U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Students' dropout and academic success

INTELIGÊNCIA ARTIFICIAL, 3LEIC01,

GRUPO 21_1D:

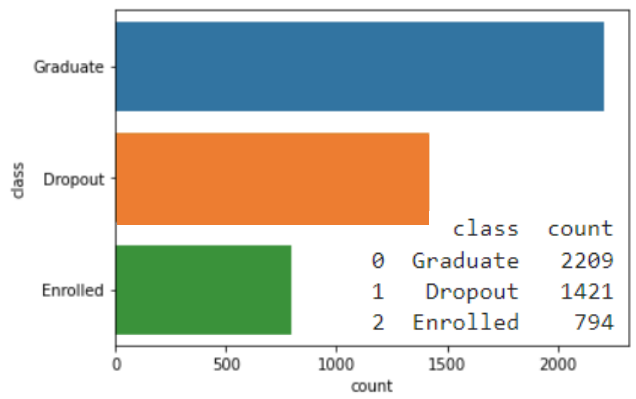
HENRIQUE RIBEIRO NUNES, UP201906852

MARGARIDA ASSIS FERREIRA, UP201905046

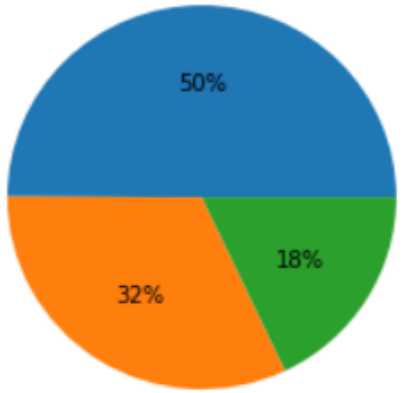
PATRÍCIA DO CARMO NUNES OLIVEIRA, UP201905427

```
# Get all 36 metrics used and 3 possible results
metrics = list(data.columns)
metrics.remove("class")
print("Number of Metrics: {}".format(len(metrics)))
print("Metrics: {}".format(metrics))

targets = list(data["class"].unique())
print("\nNumber of Results: {}".format(len(targets)))
print("Result/Prediction: {}".format(targets))
```

```
Number of Metrics: 36
Metrics: ['Marital status', 'Application mode', 'Application order', 'Course', 'Daytime/evening attendance\t', 'Previous
ification (grade)', 'Nacionality', "Mother's qualification", "Father's qualification", "Mother's occupation", "Father's
'Displaced', 'Educational special needs', 'Debtor', 'Tuition fees up to date', 'Gender', 'Scholarship holder', 'Age at e
urricular units 1st sem (credited)', 'Curricular units 1st sem (enrolled)', 'Curricular units 1st sem (evaluations)', 'C
ed)', 'Curricular units 1st sem (grade)', 'Curricular units 1st sem (without evaluations)', 'Curricular units 2nd sem (c
d sem (enrolled)', 'Curricular units 2nd sem (evaluations)', 'Curricular units 2nd sem (approved)', 'Curricular units 2n
ts 2nd sem (without evaluations)', 'Unemployment rate', 'Inflation rate', 'GDP']

Number of Results: 3
Result/Prediction: ['Dropout', 'Graduate', 'Enrolled']
```



	class	count
0	Graduate	2209
1	Dropout	1421
2	Enrolled	794



Problema "**single label multiclass classification**":

- 36 métricas distintas para descrever o aluno.
- 1 coluna objetivo com 3 resultados possíveis (***Dropout***, ***Enrolled***, ***Graduate***).

Objetivo: usar a informação sobre o percurso académico, demografia, fatores socioeconómicos e o desempenho académico dos alunos para construir modelos de classificação para prever o **sucesso académico e a desistência dos estudos**.

Forte **desbalanceamento** entre as três classes.

- Tipo de Organização dos dados = *Data Matrix*
- Dimensionalidade = *37*
- Tamanho do conjunto de dados = 4424

```
# Data size
print("Data Size: {}".format(len(data)))
```

```
Data Size: 4424
```

- Tipo e intervalo de valores de cada um dos atributos = *Nominal and Discrete (incluindo Binários)*

```
# Check atribute types and values interval
for name, dtype in data.dtypes.iteritems():
    print("{} | {} | [{} , {}] ".format(name.ljust(46), str(dtype).ljust(7),
                                        data[name].min(), data[name].max()))
```

```
Marital status                                 | int64   | [1 , 6]
Application mode                               | int64   | [1 , 57]
Application order                              | int64   | [0 , 9]
Course                                         | int64   | [33 , 9991]
Daytime/evening attendance                     | int64   | [0 , 1]
Previous qualification                         | int64   | [1 , 43]
Previous qualification (grade)                 | float64 | [95.0 , 190.0]
```

- Valores nulos = Não

```
# Check if there are columns with N/A values
print("N/A values found: {}".format(data.isnull().values.any()))
```

```
N/A values found: False
```

- Entradas duplicadas = Não

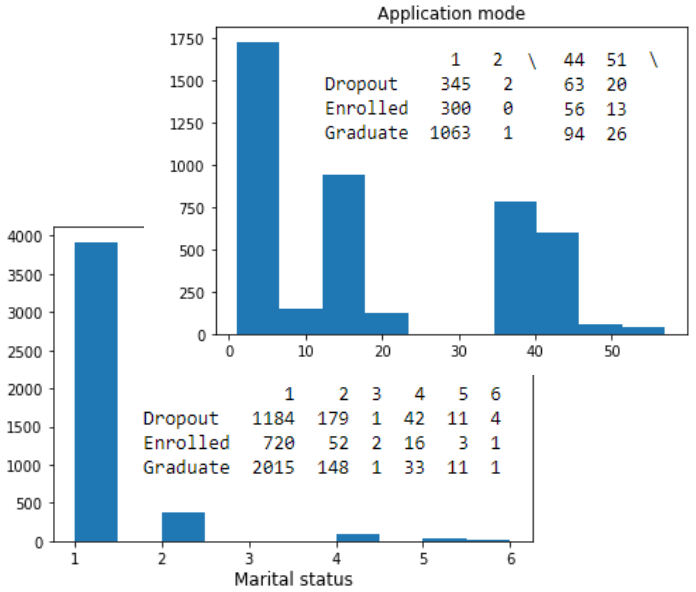
```
# Check if there are duplicated Data
bool_series = data.duplicated()
print(bool_series)

old_size = len(data)

# Removing all duplicated data if exists
data = data[~bool_series]

new_size = len(data)

# check if there were actualy duplicated data
print()
if (new_size == old_size):
    print("No data was removed: there were no duplicated data")
else:
    print("Was found and removed {} duplicated data".format(old_size-new_size))
```

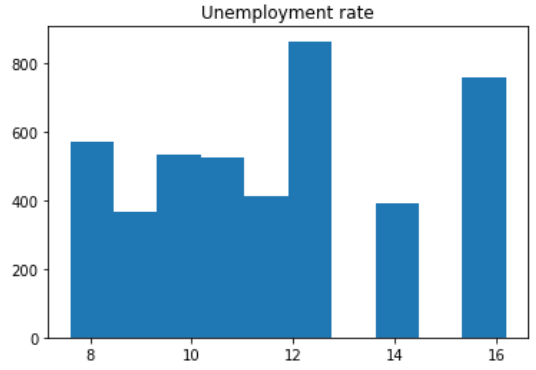
```
0       False
1       False
        ...
4422    False
4423    False
Length: 4424, dtype: bool

No data was removed: there were no duplicated data
```

- *Outliers* = Nenhuns identificados

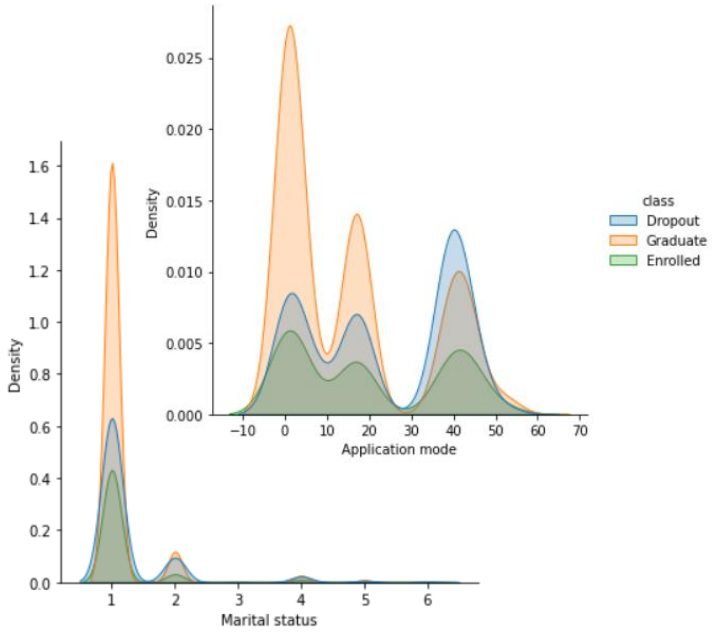
- Distribuição de valores por atributo

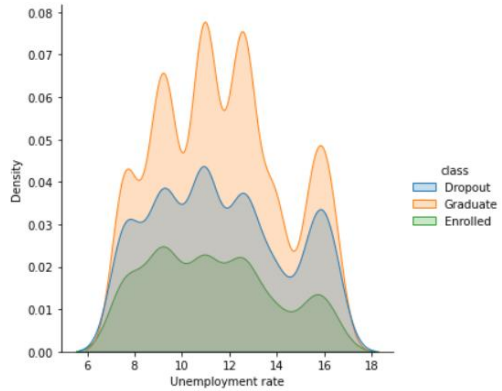
```
for i in range(36):
    plt.title(features[i])
    plt.hist(data[features[i]].values)
    plt.show()
    print(count_targets_for_each_value(data, features[i]))
```





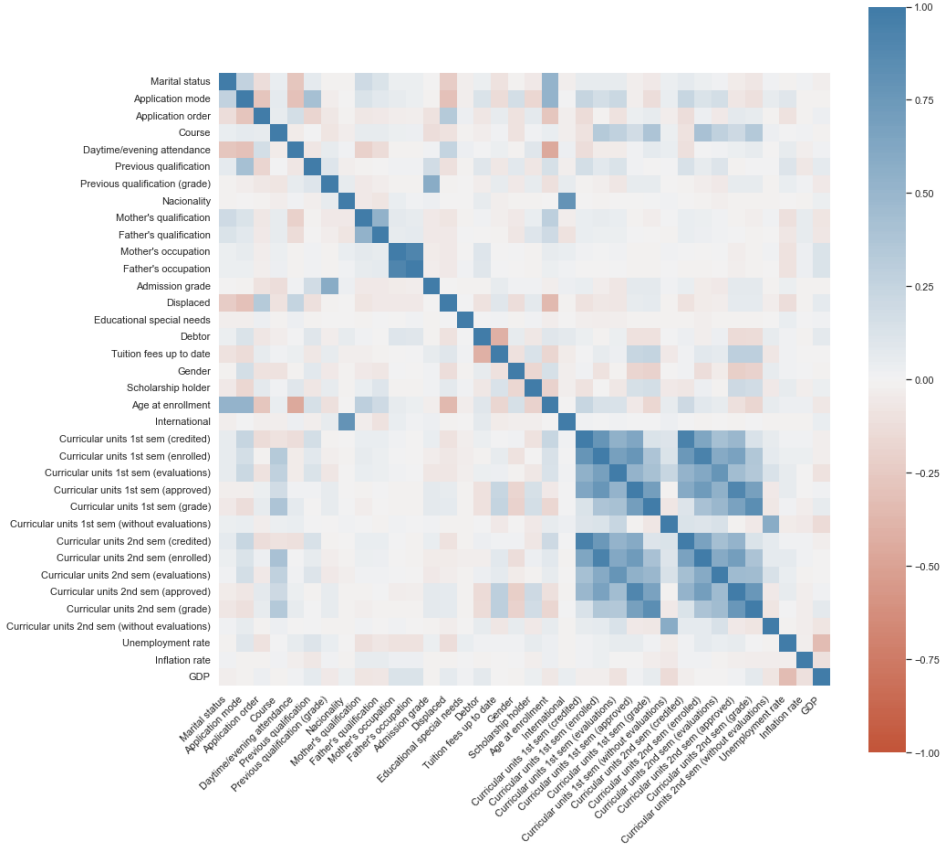

```
for attribute in attributes:
    densityPlot(attribute, 'class', True)
```





- Mapa de calor de correlação

```
# Correlation Heatmap
corr = data.corr()

sb.set(rc = {'figure.figsize': (15,15) })
ax = sb.heatmap(
    corr, vmin=-1, vmax=1, center=0,
    cmap=sb.diverging_palette(20, 240, n=200), square=True,
    xticklabels=True, yticklabels=True,)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right');
plt.show()
sb.reset_orig()
```



Data integration

- Símbolo de separação dos campos de '; ' para ','

Data cleaning

- Padronização dos dados (*standardization*) - apenas dos atributos não binários

```
def standardize(data, to_standardize):
    data_to_standardize = data[to_standardize]
    scaler = StandardScaler()
    stand_values = scaler.fit_transform(data_to_standardize.values)

    stand_values_df = pd.DataFrame(
        stand_values,
        index=data_to_standardize.index,
        columns=to_standardize)
    data[to_standardize] = stand_values_df[to_standardize]
    return data
```

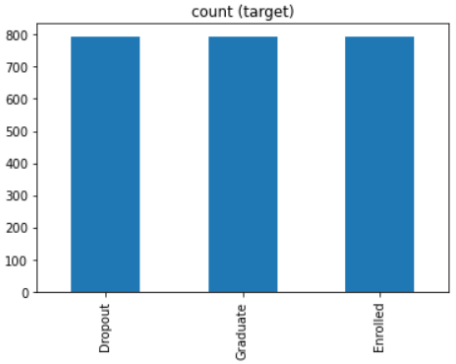
```
non_binary_features = \
    [feature for feature in features if len(data[feature].unique()) != 2]
data_standard = standardize(data.copy(), non_binary_features)
```

(A padronização é efetuada com base na média e no desvio padrão; Os algoritmos ficam menos sensíveis a *outliers*)

Data transformation

- Tratamento dos dados desbalanceados

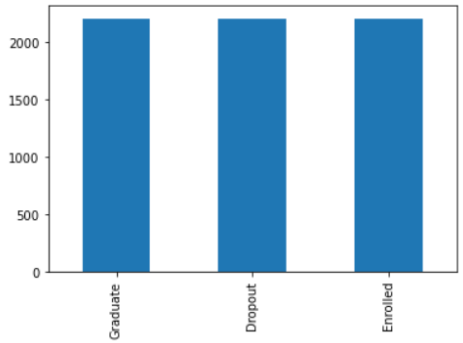
Dropout	794
Graduate	794
Enrolled	794



Under-sampling

(São removidas informações que podem ser valiosas)

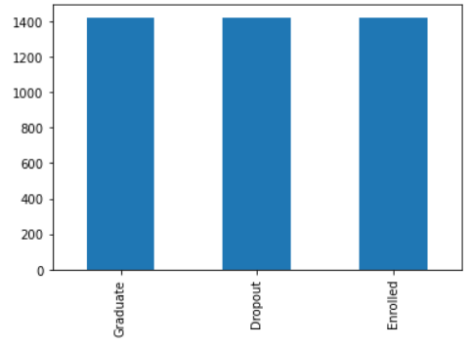
Graduate	2209
Dropout	2209
Enrolled	2209



Over-sampling

(Pode causar *overfitting* e pobre generalização)

Graduate	1421
Dropout	1421
Enrolled	1421



Combinação entre Under e Over-sampling

(Converge-se para o número de amostras da classe 'Dropout')

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(
  criterion= 'entropy',
  splitter= 'best',
  max_depth= 7,
  max_features = None,
  max_leaf_nodes= None)

if GRIDSEARCH: dtc = grid_search(DecisionTreeClassifier(), grid_params_dtc, values, targets)

test_report_dtc, train_report_dtc = validate(dtc, values, targets)
```

K-Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(
    n_neighbors = 9,
    weights = 'distance',
    metric = 'manhattan')

if GRIDSEARCH: knn = grid_search(KNeighborsClassifier(), grid_params_knn, values, targets)

test_report_knn, train_report_knn = validate(knn, values, targets)
```

Neural Network

```
from sklearn.neural_network import MLPClassifier

nnc = MLPClassifier(
    solver = 'sgd',
    alpha = 1e-5,
    hidden_layer_sizes = (40, 50),
    max_iter = 5000,
    random_state = 1,
    activation = 'relu',
    learning_rate = 'constant',
    learning_rate_init = 0.01)

if GRIDSEARCH and False: nnc = grid_search(MLPClassifier(), grid_params_nnc, values, targets)

test_report_nnc, train_report_nnc = validate(nnc, values, targets)
```

Support Vector Machine

```
from sklearn.svm import SVC

svm = SVC(
    decision_function_shape='ovr',
    C = 10,
    gamma=0.01,
    kernel='rbf')

if GRIDSEARCH: svm = grid_search(SVC(), grid_params_svm, values, targets)

test_report_svm, train_report_svm = validate(svm, values, targets)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(
    n_estimators=40,
    max_depth=None,
    max_features=None,
    min_samples_leaf=1,
    min_samples_split=2)

if GRIDSEARCH: rfc = grid_search(RandomForestClassifier(), grid_params_rfc, values, targets)

test_report_rfc, train_report_rfc = validate(rfc, values, targets)
```

Extra Trees

```
from sklearn.ensemble import ExtraTreesClassifier

etc = ExtraTreesClassifier(
    n_estimators=40,
    max_depth=100,
    min_samples_split=2,
    random_state=None)

if GRIDSEARCH: etc = grid_search(ExtraTreesClassifier(), grid_params_etc, values, targets)

test_report_etc, train_report_etc = validate(etc, values, targets)
```

Parameter Tunning

- Ajuste e refinamento dos parâmetros a usar na construção dos modelos por cada um dos algoritmos com ***GridSearch.***

```python
from sklearn.model_selection import GridSearchCV

def grid_search(model, grid_params, features, targets):
    test_size = 0.2
    cross_validation_split = 5
    feat_train, feat_test, target_train, target_test = split_data(features, targets, test_size)

    gs = GridSearchCV(
        model,
        grid_params,
        verbose = 1,
        n_jobs = 1,
        cv = cross_validation_split,
    )

    gs_results = gs.fit(feat_train, target_train)

    print("best score: " + str(gs_results.best_score_))
    print("best estimator: " + str(gs_results.best_estimator_))
    print("best parameters: " + str(gs_results.best_params_))

    return gs_results.best_estimator_
```

```python
grid_params_dtc = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'splitter': ['best', 'random'],
    'max_depth': [5, 7, 10, 20],
    'max_features': ['sqrt', 'log2', None],
    'max_leaf_nodes': [None, 5, 10, 20],
}
```

```python
if GRIDSEARCH: dtc = grid_search(DecisionTreeClassifier(), grid_params_dtc, values, targets)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
best score: 0.6876640419947506
best estimator: DecisionTreeClassifier(criterion='log_loss', max_depth=20)
best parameters: {'criterion': 'log_loss', 'max_depth': 20, 'max_features':
None, 'max_leaf_nodes': None, 'splitter': 'best'}
```

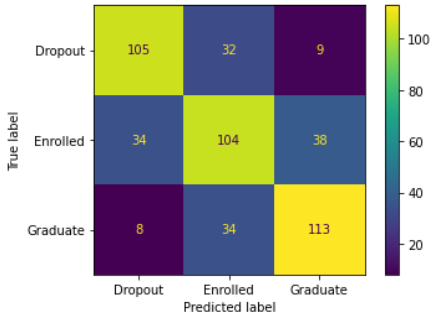
- O modelo avaliado corresponde ao melhor estimador encontrado pelo ***GridSearch***.
- Por defeito, por razões de performance e tempo, o *GridSearch* não é computado, sendo utilizados como parâmetros de cada algoritmo um conjunto de parâmetros defeito encontrados previamente com *GridSearch*.

```python
def validate(model, features, targets):
    test_size = 0.2
    cross_validation_split = 5
    feat_train, feat_test, target_train, target_test = split_data(features, targets, test_size)

    model.fit(feat_train, target_train)
    predictions = model.predict(feat_test)

    acc = metrics.accuracy_score(target_test, predictions)
    print("Accuracy: {:.2f}\n".format(acc*100))

    cm = confusion_matrix(target_test, predictions)
    print("Precision:")
    for i in range(3): print("\t{:.2f}% - {}".format(precision(cm, i)*100, classes[i]))
    print("Recall:")
    for i in range(3): print("\t{:.2f}% - {}".format(recall(cm, i)*100, classes[i]))
    print("F1-measure:")
    for i in range(3): print("\t{:.2f}% - {}".format(f_measure(cm, i), classes[i]))

    scores = cross_validation(model, features, targets, cross_validation_split)
    print("Cross validation {}-fold: {}".format(cross_validation_split, [round(x, 3) for x in scores]))
    print("\t{:.2f} - accuracy".format(scores.mean()))
    print("\t{:.2f} - standard deviation".format(scores.std()))

    test_report = metrics.classification_report(target_test, predictions)
    print("\nTest Classification Report:\n", test_report)

    predictions_train = model.predict(feat_train)
    train_report = metrics.classification_report(target_train, predictions_train)
    print("\nTrain Classification Report:\n", train_report)

    display_confusion_matrix(cm)

    return test_report, train_report
```

(Decision Tree Classifier)

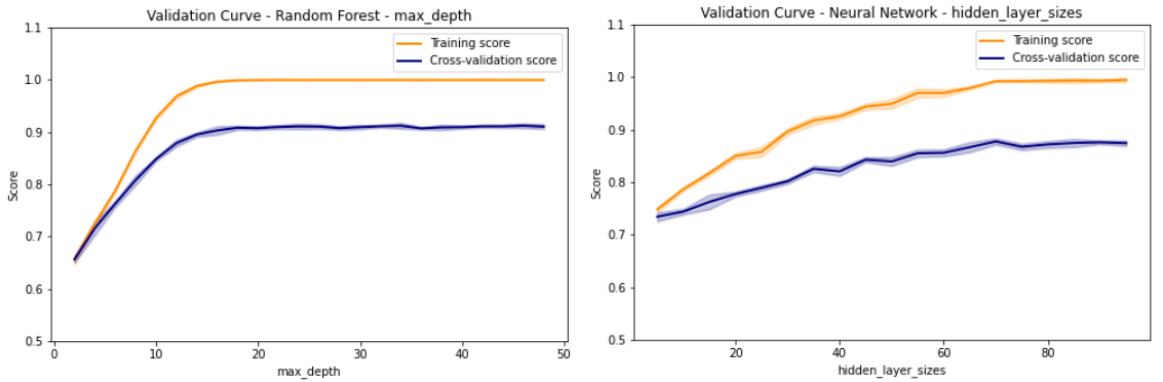
[Standardization + Under sampling]

```
Accuracy: 67.51

Precision:
        71.43% - Dropout
        61.18% - Enrolled
        70.62% - Graduate
Recall:
        71.92% - Dropout
        59.09% - Enrolled
        72.90% - Graduate
F1-measure:
        71.67% - Dropout
        60.12% - Enrolled
        71.75% - Graduate
Cross validation 5-fold: [0.656, 0.662, 0.666, 0.622, 0.687]
        0.66 - accuracy
        0.02 - standard deviation
```



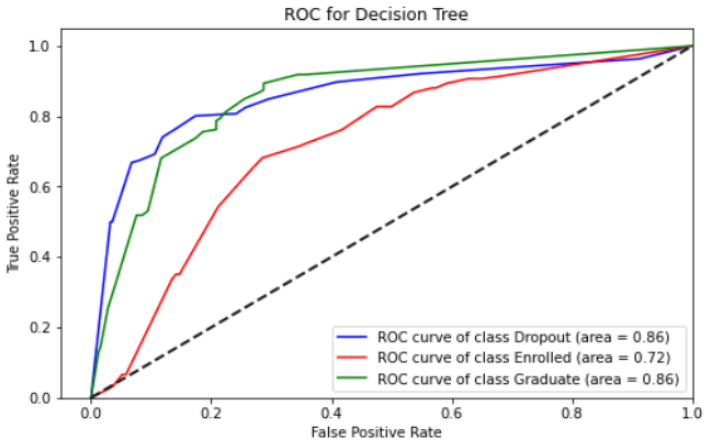

```
Test Classification Report:
              precision    recall  f1-score   support

     Dropout       0.71      0.72      0.72       146
    Enrolled       0.61      0.59      0.60       176
    Graduate       0.71      0.73      0.72       155

    accuracy                           0.68       477
   macro avg       0.68      0.68      0.68       477
weighted avg       0.67      0.68      0.67       477

Train Classification Report:
              precision    recall  f1-score   support

     Dropout       0.87      0.79      0.83       648
    Enrolled       0.69      0.73      0.71       618
    Graduate       0.78      0.81      0.80       639

    accuracy                           0.78      1905
   macro avg       0.78      0.78      0.78      1905
weighted avg       0.78      0.78      0.78      1905
```

Overfit dos modelos



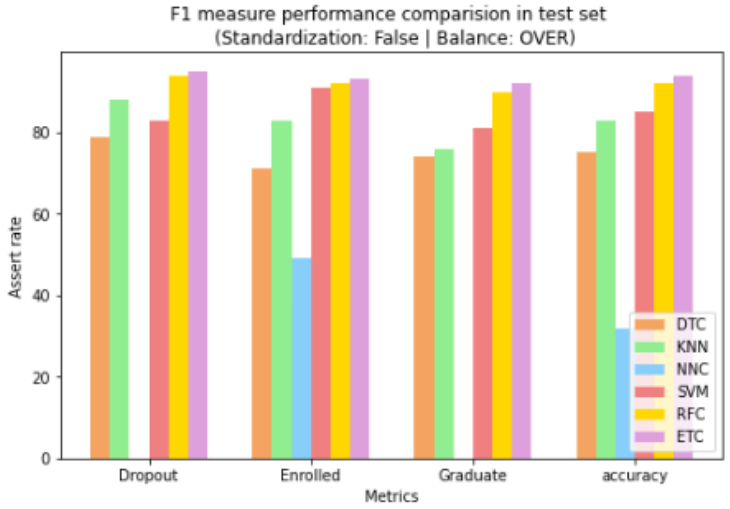
- Independentemente do modelo utilizado é necessário fazer uma avaliação dos parâmetros utilizados para evitar *overfit*.

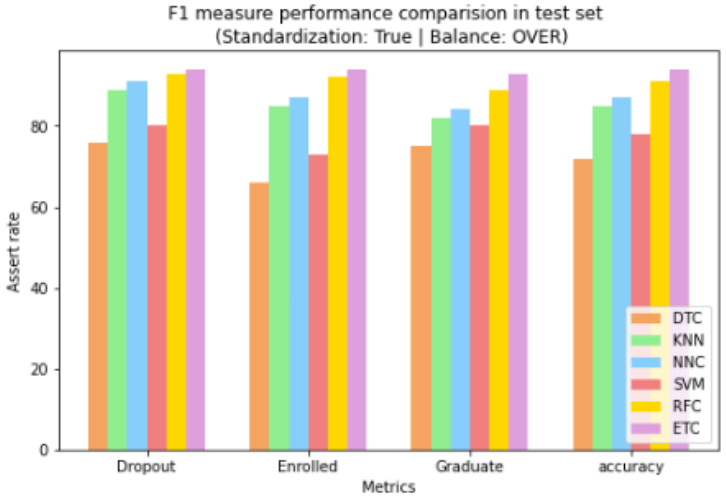
ROC Curve



- Utilização do objeto *OneVsRestClassifier* para a obtenção da curva ROC para cada uma das classes possíveis para cada modelo.

Impacto de padronização dos dados na performance dos modelos





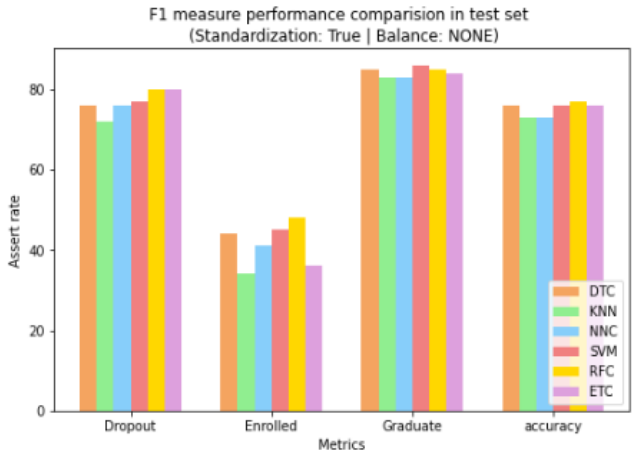
- Independentemente do tipo de técnica usada para corrigir o balanceamento dos dados, os modelos criados sem padronização apresentam baixa performance.
- Caso, por exemplo, da ***Neural Network***, que não lida bem com pobres dados de entrada.
- Construir os modelos sobre os dados com padronização tem um grande impacto na melhoria da performance geral de todos os algoritmos, para qualquer técnica de balanceamento.

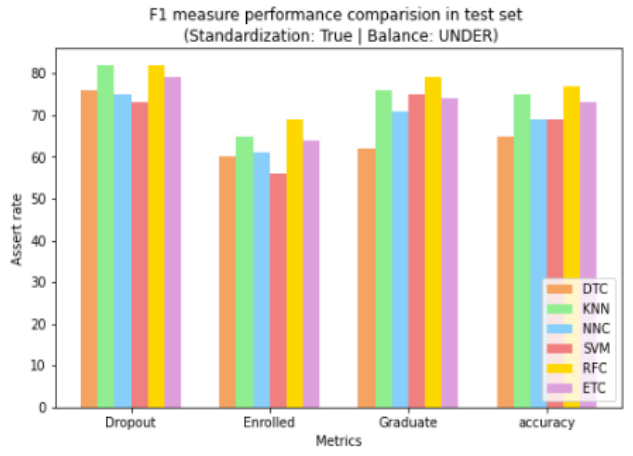
Impacto da técnica de balanceamento dos dados na performance dos modelos

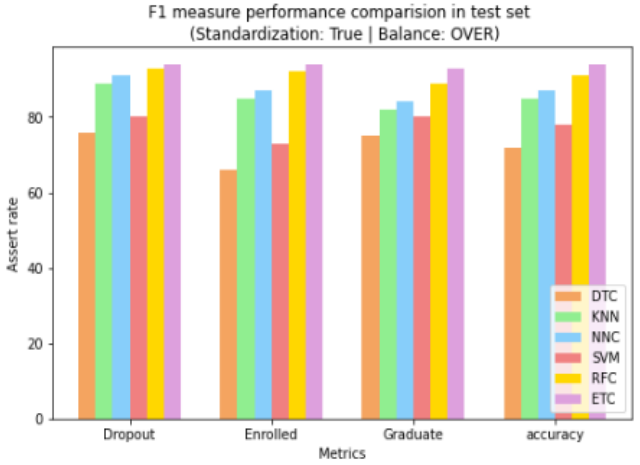
- Sem aplicar qualquer técnica de balanceamento os resultados obtidos são evidentemente piores, principalmente na classe menos representada (***Enrolled***).

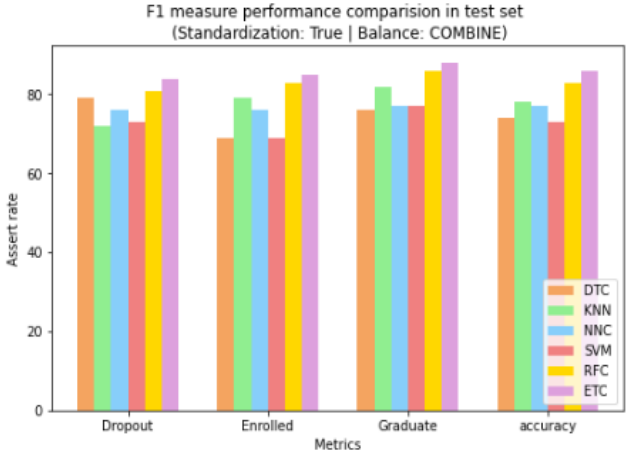
- Concluímos que obtemos melhores resultados processando os dados **com padronização**, balanceamento ***oversampling*** e utilização do modelo ***Random Forest***, obtendo 5-*fold cross-validation* ***accuracy*** **de 92%**.

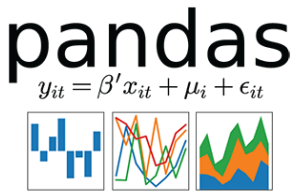
- No entanto, o **modelo *SVM*** obtido, que tem um *accuracy* apenas de 77%, aparenta sofrer menos de *overfit* (83% vs 100% de *accuracy* para quando validadas as amostras de treino) por isso achamos que pode ser uma **opção mais segura** para prever o sucesso académico neste conjunto de dados.











- https://towardsdatascience.com/how-to-balance-a-dataset-in-python-36dff9d12704
- https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/
- https://www.machinelearningplus.com/pandas/pandas-duplicated/
- https://seaborn.pydata.org/tutorial/distributions.html
- https://scikit-learn.org/stable/user_guide.html
- https://www.geeksforgeeks.org/plotting-multiple-bar-charts-using-matplotlib-in-python/
- https://ai.plainenglish.io/hyperparameter-tuning-of-decision-tree-classifier-using-gridsearchcv-2a6ebcaffeda
- https://medium.datadriveninvestor.com/hyperparameter-tuning-with-deep-learning-grid-search-8630aa45b2da
- https://www.sciencedirect.com/science/article/pii/S2666920X22000212
- https://www.researchgate.net/publication/340406248