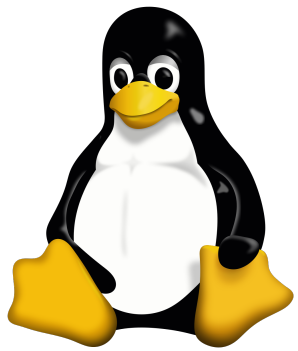
Algoritmos de Ensemble para Identificação de Biomarcadores da COVID-19

Pedro Henrique Mendes

Rafael Júnior Ribeiro

Orientador: Marco A. P. Idiart

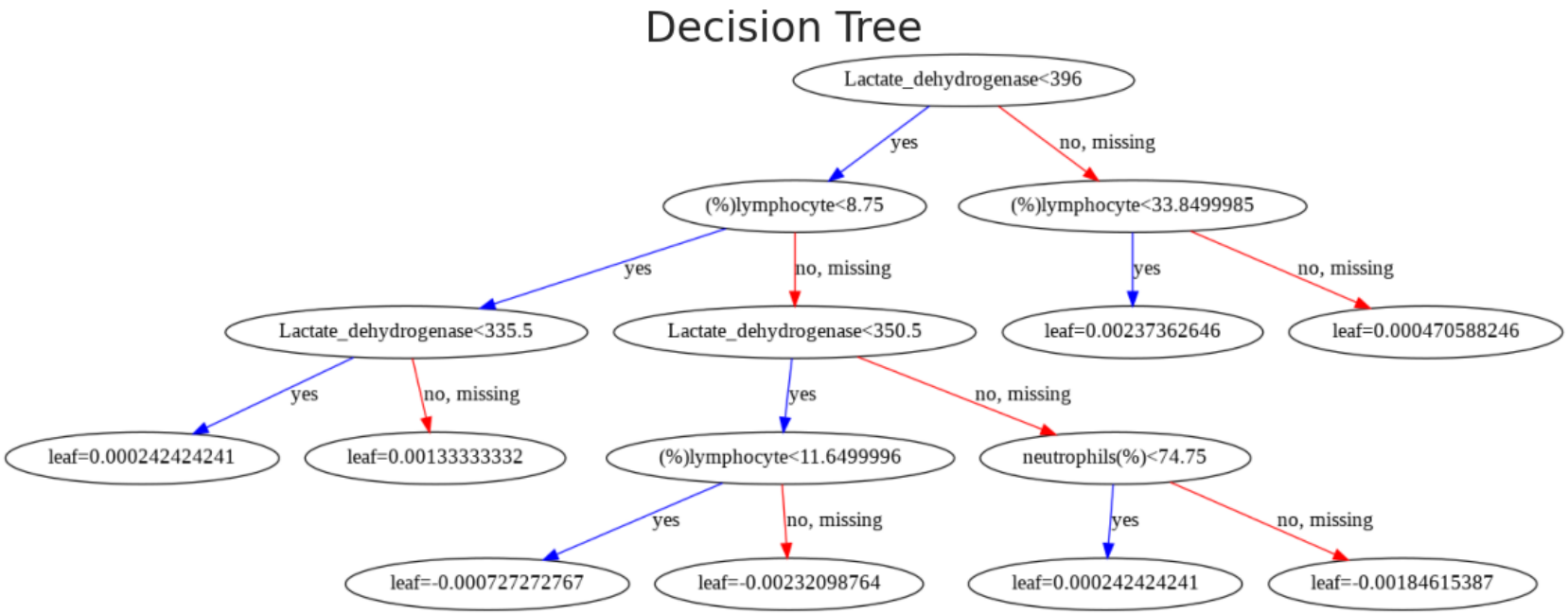


Instituto de Física
Universidade Federal do Rio Grande do Sul

SIC 2021

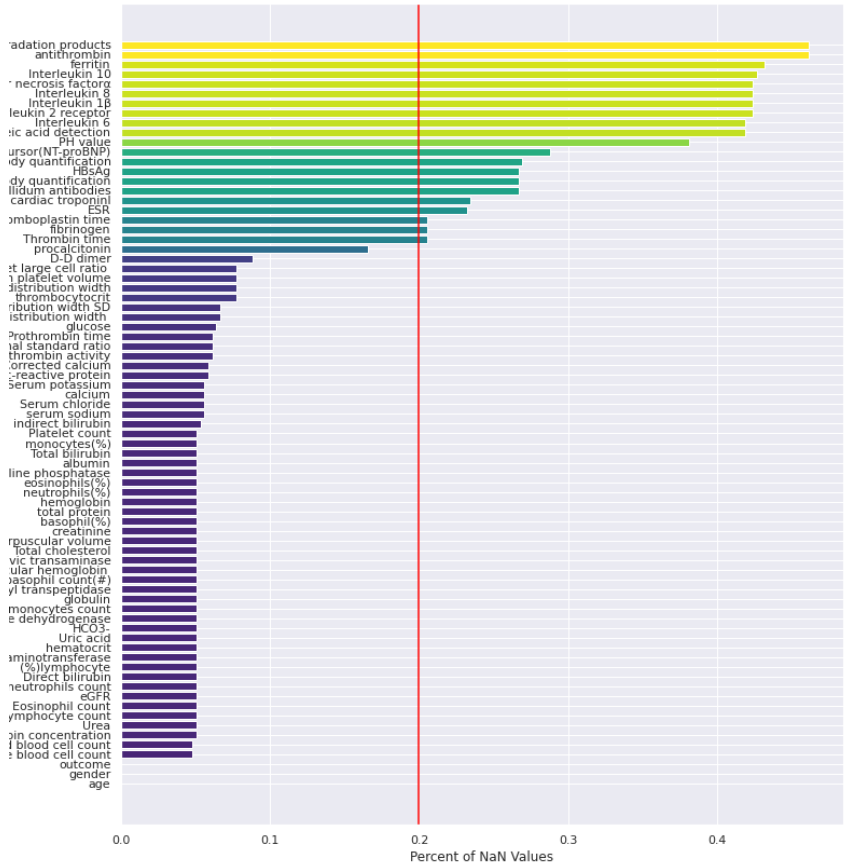
Algoritmos de Ensemble

- Múltiplos modelos menores se juntam para criar um modelo único;
- Criam resultados com menor variância e que obtém melhor performance;
- Existem diversos algoritmos dessa classe, vamos utilizar os que tem como base árvores de decisão. Os escolhidos foram *Random Forest* e *Extreme Gradient Boosting*.



Dados

Dados Médicos coletados entre 10 de Janeiro e 18 de Fevereiro de 2020, do Hospital de Tongji, da China. Não foram levados em conta as variáveis que tinham mais de 20% de informações faltantes.



Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,             #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,              #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,             #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,             #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

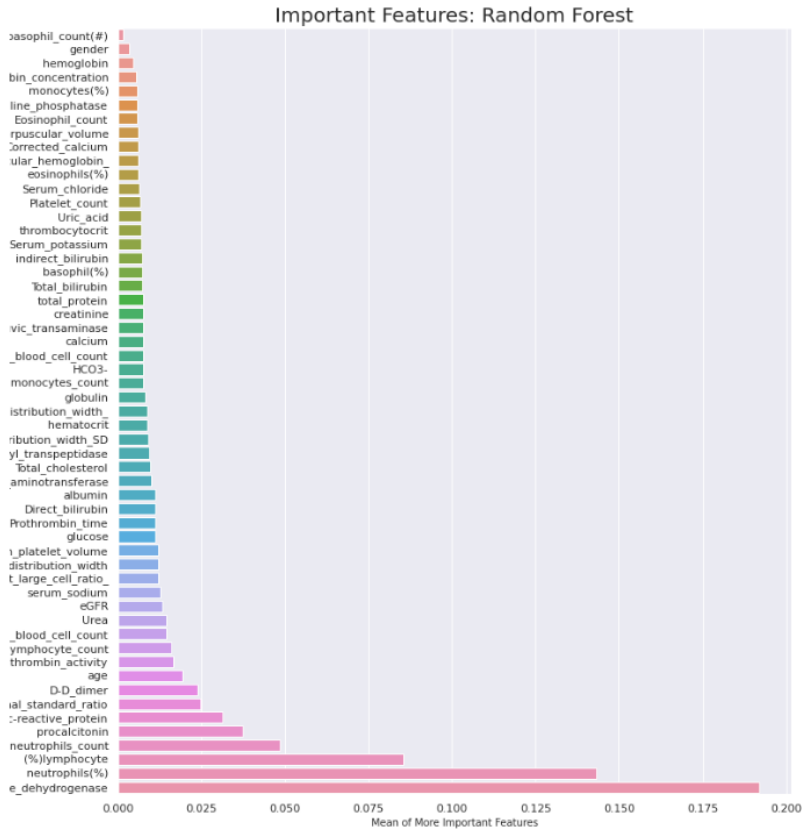
```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,              #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

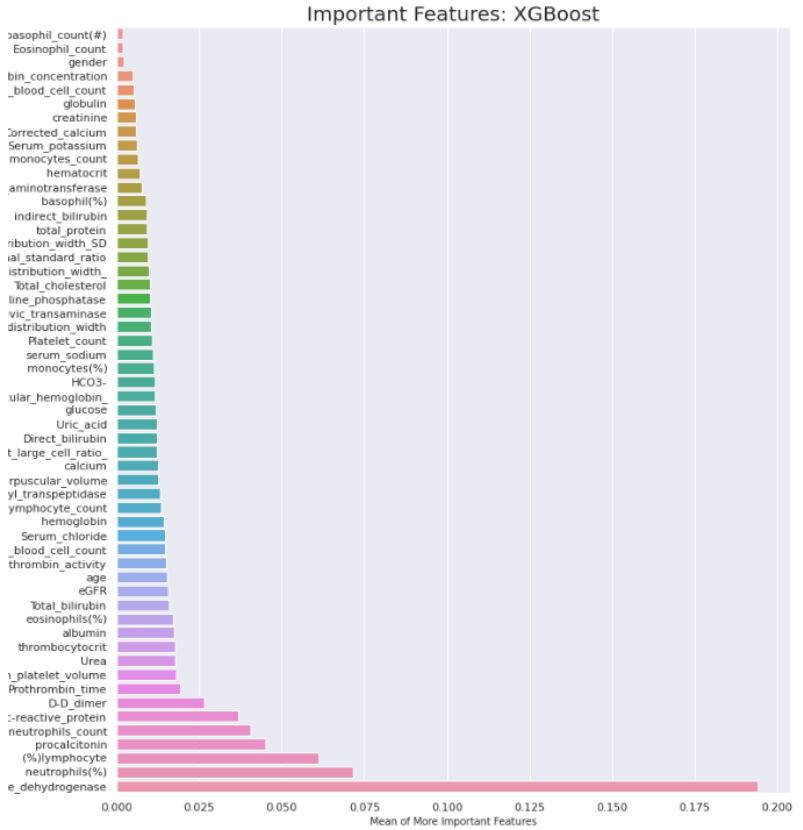
Treinamento e Avaliação dos Pesos

Usando os classificadores da biblioteca XGBoost.

```
[12] import_feature_RF = pd.DataFrame(columns=X.columns)

     tmax = 50 #100, 150, 200

     for i in range(tmax):
         X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             test_size=0.3,
                                                             random_state=i) #state for iteration

         model_RF = XGBRFClassifier(max_depth=4,              #Maximum tree depth for base learners.
                                    learning_rate=0.2,       #learning rate ("eta")
                                    reg_lambda=1,            #L2 regularization term on weights
                                    n_estimators=150,        #Number of boosting rounds.
                                    subsample = 0.9,         #Subsample ratio of the training instance.
                                    colsample_bytree = 0.9)  #Subsample ratio of columns when constructing each tree.

         model_RF.fit(X_train, y_train)
         import_feature_RF = import_feature_RF.append(pd.DataFrame(model_RF.feature_importances_,
                                                                   index=X.columns).transpose())
```

Biomarcadores





Os biomarcadores mais importantes são: Desidrogenase Lática, porcentagem de Neutrófilos, porcentagem de Linfócitos, contagem de Neutrófilos e Procalcitonina.

Avaliação do Modelo

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_best_RF, #x = x_best
                                                         y, #same
                                                         test_size=0.3, #same
                                                         random_state=3463) #def state

     model_RF = XGBRFClassifier(max_depth=4,
                                learning_rate=0.2,
                                reg_lambda=1,
                                n_estimators=150,
                                subsample=0.9,
                                colsample_bytree=0.9,
                                verbosity=0)

     model_RF.fit(X_train,y_train)

     predict_labels = model_RF.predict(X_test)
     c_matrix_RF = confusion_matrix(y_test, predict_labels)
```

Similar ao treinamento, porém não são feitas iterações, o modelo é ajustado diretamente.

Avaliação do Modelo

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X_best_RF, #x = x_best
                                                        y, #same
                                                        test_size=0.3, #same
                                                        random_state=3463) #def state

    model_RF = XGBRFClassifier(max_depth=4,
                               learning_rate=0.2,
                               reg_lambda=1,
                               n_estimators=150,
                               subsample=0.9,
                               colsample_bytree=0.9,
                               verbosity=0)

    model_RF.fit(X_train,y_train)

    predict_labels = model_RF.predict(X_test)
    c_matrix_RF = confusion_matrix(y_test, predict_labels)
```

Similar ao treinamento, porém não são feitas iterações, o modelo é ajustado diretamente.

Avaliação do Modelo

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_best_RF, #x = x_best
                                                         y, #same
                                                         test_size=0.3, #same
                                                         random_state=3463) #def state

     model_RF = XGBRFClassifier(max_depth=4,
                                learning_rate=0.2,
                                reg_lambda=1,
                                n_estimators=150,
                                subsample=0.9,
                                colsample_bytree=0.9,
                                verbosity=0)

     model_RF.fit(X_train,y_train)

     predict_labels = model_RF.predict(X_test)
     c_matrix_RF = confusion_matrix(y_test, predict_labels)
```

Similar ao treinamento, porém não são feitas iterações, o modelo é ajustado diretamente.

Avaliação do Modelo

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_best_RF, #x = x_best
                                                         y, #same
                                                         test_size=0.3, #same
                                                         random_state=3463) #def state

     model_RF = XGBRFClassifier(max_depth=4,
                                learning_rate=0.2,
                                reg_lambda=1,
                                n_estimators=150,
                                subsample=0.9,
                                colsample_bytree=0.9,
                                verbosity=0)

     model_RF.fit(X_train,y_train)

     predict_labels = model_RF.predict(X_test)
     c_matrix_RF = confusion_matrix(y_test, predict_labels)
```

Similar ao treinamento, porém não são feitas iterações, o modelo é ajustado diretamente.

Avaliação do Modelo

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_best_RF, #x = x_best
                                                         y, #same
                                                         test_size=0.3, #same
                                                         random_state=3463) #def state

     model_RF = XGBRFClassifier(max_depth=4,
                                learning_rate=0.2,
                                reg_lambda=1,
                                n_estimators=150,
                                subsample=0.9,
                                colsample_bytree=0.9,
                                verbosity=0)

     model_RF.fit(X_train,y_train)

     predict_labels = model_RF.predict(X_test)
     c_matrix_RF = confusion_matrix(y_test, predict_labels)
```

Similar ao treinamento, porém não são feitas iterações, o modelo é ajustado diretamente.

Resultados

Os valores de *Precision* e *Recall* são dados por

$$P = \frac{TP_i}{TP_i + FP_i} \tag{1}$$

$$R = \frac{TP_i}{TP_i + FN_i} \tag{2}$$

Random Forest		
	Precision	Recall
Sobrev.	0.90	0.75
Óbitos	0.76	0.90

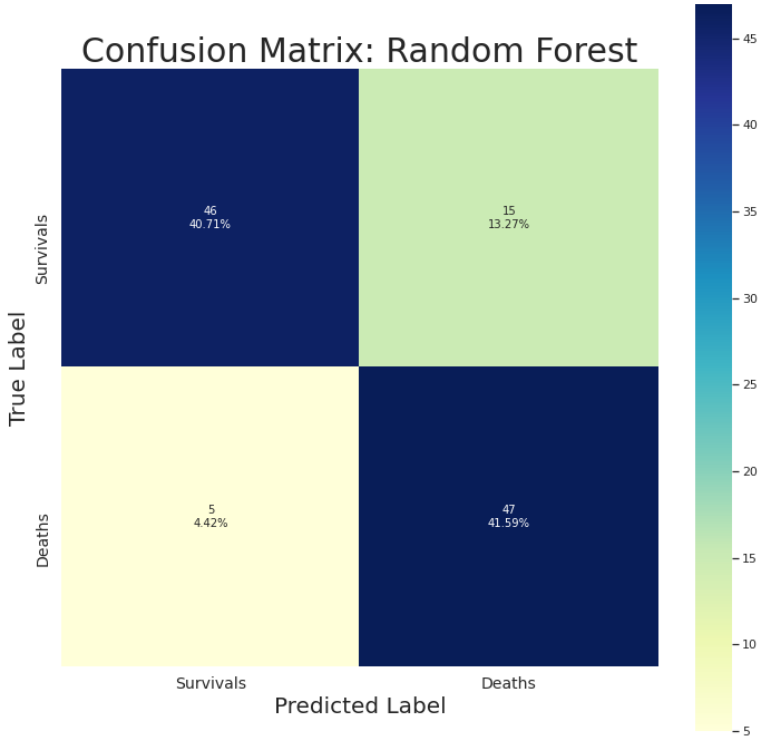
Extreme Gradient Boosting		
	Precision	Recall
Sobrev.	0.86	0.79
Óbitos	0.77	0.85

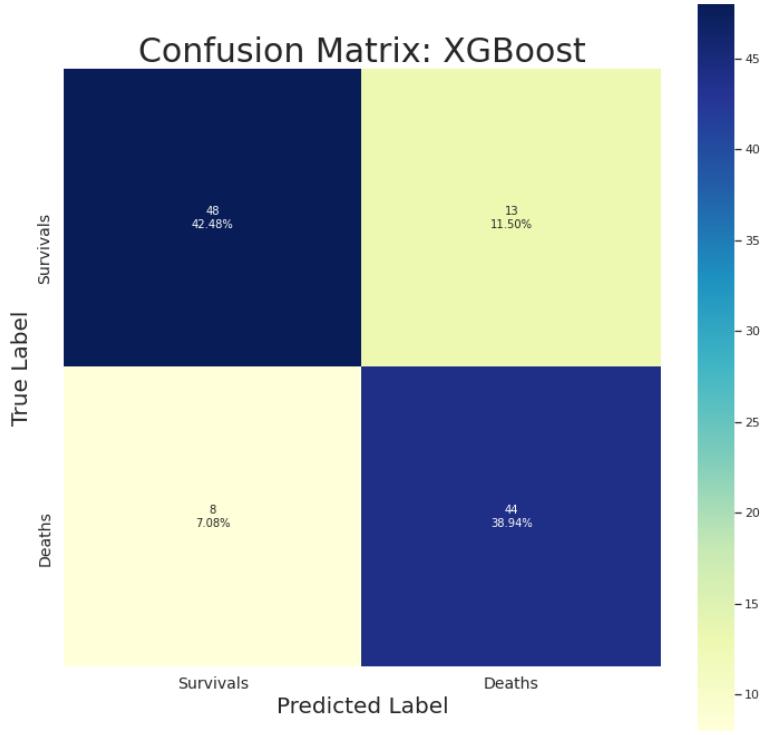
Tabela 1: Valores de Precision e Recall para ambos métodos utilizados.

Os F1-Scores obtidos foram de 0.82(4) para *Random Forest* e 0.80(7) para *Extreme Gradient Boosting*.

Resultados

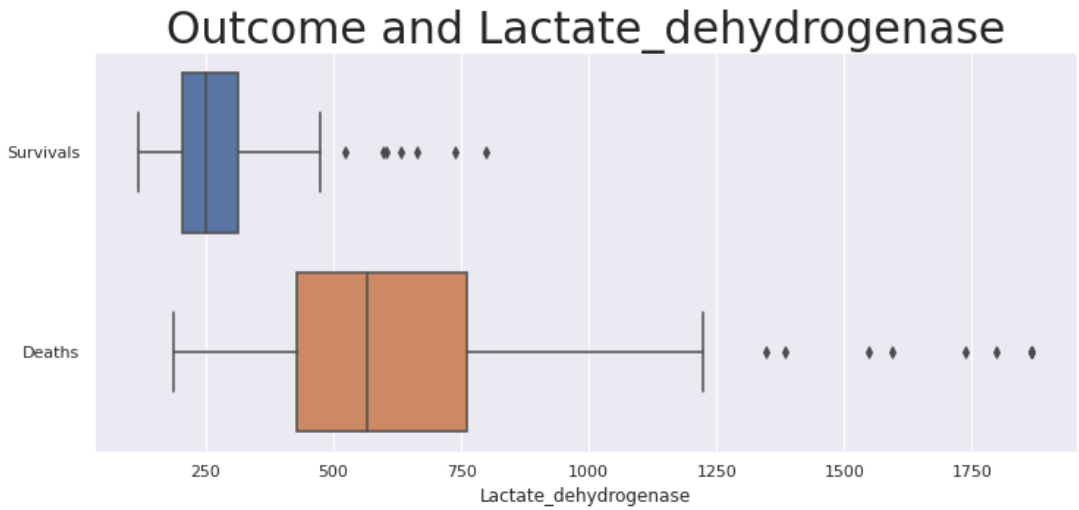
Vendo melhor os resultados em matriz de confusão.





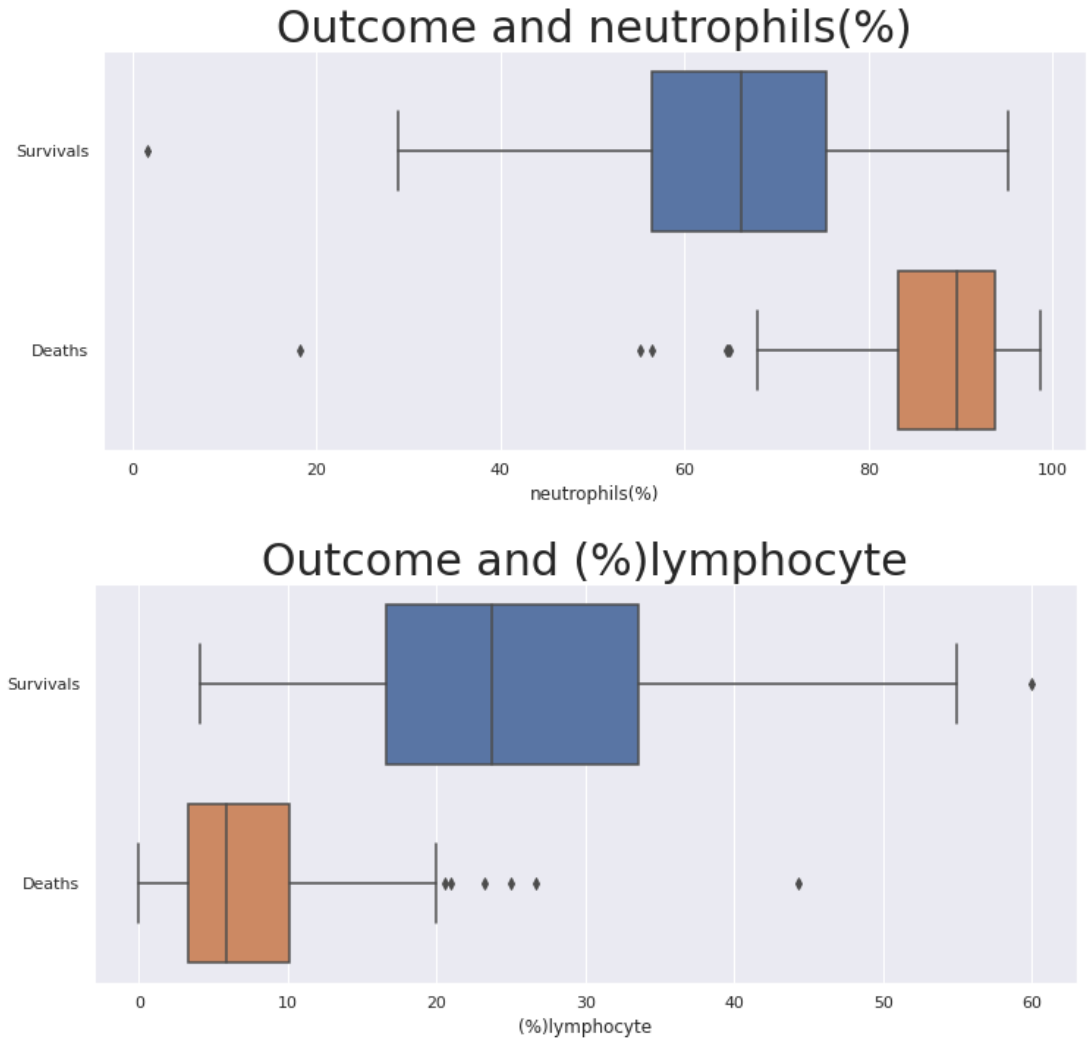
Resultados

Podemos realizar gráficos do tipo box-plot para visualizar melhor a distribuição dos biomarcadores para quem teve um resultado positivo ou negativo da internação.



Uma elevação dos níveis de DHL pode estar ligado a dano celular ou tecidual.

Resultados



O aumento de Neutrófilos (neutrofilia) com consequente diminuição de Linfócitos (lifocitopenia) é um indicador de infecção.

Comentários Finais

- Resultados
  - As acurácias obtidas foram 82% para *Random Forest* e 81% para *Extreme Gradient Boosting*;
  - Os biomarcadores obtidos tem relevância biológica, uma vez que eles dividem bem os grupos;
  - Alguns candidatos a biomarcadores a serem utilizados são: Desidrogenase Lática, porcentagem de Neutrófilos e porcentagem de Linfócitos;

- Perspectivas
  - Utilizar outros algoritmos de *ensemble*, como *Bayesian Model Averaging* (BMA) ou *Bayesian Model Combination* (BMC);
  - Buscar outros *datasets* para aplicar esse método.

Referências

- Yang et al, "*An interpretable mortality prediction model for COVID-19 patients*", 2020;
- `https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.` Acessado em Maio de 2021;
- `https://www.gov.br/pt-br/servicos-estaduais/exame-laboratorial-dosagem-de-desidrogenase-latica` Acessado em Maio de 2021;
- Zahorec R., "*Ratio of neutrophil to lymphocyte counts–rapid and simple parameter of systemic inflammation and stress in critically ill*", 2001;
- Rafael Júnior Ribeiro, `https://github.com/rjribeiro/covid_ic`;
- Todos códigos e dados estão disponíveis no meu github: `https://github.com/pedhmendes`

Agradecimentos

Obrigado a todos!