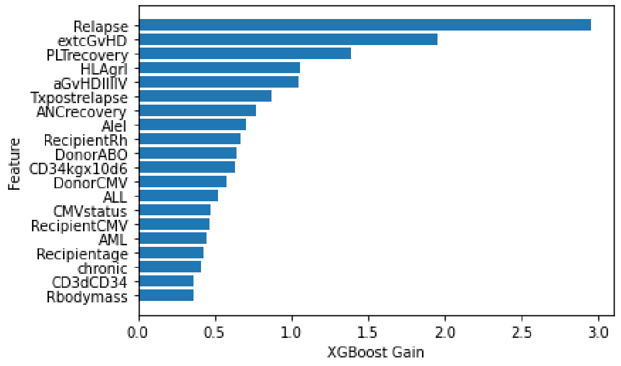

Figure 6: Features that contributed the most to accuracy gain in the GBCT.

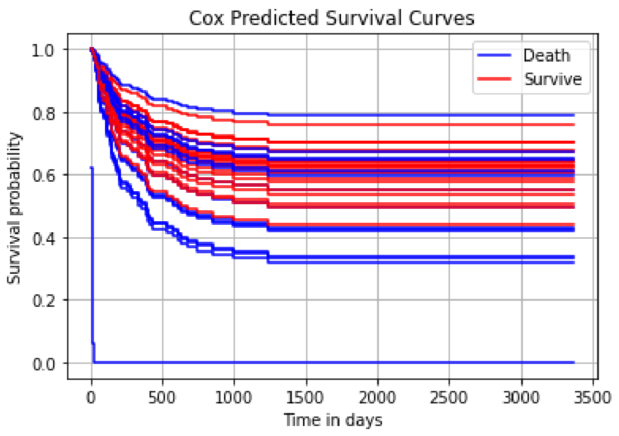

Figure 7: Survival curve comparisons of Penalized Cox Proportional Hazards model versus Random Survival Forests.

Tables

Table 1. Evaluation of missing data methods using logistic regression.

	F1	AUC	ACC
MICE	0.3846	0.5536	0.5789
Complete Cases	0.3000	0.4870	0.5172

Table 2. Evaluation of prediction models.

	F1	AUC	ACC
MICE LogReg	0.3846	0.5536	0.5789
Complete Cases LogReg	0.3000	0.4870	0.5172
GBCT	0.6875	0.7301	0.7368

Table 3. Comparison of survival models.

	c.index
Cox	0.6895
RSF	0.7152

Code

Data importing and tidying

```
library(tidyverse)
headers <- read_csv("bone-marrow.arff", skip=67, n_max = 37)
headers <- (headers %>% pull() %>% str_split(" ", simplify = T))[, 2]
#headers <- paste(shQuote(headers), collapse = ", ") # modify original file
dat <- read_csv("bone-marrow.arff", skip = 108)
colnames(dat) <- headers
write_csv(dat, 'bone-marrow.csv')
```

Analysis and modeling

```
import numpy as np
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_curve, roc_auc_score
from torch.utils.data import DataLoader, Dataset
np.random.seed(42)
```

```
dat = pd.read_csv('bone-marrow.csv')
dat
```

```
dat['survival_status'].sum()/dat.shape[0]
```

Investigating Correlation Structure

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize = (13, 10))
sns.heatmap(dat.corr(), cmap = 'BrBG')
```

Missing Data and Non-Numerics

Let's make sure the data is in numerical format.

```
dat.apply(pd.to_numeric, errors='coerce').isnull().sum()
```

We see some non-numeric data!

```
set(dat['RecipientCMV'])
```

```
set(dat['RecipientRh'])
```

```
set(dat['ABOmatch'])
```

```
set(dat['CMVstatus'])
```

```
set(dat['DonorCMV'])
```

```
set(dat['RecipientCMV'])
set(dat['Disease'])
```

```
set(dat['Antigen'])
```

```
set(dat['Alel'])
```

```
set(dat['extcGvHD'])
```

```
# first we should get numerical structure
dat = pd.concat([
    dat,
    pd.get_dummies(dat['Disease'])
    ],
    axis = 1).drop(['Disease'], axis = 1)
dat = dat.apply(pd.to_numeric, errors='coerce')
dat
```

Imputation

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer, KNNImputer

X = dat.drop(['survival_time', 'survival_status'], axis = 1)


Death = dat['survival_status']
STime = dat['survival_time']

mulpute = IterativeImputer(max_iter=10, random_state=0, verbose = 2).fit(X)
X_mp = mulpute.transform(X)

missing_idx = [x[1] for x in list(dat.unstack()[dat.isnull().unstack()].index)]
X_cc = dat.drop(index=missing_idx)
Death_cc = X_cc['survival_status']
STime_cc = X_cc['survival_time']
X_cc = X_cc.drop(['survival_time', 'survival_status'], axis = 1)
```

First, we'll split an 80/20 train-test in order to work with both the MICE and complete case training set.

```
trainX_mp, testX_mp, \
    trainDeath_mp, testDeath_mp, \
    trainSTime_mp, testSTime_mp = train_test_split(
             X_mp,
             Death,
             STime,
             test_size = 0.2
)

trainX_cc, testX_cc, \
    trainDeath_cc, testDeath_cc, \
    trainSTime_cc, testSTime_cc = train_test_split(
             X_cc,
             Death_cc,
             STime_cc,
             test_size = 0.2
)
```

```
p = trainX_mp.shape[1]

print(f"MICE Training shape: {trainX_mp.shape, trainDeath_mp.shape}")
print(f"MICE Testing shape: {testX_mp.shape, testDeath_mp.shape}")
```

PCA

```

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1. Standardize rows to training set
stdizer = StandardScaler().fit(trainX_mp)
trainX_std = stdizer.transform(trainX_mp) # trainX_std.sum(axis = 0) = 0 down columns
testX_std = stdizer.transform(testX_mp) # standardize using trainX

# 1. PCA
pca = PCA(n_components = p).fit(trainX_std)
trainX_pca = pca.transform(trainX_std)
plt.scatter(range(p), pca.explained_variance_)
plt.xlabel("PCs")
plt.ylabel("Explained Variance")
```

For MICE

```

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1. Standardize rows to training set
stdizer_cc = StandardScaler().fit(trainX_cc)
trainX_stdcc = stdizer.transform(trainX_cc) # trainX_std.sum(axis = 0) = 0 down columns
testX_stdcc = stdizer.transform(testX_cc) # standardize using trainX

# 1. PCA
pca_cc = PCA(n_components = p).fit(trainX_stdcc)
trainX_pcacc = pca.transform(trainX_stdcc)
plt.scatter(range(p), pca_cc.explained_variance_)
plt.xlabel("PCs")
plt.ylabel("Explained Variance")
```

For Complete Cases

Predicting Survival Status

An Initial Logistic Model

```
from sklearn.linear_model import LogisticRegressionCV
logreg_mp = LogisticRegressionCV(cv= 5, random_state = 0)
logreg_mp.fit(trainX_mp, trainDeath_mp)
y_pred_mp = logreg_mp.predict(testX_mp)
```

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, roc_curve, confusion_matrix
print(f"MICE F1 Score: {f1_score(y_pred_mp, testDeath_mp)}")
print(f"MICE AUC Score: {roc_auc_score(y_pred_mp, testDeath_mp)}")
print(f"MICE ACC Score: {accuracy_score(y_pred_mp, testDeath_mp)}")
print(f"MICE Confusion Matrix:\n {confusion_matrix(y_pred_mp, testDeath_mp, normalize = 'true')}")
tpr_mp, fpr_mp, threshold_mp = roc_curve(y_pred_mp, testDeath_mp)
plt.plot(tpr_mp, fpr_mp)
```

```
logreg_cc = LogisticRegressionCV(cv= 5, random_state = 0)
logreg_cc.fit(trainX_cc, trainDeath_cc)
y_pred_cc = logreg_mp.predict(testX_cc)
```

```
print(f"CC F1 Score: {f1_score(y_pred_cc, testDeath_cc)}")
print(f"CC AUC Score: {roc_auc_score(y_pred_cc, testDeath_cc)}")
print(f"CC ACC Score: {accuracy_score(y_pred_cc, testDeath_cc)}")
print(f"CC Confusion Matrix:\n {confusion_matrix(y_pred_cc, testDeath_cc, normalize = 'true')}")

plt.plot(tpr_mp, fpr_mp, label = 'MICE')

tpr_cc, fpr_cc, threshold_cc = roc_curve(y_pred_cc, testDeath_cc)
plt.plot(tpr_cc, fpr_cc, label = 'Complete Cases')
plt.legend()
```

Can we improve this with XGBoost?

```
# hyperparameter tuning
params = {'n_estimators':[150, 250, 350],
         'max_depth':[3, 6, 10],  # XGB complexity
         'min_child_weight':[1, 6], # min sum of obs req in a child
         'learning_rate':[0.2, 0.3], # step size shrinkage
         'gamma':[1], # complexity penalty term
         'subsample':[0.5, 1], # subsample for each tree
         'objective':['binary:logistic']}

xgb_gs = xgb.XGBClassifier(seed = 0, tree_method='gpu_hist')
gs = GridSearchCV(xgb_gs, params,
                 cv = 5, verbose = 3)
gs.fit(trainX_mp, trainDeath_mp)
```

```
gs.best_params_
```

```
best_params = {'gamma': 1,
 'learning_rate': 0.2,
 'max_depth': 3,
 'min_child_weight': 1,
 'n_estimators': 250,
 'objective': 'binary:logistic',
 'subsample': 1}

xgb_c = xgb.XGBClassifier(**best_params, seed = 0, tree_method='gpu_hist')
xgb_c.fit(trainX_mp, trainDeath_mp)
y_pred = xgb_c.predict(testX_mp)
```

```python
print(f"CC F1 Score: {f1_score(y_pred, testDeath_mp)}")
print(f"CC AUC Score: {roc_auc_score(y_pred, testDeath_mp)}")
print(f"CC ACC Score: {accuracy_score(y_pred, testDeath_mp)}")
print(f"CC Confusion Matrix:\n {confusion_matrix(y_pred, testDeath_mp, normalize = 'true')}")

tpr, fpr, threshold = roc_curve(y_pred, testDeath_mp)
plt.plot(tpr, fpr, label = 'MICE GBCT')
plt.plot(tpr_mp, fpr_mp, label = 'MICE LogReg')
plt.plot(tpr_cc, fpr_cc, label = 'CC LogReg')
plt.legend()
```

```python
xgb_c.get_booster().feature_names = list(X.columns)
importance = xgb_c.get_booster().get_score(importance_type= 'gain')
for key in importance:
    importance[key] = [importance[key]]
imp_df = pd.DataFrame(importance).T.reset_index()
imp_df = imp_df.sort_values(0, ascending = False).head(n = 20)
plt.barh(imp_df['index'][::-1], imp_df[0][::-1])
plt.xlabel('XGBoost Gain')
plt.ylabel('Feature')
```

Modeling Survival Time

```python
!pip install scikit-survival
```

```python
dat.columns
```

```python
from sksurv.nonparametric import kaplan_meier_estimator

# disease idx
ALL_idx = dat.index[dat['ALL'] == 1].tolist()
AML_idx = dat.index[dat['AML'] == 1].tolist()
chronic_idx = dat.index[dat['chronic'] == 1].tolist()
lymphoma_idx = dat.index[dat['lymphoma'] == 1].tolist()
nonmalignant_idx = dat.index[dat['nonmalignant'] == 1].tolist()

time, survival_prob = kaplan_meier_estimator(Death.astype(bool), STime)
time_all, survival_prob_all = kaplan_meier_estimator(Death.astype(bool)[ALL_idx], STime[ALL_idx])
time_aml, survival_prob_aml = kaplan_meier_estimator(Death.astype(bool)[AML_idx], STime[AML_idx])
time_chr, survival_prob_chr = kaplan_meier_estimator(Death.astype(bool)[chronic_idx], STime[chronic_idx
time_lym, survival_prob_lym = kaplan_meier_estimator(Death.astype(bool)[lymphoma_idx], STime[lymphoma_i
time_nm, survival_prob_nm = kaplan_meier_estimator(Death.astype(bool)[nonmalignant_idx], STime[nonmalig
plt.step(time, survival_prob, where="post", label = 'Total')
plt.step(time_all, survival_prob_all, where="post", label = 'ALL')
plt.step(time_aml, survival_prob_aml, where="post", label = 'AML')
plt.step(time_chr, survival_prob_chr, where="post", label = 'Chronic')
plt.step(time_lym, survival_prob_lym, where="post", label = 'Lymphoma')
plt.step(time_nm, survival_prob_nm, where="post", label = 'Non-Malignant')
plt.ylabel("est. probability of survival $\hat{S}(t)$")
plt.xlabel("time $t$")
plt.title("Non-Parametric Kaplan-Meier Estimator")
plt.legend()
```

Survival Analysis

```
# https://scikit-survival.readthedocs.io/en/stable/user_guide/coxnet.html
import warnings
from sksurv.linear_model import CoxnetSurvivalAnalysis
from sklearn.exceptions import FitFailedWarning
from sklearn.pipeline import make_pipeline

# make [(outcome, time), ...] dataset
trainy_surv = np.array(list(zip(trainDeath_mp.astype(bool), trainSTime_mp)),
                    dtype=[('Status', '?'), ('Survival_in_days', '<f8')])

coxnet_pipe = make_pipeline(
    StandardScaler(),
    CoxnetSurvivalAnalysis(l1_ratio=0.9, # elasticnet mixing (l1_ratio*l1 + (1-l1_ratio)*l2)
                           alpha_min_ratio=0.01, # min weighting of penalty
                           max_iter=100, # optimization
                           fit_baseline_model = True) # estimating baseline survival/cumhazard
)
warnings.simplefilter("ignore", UserWarning)
warnings.simplefilter("ignore", FitFailedWarning)
coxnet_pipe.fit(trainX_mp, trainy_surv) # searches for 100 alphas up to 1% of max

# choose best alpha from estimated
estimated_alphas = coxnet_pipe.named_steps["coxnetsurvivalanalysis"].alphas_
gcv = GridSearchCV(
    make_pipeline(StandardScaler(), CoxnetSurvivalAnalysis(l1_ratio=0.9,
                                                           fit_baseline_model=True)),
    param_grid={"coxnetsurvivalanalysis__alphas": [[v] for v in estimated_alphas]},
    cv=5,
    error_score=0.5,
    n_jobs=1).fit(trainX_mp, trainy_surv)

cv_results = pd.DataFrame(gcv.cv_results_)
```

```
gcv.best_params_
```

```
best_params = {'coxnetsurvivalanalysis__alphas': [0.06523406814746949]}
cox = gcv.best_estimator_
#dat_orig = pd.read_csv('bone-marrow.csv')
#X_surv = trainX_mp.drop(['survival_status', 'survival_time'], axis = 1)

cox.fit(trainX_mp, trainy_surv)
```

```
from sksurv.ensemble import RandomSurvivalForest
rsf = RandomSurvivalForest(n_estimators=1000,
                           min_samples_split=10,
                           min_samples_leaf=15,
                           n_jobs=-1,
                           random_state=0)
rsf.fit(trainX_mp, trainy_surv)
```

```
from sksurv.metrics import (
    concordance_index_censored,
    concordance_index_ipcw,
    cumulative_dynamic_auc,
    integrated_brier_score,
)
from sksurv.functions import StepFunction

testy_surv = np.array(list(zip(testDeath_mp.astype(bool), testSTime_mp)),
                    dtype=[('Status', '?'), ('Survival_in_days', '<f8')])


score_cindex = pd.Series(
    [
        cox.score(testX_mp, testy_surv),
        rsf.score(testX_mp, testy_surv)
    ],
    index=["Cox", "RSF"], name="c-index",
)
score_cindex
```

```
testy_surv[0][0]
```

```
# predicted cox curves
surv_cox = cox.predict_survival_function(testX_mp, return_array=True)

a = 0
b = 0
for i, s in enumerate(surv_cox):
    if testy_surv[i][0]:
        c = 'blue'
        label = 'Death'
    else:
        c = 'red'
        label = 'Survive'
    plt.step(cox[1].event_times_, s, where="post", label=label, c = c)
    if label == 'Death':
        if b == 0:
            plt.legend()
            b += 1
    if label == 'Survive':
        if a == 0:
            plt.legend()
            a += 1
plt.ylabel("Survival probability")
plt.xlabel("Time in days")
plt.title("Cox Predicted Survival Curves")
plt.grid(True)
plt.show()

# predicted rsf curves
surv_rsf = rsf.predict_survival_function(testX_mp, return_array=True)
a, b = 0, 0
for i, s in enumerate(surv_rsf):
    if testy_surv[i][0]:
        c = 'blue'
        label = 'Death'
    else:
        c = 'red'
        label = 'Survive'
    plt.step(rsf.event_times_, s, where="post", label=label, c = c)
    if label == 'Death':
        if b == 0:
            plt.legend()
            b += 1
    if label == 'Survive':
        if a == 0:
            plt.legend()
            a += 1
plt.ylabel("Survival probability")
plt.xlabel("Time in days")
plt.title("RSF Predicted Survival Curves")
plt.grid(True)
plt.show()
```