Recuperação e Preparação de dataset

Os notebooks covid19_e_aspectos_socioeconomicos-01.ipynb e covid19_e_aspectos_socioeconomicos-02.ipynb descrevem o processo de junção dos dados, além de tratamento dado a eles. A partir deles, chegamos ao dataset_02.csv, posto no mesmo repositório desses notebooks (https://github.com/juliogustavocosta/covid19_e_aspectos_socioeconomicos). Para mais detalhes a respeito do processo, recomendamos acessar esses notebooks e estudar o processo.

A origem dos dados recuperados e tratados nesses dois notebooks já está informada em seções anteriores. O maior desafio quanto a isso ficou por conta da junção dos conjuntos de dado de fontes distintas. Entretanto, foi desafio vencido e descrito no primeiro notebook.

O notebook covid19_e_aspectos_socioeconomicos_03.ipynb, por sua vez, apresenta tudo o que segue descrito daqui em diante. Ele faz uso do dataset_02.csv construído a partir dos dois notebooks anteriores. Discutiremos a seguir apenas os aspectos que julgamos mais relevantes.

Recuperando toda a discussão quanto ao tratamento do dataset para treinamento do modelo, destacamos apenas que aqui não empregamos qualquer técnica de balanceamento escrita especificamente esse fim. Confiamos no serviço da classe “StratifiedKFold” que tem por fim, também, garantir balanceamento entre as entradas da base de treinamento, mesmo em presença de desbalanceamento dos dados.

O Treinamento

Com o fim de facilitar a operação de investigação a respeito de qual classificador, e seu conjunto de hiperparâmetros, melhor se adequa a dar respostas à questão aqui posta, utilizamos um pipeline.

```
# Categrical features to pass down the categorical pipeline
categorical_features = X_train.select_dtypes("object").columns.to_list()

# Numerical features to pass down the numerical pipeline
numerical_features = X_train.select_dtypes("float64").columns.to_list()


# Defining the steps in the categorical pipeline
categorical_pipeline = Pipeline(steps = [('cat_selector', FeatureSelector(categorical_features)),
                                         ('cat_transformer', CategoricalTransformer()),
                                         ('cat_encoder','passthrough')
                                         #('cat_encoder',OneHotEncoder(sparse=False,drop="first"))
                                         ])
```

```
# Defining the steps in the numerical pipeline
numerical_pipeline = Pipeline(steps = [('num_selector', FeatureSelector(numerical_features)),
                                       ('num_transformer', NumericalTransformer())
                                       ])

# Combining numerical and categorical piepline into one full big pipeline horizontally
# using FeatureUnion
full_pipeline_preprocessing = FeatureUnion(transformer_list =
[('cat_pipeline', categorical_pipeline), ('num_pipeline', numerical_pipeline)])
```

O que destacamos no pipeline é, sobretudo, a facilidade de automação que ele oferece, além de nos dá a chance de separar os dados entre categóricos e numéricos, para facilitar eventuais transformações anteriores ao treinamento propriamente dito. Transformações como as de normalização da parte numérica da base. O trecho a seguir destaca a conclusão da configuração e execução do pipeline.

```
### Configuração e execução do Pipeline

seed = 15
num_folds = 10
# scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score)}
scoring = {'Accuracy': make_scorer(accuracy_score)}

# The full pipeline
pipe = Pipeline(steps = [('full_pipeline', full_pipeline_preprocessing),
                         ("fs", SelectKBest()),
                         ("classifier", DecisionTreeClassifier())])

# create a dictionary with the hyperparameters
search_space = [
               {"classifier": [DecisionTreeClassifier()],
                "classifier__criterion": ["gini", "entropy"],
                "classifier__splitter": ["best", "random"],
                "fs__k": [11, 13, 17, 19, 23],
                "fs__score_func": [mutual_info_classif],
                "full_pipeline__cat_pipeline__cat_encoder":
[OneHotEncoder(sparse=False,drop="first")],
                "full_pipeline__cat_pipeline__cat_transformer__new_features": [False],
                "full_pipeline__num_pipeline__num_transformer__model": [1,0]},
                {"classifier": [RandomForestClassifier()],
                "classifier__criterion": ["gini","entropy"],
                "classifier__n_estimators": [100, 200, 300, 500],
                "fs__k":[11, 13, 17, 19, 23],
                "fs__score_func": [mutual_info_classif],

"full_pipeline__cat_pipeline__cat_encoder":[OneHotEncoder(sparse=False,drop="first")]}
]

# create grid search
grid = GridSearchCV(estimator=pipe,
                    param_grid=search_space,
                    cv=StratifiedKFold(n_splits=num_folds,random_state=seed,shuffle=True),
                    scoring=scoring,
```

```
                    return_train_score=True,
                    n_jobs=-1,
                    refit="Accuracy",
                    verbose = 1)

### fit grid search
all_models = grid.fit(X_train, y_train)
```

O trecho de código acima conclui a apresentação do pipeline: o espaço de busca definido para identificar qual dos classificadores será mais adequado (e seus hiperparâmetros) para ser selecionado como modelo. Destacamos também a instância de ”GridSearchCV” que recebe como parâmetro o espaço de busca e a instância da estratégia de particionamento, “StratifiedKFold”. Por ela a base é dividida em 10 partições de treinamento e, como já dito, nessa partição, busca garantir o balanceamento entre distintas categorias da entrada, mesmo na presença de desbalanceamento (que é o caso apresentado anteriormente).

De resto, foram usados os classificadores: “DecisionTreeClassifier” e o “RandomForestClassifier”, cada um conforme os hiperparâmetros especificados no pipeline.

Resultados

O resultado do treinamento revelou a seguinte acurácia, que implicou a escolha do classificador ”DecisionTreeClassifier”:

```
print("Melhor resultado: %f usando %s" % (all_models.best_score_,all_models.best_params_))
```

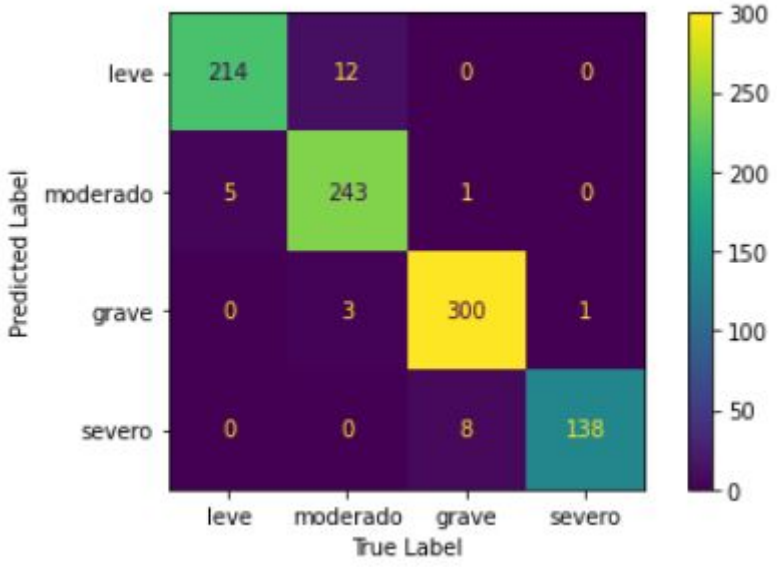
```
Melhor resultado: 0.976473 usando {'classifier': DecisionTreeClassifier(), 'classifier__criterion':
```

Quanto à acurácia do modelo agora aplicado ao Y de teste:

```
print(accuracy_score(y_test, predict))
```

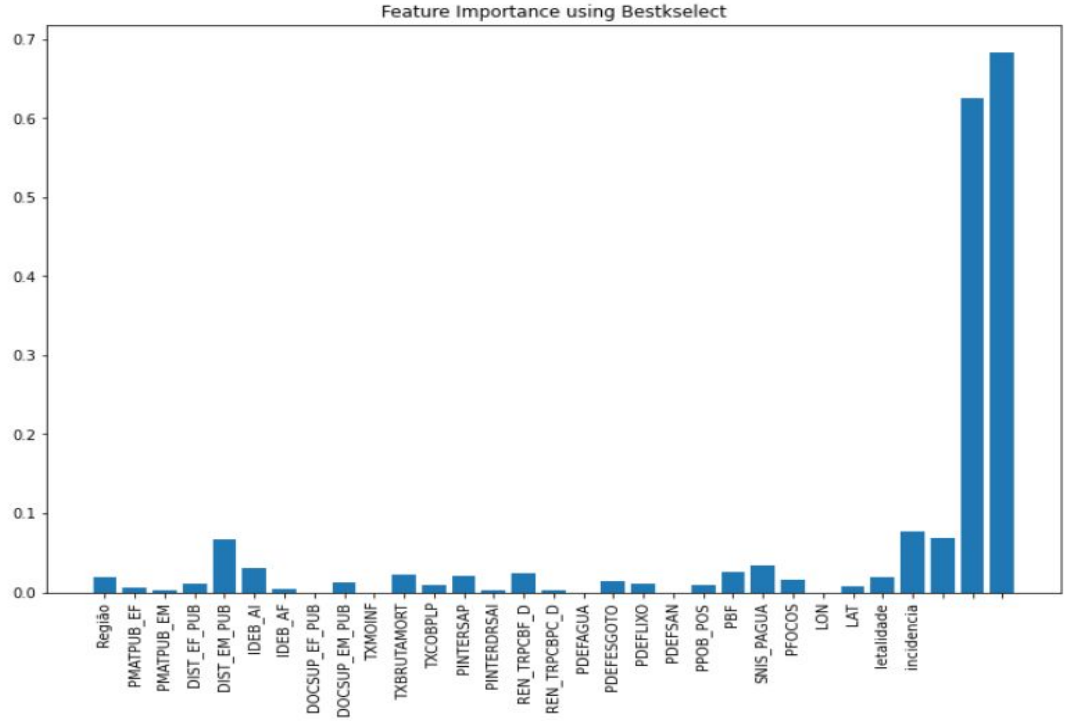
```
0.9675675675675676
```

Quanto à matriz de confusão:



Destacamos apenas a identificação de dificuldade quanto a classificar os municípios em relação às categorias “leve” e “moderado”.

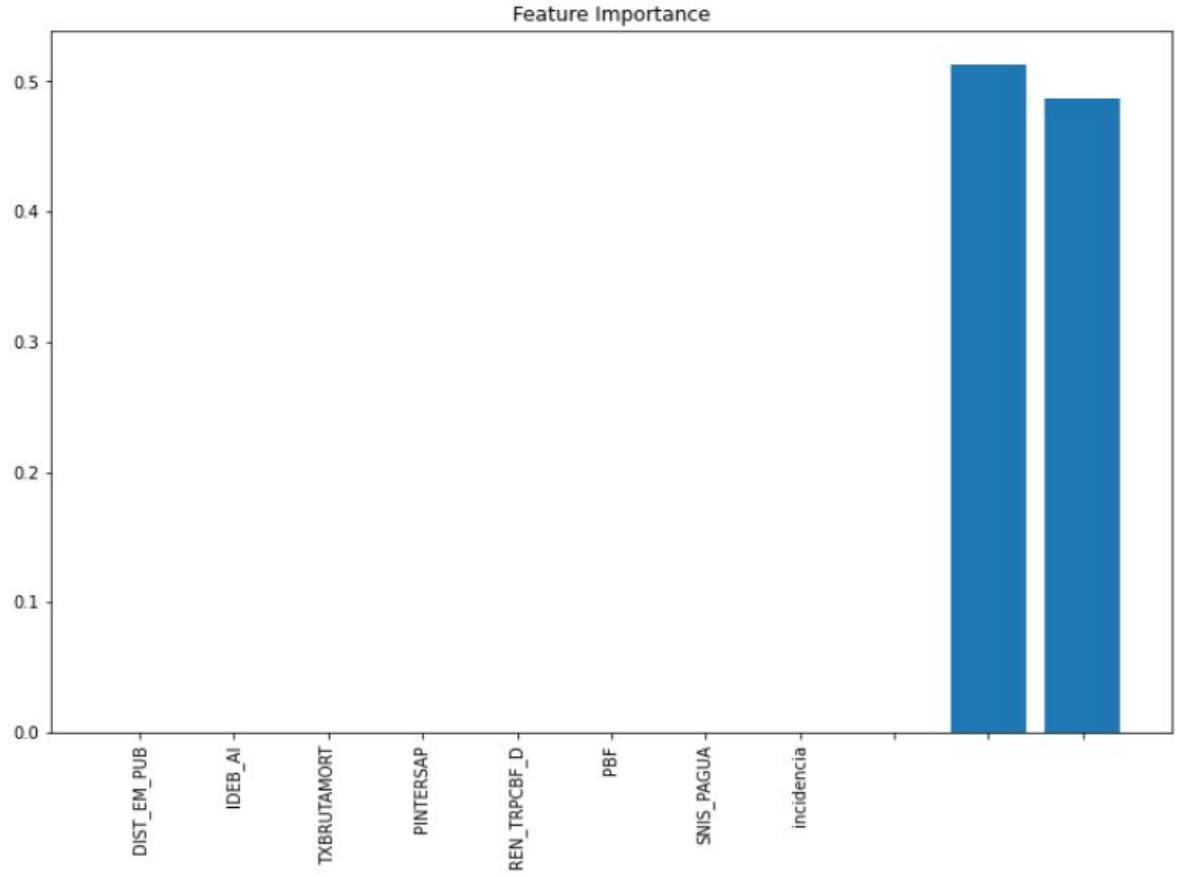
Além dessa imagem, mostramos também como as variáveis do espaço de dados se comportam em relação à categorização dos municípios.



A última imagem destaca as 11 variáveis (este é o número retornado pelo hiperparâmetro 'fs__k' que indica quantas são as melhores variáveis selecionadas

pelo modelo) empregadas na geração da árvore de decisão para a categorização dos municípios conforme a variável alvo, “Y”.

Ao final, percebe-se que as variáveis de ordem socioeconômica não são decisivas para determinar a categorização dos municípios conforme os critérios usados especificamente nesse trabalho.



Ressaltamos, contudo, que este é um trabalho cujo principal intuito está no aprendizado de técnicas de Aprendizado de Máquina, e não em responder de forma científica predições de categorias dos municípios, conforme critérios criados mais arbitrariamente voltados para o fim do aprendizado da ferramenta.