Análise Comparativa:
Lung Cancer

Bárbara da S. Oliveira, Carlos A. de S. Monteiro,
Carlos A. M. de Pinho, Hugo S. Sousa,
Larissa A. Barbosa, Luiz H. M. de Souza,
Matheus A. Constancio e Rafael G. da Silva

7 de novembro de 2022

Data Sailers

Conteúdo

Introdução

Análise Comparativa

- O objetivo dessa etapa é comparar modelos para encontrar um que possa ser melhor utilizado para a detecção de câncer pulmonar
- Para isso, precisamos antes realizar a preparação e o pré-processamento dos dados

Dados Originais

Variável	Tipo
GENDER	Nominal
AGE	Discreta
SMOKING	Nominal
YELLOW_FINGERS	Nominal
ANXIETY	Nominal
PEER_PRESSURE	Nominal
CHRONIC DISEASE	Nominal
FATIGUE	Nominal
ALLERGY	Nominal
WHEEZING	Nominal
ALCOHOL CONSUMING	Nominal
COUGHING	Nominal
SHORTNESS OF BREATH	Nominal
SWALLOWING DIFFICULTY	Nominal
CHEST PAIN	Nominal
LUNG_CANCER	Nominal

Tabela 1: Variáveis e seus tipos

Pré-processamento

Tratamento de Dados

```
# Criar Pipeline para cada tipo de variavel

# Para variaveis nominais
nominal_preprocessor = Pipeline([
    # Dados discrepantes
    ('missing', SimpleImputer(strategy='most_frequent')),
    # Dados faltantes
    ('encoder', OneHotEncoder(sparse=False)),
    # Normalizacao
    ('normalization', MinMaxScaler()),
])

# Para variaveis discretas
discrete_preprocessor = Pipeline([
    ('missing', SimpleImputer(strategy='mean')),
    ('normalization', MinMaxScaler())
])
```

Tratamento de Dados

- ***SimpleImputer***: Substitui qualquer valor faltante com um valor estático (média, moda, mediana), neste caso, a média
- ***OneHotEncoder***: Encoda as características dos nossos dados em um *array* numérico, onde cada valor representará um dos dados
- ***MinMaxScaler***: Normaliza os dados escalando eles para ficarem entre uma determinada faixa de valores

Tratamento de Dados

```
# Agora podemos utilizar os pre processadores
# Criados acima para transformar os dados
preprocessing = ColumnTransformer([
    ("discrete", discrete_preprocessor, discrete_columns),
    ("nominal", nominal_preprocessor, nominal_columns)
])
```

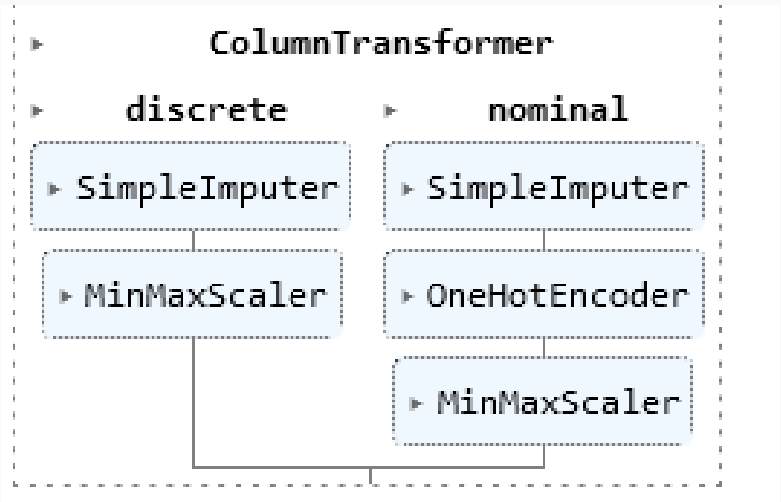

Figura 1: Transformação das variáveis

Metodologia e Experimento

Escolha do Modelo

Iremos analisar quatro modelos de aprendizado de máquina para solucionar nosso problema:

- ***Logistic Regression*** **(LR)**
- ***K-Nearest-Neighbors*** **(KNN)**
- ***Support Vector Machine*** **(SVM)**
- ***Naive Baiyes*** **(NB)**

Para encontrarmos, dentro desses modelos, a melhor configuração viável para o nosso problema, utilizaremos a classe *GridSearch* do *SickitLearn*.

Configuração do Experimento

```
# Nome do modelo, Chamada do metodo e Parametros de Teste
models = [
    ("LR", LogisticRegression(solver='saga', max_iter=1000),
    {"penalty": ['none', 'l1', 'l2']} ),

    ("KNN", KNeighborsClassifier(metric='euclidean'),
    {"n_neighbors": np.arange(1, 31, 2), 'weights': ["uniform", "
     distance"]}),

    ("SVM", SVC(max_iter=10000),
    {'C':[1, 10, 100, 1000],'gamma':[1, 0.1, 0.001, 0.0001], '
     kernel' ['linear','rbf']}),

    ("NB", BernoulliNB(), {"alpha": [1e-3, 0.5, 1]}),
]
```

Critérios de Avaliação

Os modelos serão comparados através dos seguintes parâmetros:

- ***accuracy***: Proporção entre os dados que foram corretamente previstos (como positivos ou negativos) com o total de dados observados
- ***precision***: Proporção entre dados corretamente previstos como positivos e o total de observações positivas
- ***recall***: Proporção entre dados corretamente previstos como positivos com o total de observações. Em outras palavras, esse parâmetro ajuda a identificar a sensibilidade do nosso modelo
- ***f1***: Média ponderada entre *precision* e *recall*, portanto levando em conta tanto falsos positivos quanto falsos negativos

Método de Validação

Iremos separar o conjunto de dados em dois conjuntos: **teste** e **treino** para realizar a validação cruzada.

```
X = df.drop(columns=[target_column], axis=1)
y = (
    df[[target_column]]
    .replace({"YES": 1, "NO": 0})
    .to_numpy()
    .ravel()
)

#Separa em conjuntos de teste e treino
cv = ShuffleSplit(n_splits=30, train_size=0.8, random_state=42)
```

Teste do Modelo

- Para a verificação do modelo, foi utilizado o *Cross Validation* ou Validação Cruzada, onde há uma divisão entre os dados para treinamento
- Mais especificamente, foi utilizado o método de Monte Carlo, onde a divisão feita entre os dados é arbitraria, nesse caso, **80%** foram utilizados para o treinamento do modelo
- Além disso, fazemos essa separação 30 vezes, para que o algoritmo nunca realize o treino nos mesmo valores previamente selecionados

MONTE CARLO

Figura 2: Representação do Método de Monte Carlo

Método de Validação

```
results = {}
for model_name, model, model_params in models:
    model_gs = GridSearchCV(model, model_params, scoring='
    accuracy')
    approach = Pipeline([
        ("preprocessing", preprocessing),
        ("model", model_gs)
    ])
    model_results = cross_validate(
        approach,
        X=X,
        y=y,
        # Criterios de avaliacao
        scoring=['accuracy', 'f1', 'precision', 'recall'],
        cv=cv,
        n_jobs=-1,
        return_train_score=False
    )
```

Conclusão

Resultados

score	KNN	LR	NB	SVM
fit_time	0.704 ± 0.141	0.351 ± 0.054	0.053 ± 0.008	1.314 ± 1.039
score_time	0.014 ± 0.007	0.011 ± 0.004	0.013 ± 0.007	0.014 ± 0.009
test_accuracy	0.904 ± 0.035	0.930 ± 0.034	0.912 ± 0.033	0.923 ± 0.033
test_f1	0.946 ± 0.021	0.960 ± 0.020	0.949 ± 0.020	0.956 ± 0.020
test_precision	0.933 ± 0.036	0.948 ± 0.034	0.962 ± 0.033	0.948 ± 0.033
test_recall	0.960 ± 0.021	0.974 ± 0.018	0.937 ± 0.023	0.966 ± 0.019

Tabela 2: Resultados obtidos

Persistência do modelo

Com isso, definimos que o melhor modelo é o de **Regressão Logística**, portanto podemos obter os melhores parâmetros desse modelo e salvar esse modelo em disco para utilizar na próxima fase da análise.

```
#Obtem o modelo e os parametros ganhadores
model_name, model, model_params  = [foo for foo in models if foo
    [0] == winner][0]

#Obtem o melhor modelo denovo pelo GridSearchCV
model_gs = GridSearchCV(model, model_params, scoring='accuracy')
approach = Pipeline([
    ("preprocessing", preprocessing),
    ("model", model_gs)
])
approach.fit(X, y) #Seleciona o approach
```

PERSISTÊNCIA DO MODELO

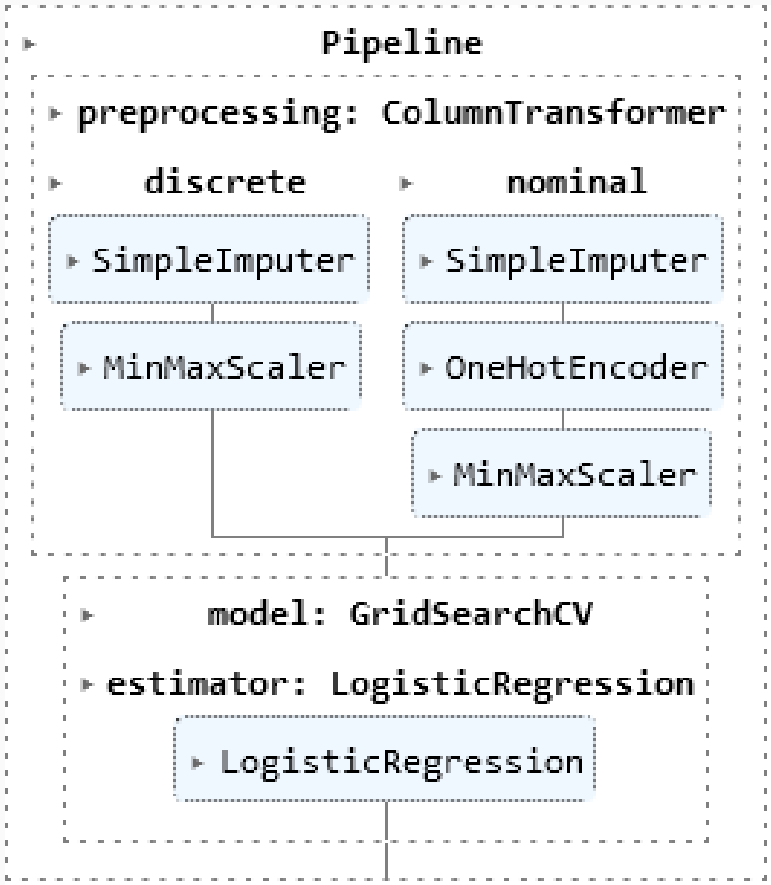

Figura 3: Modelo Escolhido

Próximos Passos

Com o modelo selecionado e treinado, agora podemos:

- Construir um formulário para obter dados de usuários
- Aplicar o modelo escolhido nesses dados para prever a possibilidade de câncer pulmonar
- Realizar o *deploy* desse formulário online (utilizando o ***Streamlit*** e ***StreamlitShare***)