FairPEK - Documentação

Manutenção

October 17, 2022

1 Introdução

Esta documentação foi criada com o objetivo de guiar o desenvolvedor de Software a entender, configurar e manter este sistema, que é dividido em 4 etapas principais:

- **Engenharia de dados:** Etapa criada com o objetivo de simular processos de transformação e limpeza de dados.
- **Pipeline de IA:** Etapa para execução de um Pipeline que simula o desenvolvimento de uma aplicação automatizada de IA, desde uma categorização dos dados mais específica do que na etapa anterior, passando pelo algoritmo utilizado e finalizando obtendo métricas para determinar qualidade do resultado final.
- **Autonomia do Pipeline (Componente MAPE-K):** Etapa que executa um componente para automatizar todas as etapas do Pipeline, com o objetivo de evitar com que perca-se tempo em execuções manuais que podem demorar dependendo do algoritimo e do conjunto de dados utilizado.
- **Interface:** Etapa criada com o objetivo de simular a etapa anterior, porém de modo a proporcionar uma experiência de usuário mais simples e intuitiva. É dividida em duas partes:
  - **Frontend:** Parte visual, exibida em um navegador.
  - **Backend:** Parte onde o Frontend se comunica para obter os dados e montar o visual corretamente, de forma que corresponda a configurações utilizadas pelo Componente MAPE-K.

2 Arquitetura do Pipeline e Framework

O Pipeline usa uma arquitetura chamada de Pipe-and-Filter, onde Pipes, que transportam os dados, são ligados por Filters, que realizam as manipulações desses dados. Pipes e Filters se encadeiam para formar uma sequência de operações, caracterizando todo o Pipeline. Para auxiliar no desenvolvimento, um pequeno framework foi criado, para facilitar as operações necessárias entre Pipes e Filters.

2.1 Estrutura

Pipes herdam a classe **BasePipe**. Essa classe possui o atributo **value**, que caracteriza os dados transformados, em formato de um dicionário Python

```
class FairnessPipe(BasePipe):
    privileged\_group = []
    unprivileged_group = []

    label_names = []
    protected_attribute_names = []

    optim_options = {}

    def __init__(self):
        self.value = {
            'privileged_group': self.privileged_group,
            'unprivileged_group': self.unprivileged_group,
            'label_names': self.label_names,
            'protected_attribute_names': self.
    protected_attribute_names,
            'optim_options': self.optim_options
        }
```

Filters herdam a classe **BaseFilter**. Essa classe possui dois Pipes (input e output) e um método chamado **execute**, que é onde as operações de transformação do Pipe de input são executadas para serem colocadas no Pipe de output

```
from sklearn.model_selection import train_test_split

from src.pipeline.pipe_filter.pipe import BaseFilter


class TrainTestSplit(BaseFilter):
    test_size = 0.2

    def __init__(self, test_size=0.2):
        self.test_size = test_size

    def execute(self):
        df_x = self.input['df_x']
        df_y = self.input['df_y']

        x_train, x_test, y_train, y_test = train_test_split(df_x,
    df_y, test_size=self.test_size, random_state=42)

        self.output = {
            'x_train': x_train,
            'x_test': x_test,
            'y_train': y_train,
            'y_test': y_test,
            'checksum': self.input['checksum']
        }
```

2.2 Operações

No Framework, estão presentes as seguintes operações:

2.3 Ligação de Pipe com Filter

Para juntar um Pipe com um Filter, basta realizar o comando >=, caracterizando o transporte a um Filter.

```
Pipe1() >= Filter1()
```

Se o Pipe estiver carregando os dados corretos, o Filter será automaticamente executado.

2.4 Ligação de Filter com Pipe

Para juntar um Filter com um Pipe, basta realizar o comando ==, caracterizando o transporte a um Pipe.

```
Filter1() == Pipe1()
```

Se o Filter esiver corretamente implementado, o Pipe será caracterizado como a saída desse mesmo Filter.

2.5 Seleção parcial de dados presentes no Pipe

Para selecionar apenas alguns atributos presentes no Pipe, basta adicionar colchetes e colocar os campos desejados.

```
pipe1['campo1', 'campo2', 'campo3']
```

2.6 Junção de Pipes

Pra juntar os dados de dois pipes em um só, basta realizar o comando +, caracterizando uma junção de Pipes.

```
pipe1 + pipe2
```

3 Incrementos no Pipeline

Aqui estão dois exemplos de como é possível realizar a manuntenção do Pipeline

3.1 Adicionando um novo Conjunto de Dados

3.1.1 Engenharia de dados

1. Geralmente conjuntos de dados não vem filtrados, e é preciso um trabalho de Engenharia de dados para realizar experimentos com melhores resultados. Neste sistema, a parte de Engenharia de dados se encontra na pasta **src/data_engineering**.

2. Os métodos onde os processamentos são realizados ficam no arquivo **data_engineering.py**. Para adicionar um novo processamento, basta adicionar um novo método neste arquivo.

3. Importar o novo método no arquivo **data_engineering_start.py**.

4. Adicionar uma nova opção no parâmetro **choices** presente no método **parser.add_argument**

```
    parser.add_argument("--data", help="Selecao do gerador do
    conjunto de dados tratado",
                        choices=['GERMAN_CREDIT', 'LENDINGCLUB
    ', 'METRICS'])
```

5. Adicionar uma nova condição com a opção adicionada

3.1.2 Pipeline

1. Neste sistema, a parte do Pipeline se encontra na pasta **src/pipeline**. Dentro dela, os Pipes que armazenam os conjuntos de dados se encontram na pasta **processors/preprocessors/data**.

2. No arquivo **dataset.py**, criar uma classe que herda a classe **Dataset**, preenchendo os atributos **dataset_path** com o caminho do arquivo.

```
class LendingclubDataset(Dataset):
    dataset_path = 'datasets/lendingclub_dataset.csv'

    def __init__(self):
        super().__init__()
```

3. Criar um novo arquivo.

4. Criar uma classe que herda a classe **FairnessPipe**, preenchendo os atributos **privileged_group**, **unprivileged_group**, **label_names**, **protected_attribute_names** e **optim_options**.

```
class LendingclubIncomeFairnessPipe(FairnessPipe):
    privileged_group = [{'annual_inc': 1}]
    unprivileged_group = [{'annual_inc': 0}]

    label_names = ['loan_status']
    protected_attribute_names = ['annual_inc']

    optim_options = {
        "distortion_fun": get_distortion_german,
        "epsilon": 0.05,
        "clist": [0.99, 1.99, 2.99],
        "dlist": [.1, 0.05, 0]
    }

    def __init__(self):
        super().__init__()
```

5. Criar uma classe que herda a classe **FairnessPreprocessor**, preenchendo o método **dataset_preprocess** e retornando um DataFrame Pandas de dados de entrada e um DataFrame de dados de saída.

```
class LendingclubIncomePreprocessor(FairnessPreprocessor):
    def dataset_preprocess(self, df):
        df.info()

        SAMPLE_PERCENTAGE = 100
        df_sample_nok = df[df['loan_status'] == 'Charged Off'].sample(frac=SAMPLE_PERCENTAGE/100)
        df_sample_ok = df[df['loan_status'] == 'Fully Paid'].sample(frac=SAMPLE_PERCENTAGE / 100)
        df_sample = pd.concat([df_sample_ok, df_sample_nok])

        df_x = df_sample.drop('loan_status', axis=1)
        df_y = pd.DataFrame(df_sample.loan_status)

        return df_x, df_y
```

6. No arquivo **enums.py**, colocar novas opções nas classes **Datasets** e **Preprocessors**.

```
class Datasets(ExtendedEnum):
    ADULT_INCOME = 1
    GERMAN_CREDIT = 2
    LENDINGCLUB = 3


class Preprocessors(ExtendedEnum):
    SEX = 1
    AGE = 2
    FOREIGN = 3
    INCOME = 4
```

7. No arquivo **validation.py**, atualizar a variável **existant_preprocessors** com as novas opções colocadas no item anterior.

```
        existant_preprocessors = \
            (dataset == Datasets.ADULT_INCOME and preprocessor == Preprocessors.SEX) or \
            (dataset == Datasets.GERMAN_CREDIT and preprocessor == Preprocessors.AGE) or \
            (dataset == Datasets.GERMAN_CREDIT and preprocessor == Preprocessors.FOREIGN) or \
            (dataset == Datasets.LENDINGCLUB and preprocessor == Preprocessors.INCOME)
```

8. No arquivo **pipeline.py**, encontrar o método **select_data_preprocessor**, atualizar a variável **options** com as novas opções e classes implementadas nos itens anteriores.

```
    def select_data_preprocessor(self, dataset, preprocessor):
        choice = [dataset, preprocessor]
        options = [
```

```
            ([Datasets.ADULT_INCOME, Preprocessors.SEX], (AdultDataset(), AdultSexPreprocessor(), AdultSexFairnessPipe())),
            ([Datasets.GERMAN_CREDIT, Preprocessors.AGE], (GermanDataset(), GermanAgePreprocessor(), GermanAgeFairnessPipe())),
            ([Datasets.GERMAN_CREDIT, Preprocessors.FOREIGN], (GermanDataset(), GermanForeignPreprocessor(), GermanForeignFairnessPipe())),
            ([Datasets.LENDINGCLUB, Preprocessors.INCOME], (LendingclubDataset(), LendingclubIncomePreprocessor(), LendingclubIncomeFairnessPipe())),
        ]

        for option, pipe_filter in options:
            if choice == option:
                dataset_pipe, data_preprocessor_filter, fairness_pipe = pipe_filter
                preprocessed_data_pipe = dataset_pipe >> data_preprocessor_filter == self.new_pipe()
                break

        return preprocessed_data_pipe, fairness_pipe
```

9. No arquivo **pipeline_start.py**, adicionar uma nova opção no parâmetro **choices** presente no método **parser.add_argument**.

```
    parser.add_argument("--dataset", help="Conjunto de dados tratado com atributo protegido",
                        choices=['ADULT_INCOME_SEX',
                                 'GERMAN_CREDIT_FOREIGN', 'GERMAN_CREDIT_AGE',
                                 'LENDINGCLUB_INCOME'])
```

10. Adicionar uma nova condição com a opção adicionada

3.1.3 Interface (Backend)

1. Neste sistema, a parte do Backend se encontra na pasta **src/api**. Dentro dela, no arquivo **repo/pipeline.py**, adicionar a opção no método **get_dataset**.

```
    def get_dataset(self, dataset):
        indexes = {
            'Datasets.ADULT_INCOME': Datasets.ADULT_INCOME,
            'Datasets.GERMAN_CREDIT': Datasets.GERMAN_CREDIT,
            'Datasets.LENDINGCLUB': Datasets.LENDINGCLUB,
        }

        return next(filter(lambda a: a[0] == dataset, indexes.items()))[1]
```

2. No arquivo **repo/pipeline.py**, adicionar a opção no método **get_preprocessor**.

```
    def get_preprocessor(self, preprocessor):
        indexes = {
            'Preprocessors.SEX': Preprocessors.SEX,
            'Preprocessors.AGE': Preprocessors.AGE,
            'Preprocessors.FOREIGN': Preprocessors.FOREIGN
        }

        return next(filter(lambda a: a[0] == preprocessor,
    indexes.items()))[1]
```

3.1.4 Interface (Frontend)

1. Neste sistema, a parte do Backend se encontra na pasta **ml-ui/src**. Dentro dela, no arquivo **Auto-Pipeline-Menu.js**, adicionar a opção colocada na etapa de Pipeline no componente Select onde estão as outras opções de conjunto de dados.

```
<Select
  sx={{fontSize: '14px'}}
  value={dataset}
  onChange={handleDatasetChange}
  displayEmpty
  inputProps={{ 'aria-label': 'Without label' }}
>
  <MenuItem value={'Datasets.ADULT_INCOME'}>Adult Income
    Dataset</MenuItem>
  <MenuItem value={'Datasets.GERMAN_CREDIT'}>German Credit
    Dataset</MenuItem>
  <MenuItem value={'Datasets.LENDINGCLUB'}>Lendingclub Dataset
    </MenuItem>
</Select>
```

2. No arquivo **Manual-Pipeline-Menu.js**, adicionar a opção colocada na etapa de Pipeline no componente Select onde estão as outras opções de conjunto de dados.

```
<Select
  sx={{fontSize: '14px'}}
  value={dataset}
  onChange={handleDatasetChange}
  displayEmpty
  inputProps={{ 'aria-label': 'Without label' }}
>
  <MenuItem value={'Datasets.ADULT_INCOME'}>Adult Income
    Dataset</MenuItem>
  <MenuItem value={'Datasets.GERMAN_CREDIT'}>German Credit
    Dataset</MenuItem>
  <MenuItem value={'Datasets.LENDINGCLUB'}>Lendingclub Dataset
    </MenuItem>
</Select>
```

3. No arquivo **Manual-Pipeline-Menu.js**, adicionar a opção colocada na etapa de Pipeline no componente Select onde estão as outras opções de pré-processadores.

```
<FormControl sx={{ m: 1, width: 300, marginLeft: '35px' }}>
  {dataset === 'Datasets.ADULT_INCOME' ?
    <Select
      sx={{fontSize: '14px'}}
      value={protectedAtt}
      onChange={handleProtectedAttChange}
      displayEmpty
      inputProps={{ 'aria-label': 'Without label' }}
    >
      <MenuItem value={'Preprocessors.SEX'}>Sexo (Masculino/
    Feminino)</MenuItem>
    </Select>
  : dataset === 'Datasets.ADULT_INCOME' ?
    <Select
      sx={{fontSize: '14px'}}
      value={protectedAtt}
      onChange={handleProtectedAttChange}
      displayEmpty
      inputProps={{ 'aria-label': 'Without label' }}
    >
      <MenuItem value={'Preprocessors.AGE'}>Idade (-25 anos
    /+25 anos)</MenuItem>
      <MenuItem value={'Preprocessors.FOREIGN'}>Nacionalidade
    (Local/Estrangeiro)</MenuItem>
    </Select>
  :
    <Select
      sx={{fontSize: '14px'}}
      value={protectedAtt}
      onChange={handleProtectedAttChange}
      displayEmpty
      inputProps={{ 'aria-label': 'Without label' }}
    >
      <MenuItem value={'Preprocessors.INCOME'}>Renda (-1
    Salario Minimo/1+ Salarios Minimos)</MenuItem>
    </Select>
  }
  <FormHelperText>Atributo protegido para medir justica</
    FormHelperText>
</FormControl>
```

3.2 Adicionando um novo Algoritmo

3.2.1 Pipeline

1. No arquivo **enums.py**, colocar novas opções na classe **Algorithms**.

```
class Algorithms:
    LOGISTIC_REGRESSION = 1
    RANDOM_FOREST = 2
    GRADIENT_BOOST = 3
    SUPPORT_VECTOR_MACHINES = 4
    LINEAR_REGRESSION = 901
    DECISION_TREE = 902
    KERNEL_RIDGE = 903
```

2. No arquivo **pipeline.py**, colocar as novas opções colocadas no item anterior no método **find_algorithm**.

```
    def find_algorithm(self, algorithm):
        indexes = {
            'Algorithms.LOGISTIC_REGRESSION': 1,
            'Algorithms.RANDOM_FOREST': 2,
            'Algorithms.GRADIENT_BOOST': 3,
            'Algorithms.SUPPORT_VECTOR_MACHINES': 4,
            'Algorithms.LINEAR_REGRESSION': 901,
            'Algorithms.DECISION_TREE': 902,
            'Algorithms.KERNEL_RIDGE': 903,
            'UnbiasInProcAlgorithms.PREJUDICE_REMOVER': 101,
            'UnbiasInProcAlgorithms.ADVERSARIAL_DEBIASING':
    102,
            'UnbiasInProcAlgorithms.
    EXPONENTIATED_GRADIENT_REDUCTION': 103,
            'UnbiasInProcAlgorithms.RICH_SUBGROUP_FAIRNESS':
    104,
            'UnbiasInProcAlgorithms.GRID_SEARCH_REDUCTION':
    105,
            'UnbiasInProcAlgorithms.META_FAIR_CLASSIFIER':
    106,
            'UnbiasInProcAlgorithms.ART_CLASSIFIER': 107
        }

        return next(filter(lambda a: a[1] == algorithm,
    indexes.items()))[0]
```

3. Os Filters que armazenam os conjuntos de dados se encontram na pasta **processors/inprocessors**. Nesta pasta, criar um novo arquivo.

4. Os Filters que executam os algoritmos se encontram na pasta **processors/inprocessors/inproc_algorithms**. Nesta pasta, criar um novo arquivo.

5. Criar uma classe que herda a classe **BaseFilter**.

```
class GradientBoostFilter(BaseFilter):
    weighed = False

    def __init__(self, weighed=False):
        self.weighed = weighed
```

6. Implementar nesta classe o método **execute**, atribuindo em **self.output** um dicionário Python com os atributos **y_pred** e **scores**.

```
    def execute(self):
        y_pred, scores = self.gradient_boost_weighed() if self
    .weighed else self.gradient_boost()

        self.output = {
            'y_pred': y_pred,
            'scores': scores
        }
```

7. Encontrar o método **process**, atualizar a variável **process_options** com as novas opções e classes implementadas nos itens anteriores.

```
    def process(self, process_pipe, algorithm,
    unbias_data_algorithm):
        weighed_algorithm = unbias_data_algorithm ==
    UnbiasDataAlgorithms.REWEIGHING or \
                            unbias_data_algorithm ==
    UnbiasDataAlgorithms.LEARNING_FAIR_REPRESENTATIONS

        process_options = [
            (Algorithms.LOGISTIC_REGRESSION,
    LogisticRegressionFilter(weighed=weighed_algorithm)),
            (Algorithms.RANDOM_FOREST, RandomForestFilter(
    weighed=weighed_algorithm)),
            (Algorithms.GRADIENT_BOOST, GradientBoostFilter(
    weighed=weighed_algorithm)),
            (Algorithms.SUPPORT_VECTOR_MACHINES, SVMFilter(
    weighed=weighed_algorithm)),
            (UnbiasInProcAlgorithms.PREJUDICE_REMOVER,
    PrejudiceRemoverFilter()),
            (UnbiasInProcAlgorithms.ADVERSARIAL_DEBIASING,
    AdversarialDebiasingFilter()),
            (UnbiasInProcAlgorithms.
    EXPONENTIATED_GRADIENT_REDUCTION,
    ExponentiatedGradientReductionFilter(algorithm=Algorithms.
    GRADIENT_BOOST)),
            (UnbiasInProcAlgorithms.RICH_SUBGROUP_FAIRNESS,
    RichSubgroupFairnessFilter(algorithm=Algorithms.
    DECISION_TREE)),
            (UnbiasInProcAlgorithms.META_FAIR_CLASSIFIER,
    MetaFairClassifierFilter()),
            (UnbiasInProcAlgorithms.GRID_SEARCH_REDUCTION,
    GridSearchReductionFilter(algorithm=Algorithms.
    RANDOM_FOREST))
        ]

        for option, filter in process_options:
            if algorithm == option:
                prediction_pipe = process_pipe >= filter ==
    self.new_pipe()
                break

        return prediction_pipe
```

8. No arquivo **pipeline_start.py**, adicionar as opções (adaptadas a opção corrente) na variável **process_options**.

```
    process_options = [
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.NOTHING,
    UnbiasPostProcAlgorithms.NOTHING),
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.NOTHING,
    UnbiasPostProcAlgorithms.EQUALIZED_ODDS),
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.NOTHING,
         UnbiasPostProcAlgorithms.CALIBRATED_EQUALIZED_ODDS),
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.NOTHING,
         UnbiasPostProcAlgorithms.REJECT_OPTION_CLASSIFICATION
    ),
```

```
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.REWEIGHING, UnbiasPostProcAlgorithms.NOTHING),
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.DISPARATE_IMPACT_REMOVER,
         UnbiasPostProcAlgorithms.NOTHING),
        # (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.OPTIMIZED_PREPROCESSING, UnbiasPostProcAlgorithms.NOTHING),
        (Algorithms.NOVA_OPCAO, UnbiasDataAlgorithms.LEARNING_FAIR_REPRESENTATIONS,
         UnbiasPostProcAlgorithms.NOTHING)
    ]
```

9. Adicionar uma nova condição com a opção adicionada

3.2.2 Interface (Backend)

1. No arquivo **repo/pipeline.py**, colocar as novas opções colocadas no item anterior no método **get_inproc_algorithm**.

```
    def get_inproc_algorithm(self, algorithm):
        indexes = {
            'Algorithms.LOGISTIC_REGRESSION': 1,
            'Algorithms.RANDOM_FOREST': 2,
            'Algorithms.GRADIENT_BOOST': 3,
            'Algorithms.SUPPORT_VECTOR_MACHINES': 4,
            'Algorithms.LINEAR_REGRESSION': 901,
            'Algorithms.DECISION_TREE': 902,
            'Algorithms.KERNEL_RIDGE': 903,
            'UnbiasInProcAlgorithms.PREJUDICE_REMOVER': 101,
            'UnbiasInProcAlgorithms.ADVERSARIAL_DEBIASING': 102,
            'UnbiasInProcAlgorithms.EXPONENTIATED_GRADIENT_REDUCTION': 103,
            'UnbiasInProcAlgorithms.RICH_SUBGROUP_FAIRNESS': 104,
            'UnbiasInProcAlgorithms.GRID_SEARCH_REDUCTION': 105,
            'UnbiasInProcAlgorithms.META_FAIR_CLASSIFIER': 106,
            'UnbiasInProcAlgorithms.ART_CLASSIFIER': 107
        }

        return next(filter(lambda a: a[0] == algorithm, indexes.items()))[1]
```

3.2.3 Interface (Frontend)

1. No arquivo **Manual-Pipeline-Menu.js**, adicionar a opção colocada na etapa de Pipeline no componente Select onde estão as outras opções de algoritmos.

```
<Select
  sx={{fontSize: '14px'}}
  value={trainAlgorithm}
```

```
  onChange={handleTrainAlgorithmChange}
  displayEmpty
  inputProps={{ 'aria-label': 'Without label' }}
>
  <MenuItem value={'Algorithms.LOGISTIC_REGRESSION'}>Logistic
    Regression</MenuItem>
  <MenuItem value={'Algorithms.RANDOM_FOREST'}>Random Forest</
    MenuItem>
  <MenuItem value={'Algorithms.GRADIENT_BOOST'}>Gradient Boost
    </MenuItem>
  <MenuItem value={'Algorithms.SUPPORT_VECTOR_MACHINES'}>
    Support Vector Machines</MenuItem>
</Select>
```

2. No arquivo **Planning-Menu.js**, adicionar as opções (adaptadas a opção corrente) na variável **validAlgorithms**.

```
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  NOTHING", "UnbiasPostProcAlgorithms.NOTHING"],
    labels: ["Nome da nova opcao", "Sem metodo", "Sem metodo"],
    selected: true
  },
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  NOTHING", "UnbiasPostProcAlgorithms.EQUALIZED_ODDS"],
    labels: ["Nome da nova opcao", "Sem metodo", "Equalized Odds"
  ],
    selected: true
  },
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  NOTHING", "UnbiasPostProcAlgorithms.CALIBRATED_EQUALIZED_ODDS"
  ],
    labels: ["Nome da nova opcao", "Sem metodo", "Calibrated
  Equalized Odds"],
    selected: true
  },
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  NOTHING", "UnbiasPostProcAlgorithms.
  REJECT_OPTION_CLASSIFICATION"],
    labels: ["Nome da nova opcao", "Sem metodo", "Reject Option
  Classification"],
    selected: true
  },
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  REWEIGHING", "UnbiasPostProcAlgorithms.NOTHING"],
    labels: ["Nome da nova opcao", "Reweighing", "Sem metodo"],
    selected: true
  },
  {
    options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
  DISPARATE_IMPACT_REMOVER", "UnbiasPostProcAlgorithms.NOTHING"],
    labels: ["Nome da nova opcao", "Disparate Impact Remover", "
  Sem metodo"],
```

```
      selected: true
    },
    {
      options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
    OPTIMIZED_PREPROCESSING", "UnbiasPostProcAlgorithms.NOTHING"],
      labels: ["Nome da nova opcao", "Optimized Preprocessing", "
    Sem metodo"],
      selected: true
    },
    {
      options: ["Algorithms.NOVA_OPCAO", "UnbiasDataAlgorithms.
    LEARNING_FAIR_REPRESENTATIONS", "UnbiasPostProcAlgorithms.
    NOTHING"],
      labels: ["Nome da nova opcao", "Learning Fair Representations
    ", "Sem metodo"],
      selected: true
    }
```