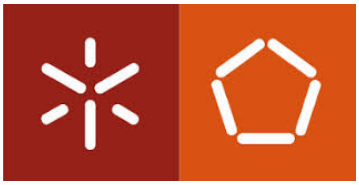


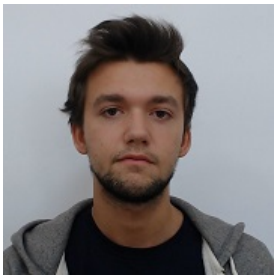
Universidade do Minho
Mestrado Integrado em Engenharia Informática

Dados e Aprendizagem Automática

Trabalho de Grupo
Grupo 20

Braga
3 de janeiro de 2022


António Santos
(PG47031)


Jorge Vieira
(PG47349)


Manuel Moreira
(PG47439)


Sara Dias
(PG47667)

Resumo

O trabalho desenvolvido pretende mostrar a conceção e desenvolvimento de um projeto de *Machine Learning* utilizando os modelos de aprendizagem abordados ao longo do semestre, na UC de Dados e Aprendizagem Automática.

O objetivo deste trabalho passa por, entre outros, desenvolver modelos de *Machine Learning* capazes de prever o fluxo de tráfego rodoviário, numa determinada hora, na referida cidade. Para além deste, procedemos a fazer o mesmo para um *dataset* de previsão de notas académicas.

Ao longo deste relatório, vamos proceder à descrição do trabalho realizado, bem como das metodologias que seguimos. Assim, vamos descrever detalhadamente ambos os *datasets* utilizados, assim como vamos apresentar as justificações para as nossas escolhas dos modelos que melhor precisão tenham.

Em suma, consideramos que o trabalho desenvolvido cumpre os objetivos propostos pela equipa docente e que este possibilitou uma melhor assimilação de conhecimentos teóricos e práticos lecionados na UC de Dados e Aprendizagem Automática, bem como dos vários tipos de aprendizagem.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este relatório é relativo ao trabalho prático da UC de Dados e Aprendizagem Automática e tem como principal objetivo a exploração de dados, a sua manipulação e utilização de forma a desenvolver modelos de aprendizagem automática capazes de resolver problemas da vida real.

Neste trabalho era pretendido o desenvolvimento de um modelo de aprendizagem automática capaz de prever o trânsito de uma rua de acordo com informações meteorológicas, e um modelo capaz de resolver um problema escolhido pelo grupo.

Capítulo 2

Metodologia

Primeiramente, analisamos extensivamente os dados presentes nos *datasets* de forma a entender as alterações necessárias para a sua utilização otimizada no desenvolvimento do modelo. De seguida, estudamos os resultados pretendidos nos *datasets* com o objetivo de concluir o tipo de modelo a escolher.

Com o *dataset* estudado, procedemos à manipulação dos dados de acordo com as necessidades dos modelos a testar.

Após termos os dados todos preparados, estes foram aplicados a diversos modelos de aprendizagem automática, de onde se retiraram as suas previsões e se calculou o desempenho de cada um, sendo escolhido o algoritmo que apresentou melhor desempenho.

Com o algoritmo escolhido, foram testados e recolhidos os melhores parâmetros de execução para obter o desempenho óptimo da solução.

Capítulo 3

Descrição e Exploração dos *datasets*

3.1 *Dataset* 'Fluxo do tráfego de carros'

Um dos *datasets* presentes neste projeto é um *dataset* com dados referentes ao trânsito de uma rua de acordo com informações meteorológicas, fornecido pela equipa docente.

Este *dataset* é composto por 14 colunas, que são as seguintes:

Coluna	**Descrição**	**Categoria**
city_name	nome da cidade em causa	Categórico
record_date	o *timestamp* associado ao registo	Numérico
average_speed_diff	a diferença de velocidade entre carros **(valor a prever)**	Categórico
average_free_flow_speed	valor médio da velocidade máxima que os carros podem atingir	Numérico
average_time_diff	valor médio da diferença do tempo que se demora a percorrer um determinado conjunto de ruas.	Numérico
average_free_flow_time	o valor médio do tempo que demora a percorrer um determinado conjunto de ruas quando não há trânsito	Numérico
luminosity	o nível de luminosidade que se verificava na cidade do Porto	Categórico
average_temperature	o valor médio da temperatura para o record_date na cidade do Porto	Numérico
average_atmosp_pressure	o valor médio da pressão atmosférica para o record_date	Numérico
average_humidity	o valor médio da humidade para o record_date	Numérico
average_wind_speed	o valor médio da velocidade do vento para o record_date	Numérico
average_cloudiness	o valor médio da percentagem de nuvens para o record_date	Categórico
average_precipitation	o valor médio de precipitação para o record_date	Numérico
average_rain	avaliação qualitativa da precipitação para o record_date	Categórico

Após a análise do *dataset* chegamos às seguintes conclusões:

1. O *dataset* continha dados irrelevantes para o desenvolvimento do modelo.
2. O *dataset* continha elementos com valores nulos.
3. Os dados discretos presentes no *dataset* estavam representados como *Strings*.
4. Os valores numéricos não estavam equilibrados.

Sendo deparados com estes factos, concluímos que seria necessário proceder com as devidas alterações e preparação do *dataset* de forma a resolver os problemas presentes no mesmo.

3.1.1 Preparação dos dados

Começamos por remover as colunas *city_name*, *average_precipitation* e *record_date*, visto que, numa inspeção inicial, estas colunas não apresentam nenhum dado relevante no desenvolvimento do modelo, isto é, não influenciam de certa forma o trânsito esperado.

```
df = df.drop(['city_name', 'AVERAGE_PRECIPITATION','record_date'], axis=1)
df_final = df_final.drop(['city_name',
    'AVERAGE_PRECIPITATION','record_date'], axis=1)
```

Tal como referido, alguns atributos do *dataset* continham valores nulos, pelo que foi necessário que adotássemos alguma técnica de preenchimento de valores em falta. Após testarmos vários métodos, concluímos que o que apresentou melhor desempenho no modelo foi a substituição de todos os valores nulos por 0.

```
#Ver valores nulos
plt.figure()
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')

#Converter os valores nulos para 0
df = df.fillna(0)
df_final = df_final.fillna(0)
```

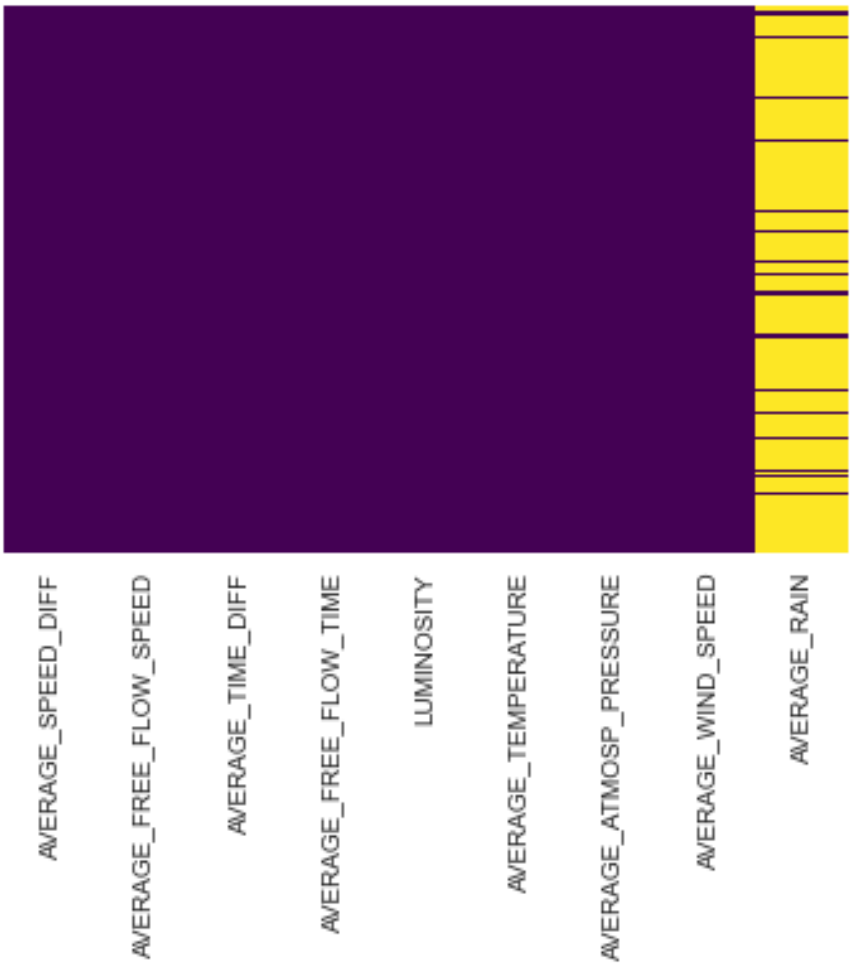


Figura 3.1: Valores nulos no *dataset*

Com o apoio do ***seaborn*** concluimos que o *dataset* não se encontrava balanceado, mas felizmente este problema foi resolvido facilmente ao adicionar um parâmetro aos algoritmos de aprendizagem. Este paramêtro faz *oversampling* implicitamente na execução do mesmo.

```
sns.set_style('whitegrid')
sns.countplot(x='AVERAGE_SPEED_DIFF',data=df)
```

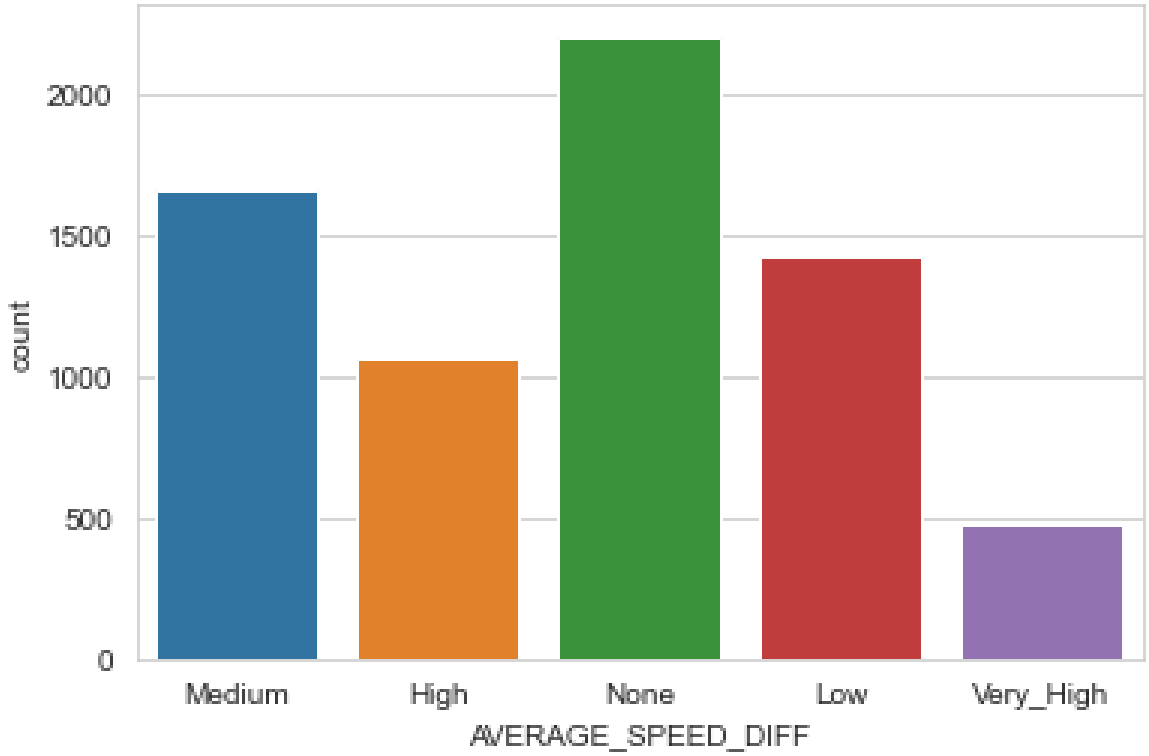


Figura 3.2: *Countplot* de trânsito

Quanto ao problema da representação dos dados discretos em *strings* e a sua incompatibilidade na modelação da solução, isto é, o modelo não reconhecer dados categóricos, procedemos à uilização da técnica ***Label Encoding*** de forma a convertê-los para dados numéricos. Para isso, recorremos à severidade da categoria em questão como critério decisivo do valor a atribuir-lhe, tendo em conta alguma subjetividade em certos termos mais ambíguos, bem como a cobertura de todos os valores, mesmo considerando erros ortográficos ou sinónimos.

```
#Converter os valores qualitativos para quantitativos
speed_diff = {'None': 0, 'Low': 1, 'Medium': 2, 'High': 3, 'Very_High': 4}
luminosity = {'DARK': 2, 'LOW_LIGHT': 1, 'LIGHT': 0}
cloudiness = {'céu claro': 0, 'céu limpo': 0, 'nuvens dispersas': 1, 'céu
    pouco nublado': 1, 'algumas nuvens': 1, 'nuvens quebrados': 2, 'nuvens
    quebradas': 2, 'tempo nublado': 3, 'nublado': 3}
rain = {'chuvisco fraco': 1, 'chuva fraca': 1, 'chuva leve': 1, 'aguaceiros
    fracos': 1, 'chuvisco e chuva fraca': 1, 'chuva': 2, 'chuva moderada':
    2, 'aguaceiros': 2, 'trovoada com chuva leve': 2, 'chuva de intensidade
    pesada': 3, 'chuva de intensidade pesado': 3, 'chuva forte': 3,
    'trovoada com chuva': 3}

df = df.replace({'AVERAGE_SPEED_DIFF': speed_diff, 'LUMINOSITY':
    luminosity, 'AVERAGE_RAIN': rain})
df_final = df_final.replace({'LUMINOSITY': luminosity,
    'AVERAGE_CLOUDINESS': cloudiness, 'AVERAGE_RAIN': rain})
```

Já com o *dataset* formatado corretamente, desenhamos uma matriz de correlação de forma a identificar colunas que não tenham impacto na previsão de resultados.

```
corr_matrix = df.corr()
f, ax = plt.subplots(figsize=(8,6))
sns.heatmap(corr_matrix,vmin=-1, vmax=1, square=True, annot=True)
```

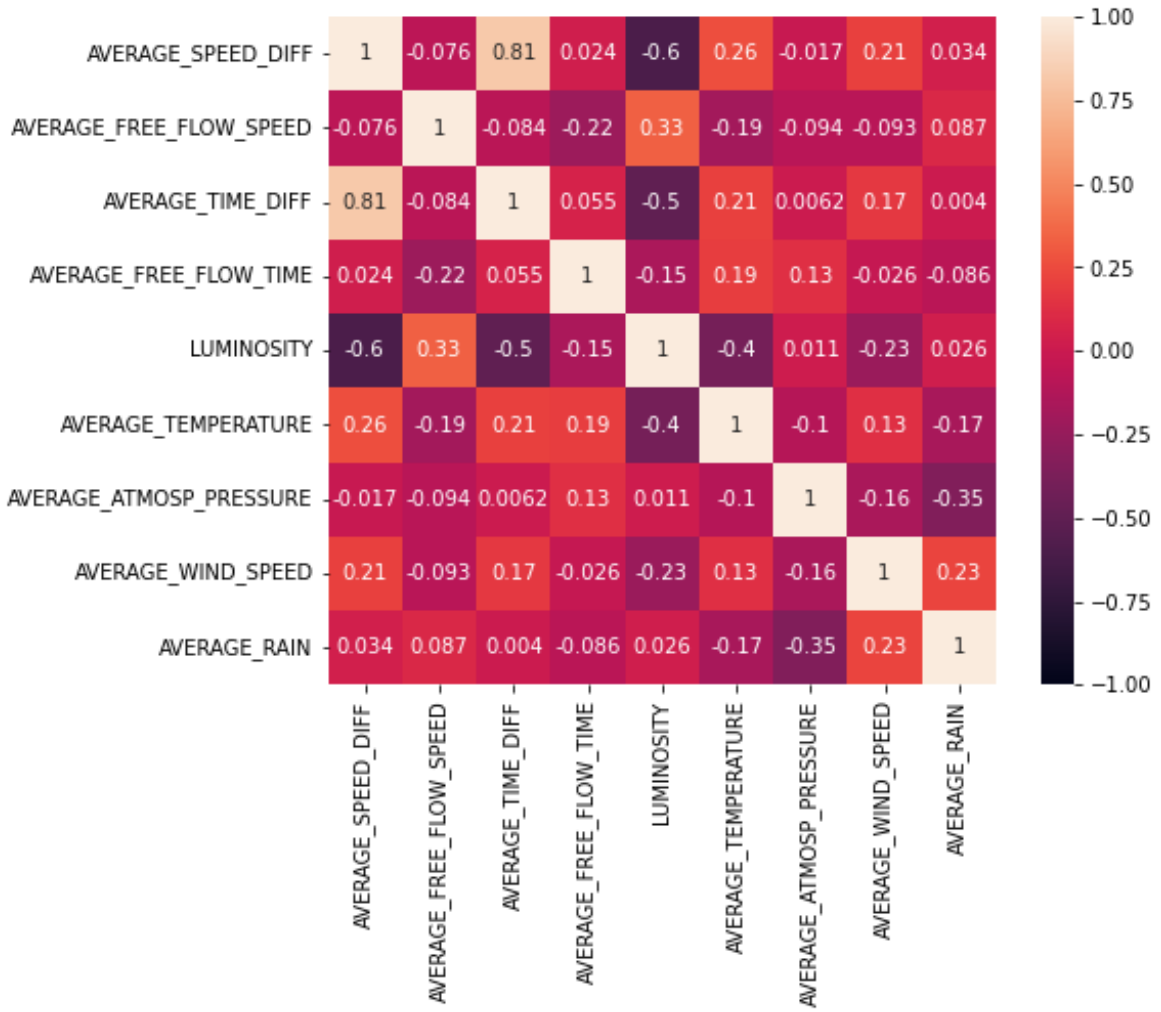


Figura 3.3: Matriz de Correlação

Após o estudo da matriz, pudemos concluir que a coluna *average_humidity* e *average_cloudiness* poderiam ser removidas sem afetar o modelo.

```
df = df.drop(['AVERAGE_HUMIDITY','AVERAGE_CLOUDINESS'], axis=1)
df_final = df_final.drop(['AVERAGE_HUMIDITY','AVERAGE_CLOUDINESS'], axis=1)
```

De forma a podermos calcular o desempenho dos modelos, o *dataset* foi dividido em 2 partes: uma para treino e uma para teste.

```
x = df.drop(['AVERAGE_SPEED_DIFF'],axis=1)
y = df['AVERAGE_SPEED_DIFF'].to_frame()
```

```
x_train, x_test, y_train, y_test =
    train_test_split(x,y,test_size=0.25,random_state=2021)
```

3.1.2 Descrição dos modelos desenvolvidos

Primeiramente, começamos por selecionar alguns modelos oportunos para o problema fornecido. Como o problema fornecido pela equipa docente pede a previsão de valores categóricos (discretos) decidimos então estudar os algoritmos de classificação para utilizar como solução.

Assim, procedemos a comparar os algoritmos de **Árvore de Decisão**, ***Support Vector Machine***, ***Random Forest Classifier*** e **Regressão Logística** sem nenhum tipo de alterações.

```
#Decision Tree
timer=datetime.datetime.now()
clf = DecisionTreeRegressor(random_state=2)
clf.fit(x_train,y_train)
predictions = clf.predict(x_test)
print("Decision tree accuracy: " + str(accuracy_score(y_test,predictions)))
print("Decision tree time to run: " +
    str((datetime.datetime.now()-timer).total_seconds()*1000))

#Logistic Regression
timer=datetime.datetime.now()
logmod = LogisticRegression(max_iter=2000)
logmod.fit(x_train,np.ravel(y_train))
predictions = logmod.predict(x_test)
print("Logistic Regression accuracy: " +
    str(accuracy_score(y_test,predictions)))
print("Logistic Regression time to run: " +
    str((datetime.datetime.now()-timer).total_seconds()*1000))


#Random Forest Classifier
timer=datetime.datetime.now()
rfc = RandomForestClassifier()
rfc.fit(x_train,np.ravel(y_train))
predictions = rfc.predict(x_test)
print("RFC accuracy: " + str(accuracy_score(y_test,predictions)))
print("RFC time to run: " +
    str((datetime.datetime.now()-timer).total_seconds()*1000))

#Support vector machine
timer=datetime.datetime.now()
svm = SVC(random_state=2021)
svm.fit(x_train,y_train)
predictions = svm.predict(x_test)
print("SVM accuracy: " + str(accuracy_score(y_test,predictions)))
```

```
print("SVM time to run: " +
    str((datetime.datetime.now()-timer).total_seconds()*1000))
```

Ao executar o excerto de código apresentado, conseguimos desenhar a seguinte tabela:

Nome	**Precisão (%)**	**Tempo de execução (ms)**
Árvore de Decisão	70.5%	28
Support Vector Machine	56.85%	2605
Random Forest Classifier	77.7%	1173
Regressão Logística	76.57%	3764

Como é possível verificar, ***Random Forest Classifier*** parece ser a melhor opção (sem ajustes) em termos de precisão, porém o seu tempo de execução pode ser otimizado.

De forma a otimizar o desempenho, poderemos criar uma *pipeline* que torne os valores mais uniformes, utilizando o *StandardScaler*.

```
timer = datetime.datetime.now()
pipe = make_pipeline(StandardScaler(), RandomTreeClassifier())
pipe.fit(x,np.ravel(y))
predictions = pipe.predict(x)
print("RTC accuracy: " + str(accuracy_score(y,predictions)))
print("RTCtime to run: " +
    str((datetime.datetime.now()-timer).total_seconds()*1000))
```

Isto dá o seguinte resultado:

Nome	**Precisão (%)**	**Tempo de execução (ms)**
StandardScaler+Random Forest Classifier	78.3%	889

Como conseguimos ver, a utilização do *StandardScaler* melhora a precisão do modelo, tal como o tempo de execução, e será o algoritmo utilizado daqui para a frente.

Para otimizar ainda mais o modelo, foi utilizado *gridSearchCV* para encontrar os parâmetros de *tuning* que resultam numa maior precisão.

```
param_grid = [
    {'classifier' : [RandomForestClassifier()],
    'classifier__n_estimators': [200, 500],
    'classifier__max_features': ['auto', 'sqrt', 'log2'],
    'classifier__max_depth' : [4,5,6,7,8],
    'classifier__criterion' :['gini', 'entropy']},
    {'scaler' : [StandardScaler()]}
]

search = GridSearchCV(estimator=pipe, param_grid=param_grid, cv= 5)
search.fit(x_train, np.ravel(y_train))
print(search.best_params_)
```

Isto dá-nos os seguintes parâmetros:

1. 'criterion': 'gini'
2. 'max_depth': 8
3. 'max_features': 'log2'
4. 'n_estimators': 200

Adicionando o parâmetro que lida com dataset não balanceados e executando o algoritmo com os parâmetros afinados, dá-nos o seguinte resultado:

Nome	**Precisão (%)**	**Tempo de execução (ms)**
StandardScaler+RFC (Afinado)	78.8%	1402

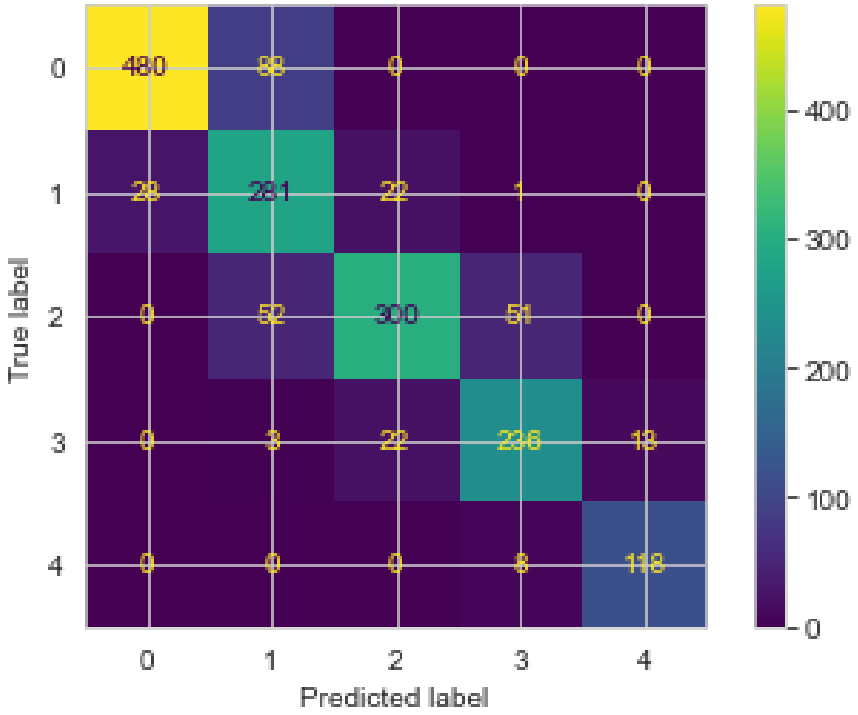


Figura 3.4: Matriz de Confusão

3.2 *Dataset* 'Performance de alunos nos exames'

O outro *dataset* presente neste projeto é um *dataset* com dados referentes à performance de cada aluno em diferentes exames, atendendo ao seu *background*.

Este *dataset* é composto por 8 colunas, que são as seguintes:

Coluna	**Descrição**	**Categoria**
gender	género do aluno em causa	Categórico
race/ethnicity	raça ou etnia do aluno em causa	Categórico
parental level of education	habilitações literárias dos pais do aluno	Categórico
lunch	grau de escalão do aluno	Categórico
test preparation course	indica se o aluno fez o teste de preparação	Categórico
math score	nota do aluno a Matemática	Numérico
reading score	nota do aluno a Leitura	Numérico
writing score	nota do aluno a Escrita **(valor a prever)**	Numérico

Após a análise do *dataset*, chegamos à seguinte conclusão:

1. Os dados discretos presentes no *dataset* estavam representados como *Strings*.

Sendo deparados com estes factos concluímos que seria necessário alterar o *dataset* de forma a resolver os problemas presentes no mesmo.

3.2.1 Preparação dos dados

Para resolver a incompatibilidade das *strings* na modelação da solução foi utilizada a técnica ***Label Encoding*** usando a severidade da categoria como critério para o valor dado.

```
#Converter os valores qualitativos para quantitativos
gender = {'male': 0, 'female': 1}
#print(df['race/ethnicity'].unique())
race = {'group A': 0, 'group B': 1, 'group C': 2, 'group D': 3, 'group E':
    4}
#print(df['parental level of education'].unique())
parental_education = {'high school': 0, 'some high school': 0,
    'associate\'s degree': 1, 'some college': 2, 'bachelor\'s degree': 2,
    'master\'s degree': 3}
#print(df['lunch'].unique())
lunch = {'free/reduced': 0, 'standard': 1}
#print(df['test preparation course'].unique())
test_preparation = {'none': 0, 'completed': 1}

df = df.replace({'gender': gender, 'race/ethnicity': race, 'parental level
    of education': parental_education, 'lunch': lunch, 'test preparation
    course': test_preparation})
```

Com o *dataset* formatado corretamente, desenhamos uma matriz de correlação de forma a identificar colunas que não tenham impacto na previsão de resultados.

```
corr_matrix = df.corr()
f, ax = plt.subplots(figsize=(8,6))
sns.heatmap(corr_matrix,vmin=-1, vmax=1, square=True, annot=True)
```

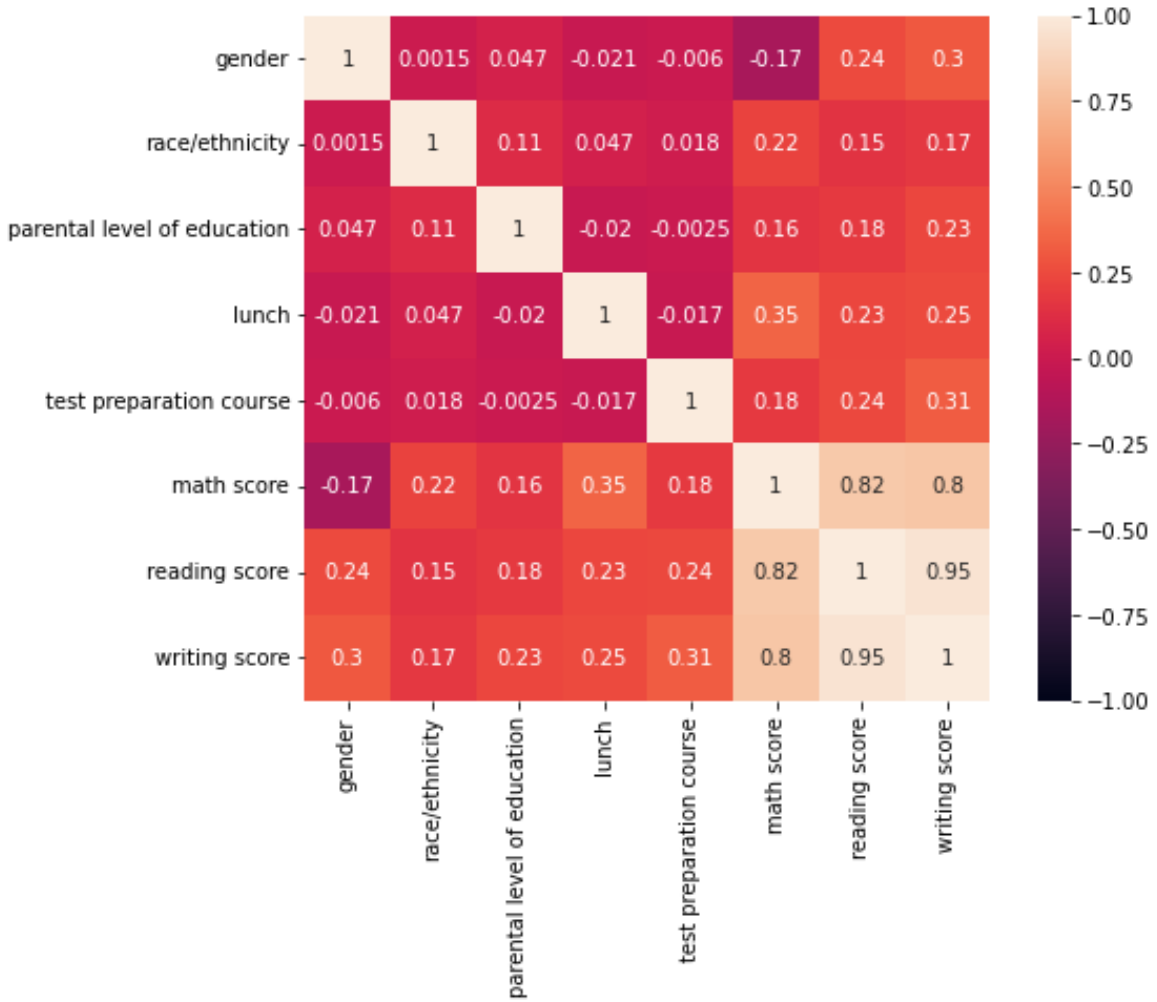


Figura 3.5: Matriz de Correlação

Após o estudo da matriz, chegamos à conclusão que o modelo nunca iria ter um bom desempenho se tentássemos prever as notas de todas as disciplinas, por isso acabamos por escolher o *writing score* como a coluna a prever.

Para podermos calcular o desempenho dos modelos, o *dataset* foi dividido em 2 partes (uma para treino e uma para teste). Neste *dataset* decidimos dividir os dados em metade: 50% para treino e 50% para teste.

```
split_value = 0.5
label = "writing score"
x = df.drop(label,axis=1)
y = df[label]
```

```
x_train, x_test, y_train, y_test =
    train_test_split(x,y,test_size=split_value,random_state=2021)
```

3.2.2 Descrição dos modelos desenvolvidos

Primeiramente começamos por selecionar alguns modelos oportunos para o problema fornecido. Desta vez o nosso objetivo é prever valores contínuos, e por isso selecionamos algoritmos de regressão para comparar.

Decidimos comparar entre os algoritmos de **Árvore de Decisão**, ***Regressão Ridge***, ***Regressão Linear*** e **Rede Elástica** sem nenhum tipo de alterações.

```
    #Ridge
    timer=datetime.datetime.now()
    pipe = Pipeline([('scaler' , StandardScaler()),('classifier' ,
        Ridge(alpha=1))])
    pipe.fit(x_train,y_train)

    predictions = pipe.predict(x_test)
    solution = y_test
    print(f"ridge regression x_test error for {label}:
        {str(mean_absolute_error(y_test,predictions))}")
    print(f"ridge regression time to run for {label}:
        {str((datetime.datetime.now()-timer).total_seconds()*1000)}")

    #Decision Tree Regressor
    timer=datetime.datetime.now()
    pipe = Pipeline([('scaler' , StandardScaler()),('classifier' ,
        DecisionTreeRegressor())])
    pipe.fit(x_train,y_train)

    predictions = pipe.predict(x_test)
    solution = y_test
    print(f"decision tree regression x_test error for {label}:
        {str(mean_absolute_error(y_test,predictions))}")
    print(f"decision tree regression time to run for {label}:
        {str((datetime.datetime.now()-timer).total_seconds()*1000)}")

    #Linear Regression
    timer=datetime.datetime.now()
    pipe = Pipeline([('scaler' , StandardScaler()),('classifier' ,
        LinearRegression())])
    pipe.fit(x_train,y_train)

    predictions = pipe.predict(x_test)
    solution = y_test
    print(f"linear regression x_test error for {label}:
        {str(mean_absolute_error(y_test,predictions))}")
    print(f"linear regression time to run for {label}:
```

```
    {str((datetime.datetime.now()-timer).total_seconds()*1000)}")

#Elastic Net
timer=datetime.datetime.now()
pipe = Pipeline([('scaler' , StandardScaler()),('classifier' ,
    ElasticNet())])
pipe.fit(x_train,y_train)

predictions = pipe.predict(x_test)
solution = y_test
print(f"elastic net regression x_test error for {label}:
    {str(mean_absolute_error(y_test,predictions))}")
print(f"elastic net time to run for {label}:
    {str((datetime.datetime.now()-timer).total_seconds()*1000)}")
```

Ao executar o excerto de código apresentado conseguimos desenhar a seguinte tabela:

Nome	**Erro médio (%)**	**Tempo de execução (ms)**
Ridge	2.87%	2.99
Decision Tree	4.20%	3.99
Linear Regression	2.86%	3.99
Elastic Net	3.65%	2.99

Como é possível ver, o ***Ridge Regression*** parece ser a melhor opção (sem ajustes) em termos de erro médio, porém a sua execução pode ser otimizada.

De forma a otimizar o desempenho, tentamos criar uma *pipeline* que torne os valores mais uniformes utilizando o *StandardScaler*.

Nome (com scale)	**Erro médio (%)**	**Tempo de execução (ms)**
Ridge	2.87%	4.99
Decision Tree	4.18%	5.98
Linear Regression	2.86%	4.986
Elastic Net	4.24%	3.99

Curiosamente e ao contrário do *dataset* anterior, o uso do *scaler* prejudica o desempenho do algoritmo, por isso não vai ser usado no desenvolvimento desta etapa.

Para otimizar mais o modelo, foi utilizado *gridSearchCV* para encontrar os parâmetros de *tuning* que resultam num menor erro médio.

```
param_grid = [
    {'classifier' : [Ridge()],
    'classifier__alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'classifier__tol': [0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000]},
]
```

```
search = GridSearchCV(estimator=pipe, param_grid=param_grid, cv=
    5,refit=True)
search.fit(x_train, np.ravel(y_train))
print(search.best_params_)
```

O *gridSearchCV* irá executar o algoritmo com todas as combinações possíveis de parâmetros dados no dicionário e irá devolver os parâmetros resultantes da execução que retornou um menor erro médio.

Isso dá-nos os seguintes parâmetros:

1. 'alpha': '0.001'
2. 'tol': 0.0001

Com os novos parâmetros, o desempenho do nosso modelo é o seguinte:

Nome	**Erro médio (%)**	**Tempo de execução (ms)**
Ridge (Afinado)	2.86%	1224

De notar que o tempo de execução tem em conta a procura dos parâmetros ótimos.

3.3 Análise crítica dos resultados

Em relação ao *dataset* fornecido pela equipa docente, conseguimos concluir que os atributos que mais influenciam os resultados são o **average_time_diff** e a **luminosity**, o que pode ser também comprovado pelos nossos conhecimentos e experiências do mundo real.

A precisão final do nosso modelo atingiu os 78.8%, mas infelizmente apenas foi possível atingir este valor depois da data limite da competição que nos colocaria na 25ª posição da tabela de resultados.

De notar que o modelo utilizado (*Random Forest Classifier*), tal como o nome indica, não tem uma precisão fixa, o valor atingido resultou do melhor valor entre 10 execuções do modelo.

Quanto ao *dataset* escolhido pelo grupo, tivemos de adotar uma tática diferente à anterior, devido à baixa quantidade de informação útil presente. Desta forma, foi necessário focarmo-nos mais na escolha e afinação dos algoritmos e não tanto na preparação dos dados.

Pela matriz de correlação, é possivel salientar a relevância das notas dos exames em relação ao resto dos atributos, que demonstram uma baixa influência no resultado desejado.

Este problema pode ser tratado como um problema de classificação ou de regressão, e como tal, de forma a variar a solução, o grupo deciciu tratar o problema como um de regressão. Quando, por exemplo, o modelo prevê o valor 83.65, podemos dizer que este acha mais provável o valor ser 84 do que 83.

Como este problema foi considerado um de regressão, não faz sentido calcular a sua precisão na escolha dos resultados, portanto optamos por utilizar o erro médio como medida de desempenho. Um erro médio de 2.86 é um valor bastante aceitável, visto que, as notas variam entre 0 e 100. Logo, podemos concluir que o nosso modelo tem um desvio de 2.86% da realidade.

Capítulo 4

Conclusões

O objetivo deste projeto passa pela conceção e desenvolvimento de um projeto de *Machine Learning* utilizando os modelos de aprendizagem abordados ao longo do semestre nas aulas da Unidade Curricular, ajudando à consolidação de conhecimentos.

Ao longo da realização do projeto, escolhemos um segundo *dataset*, e através de uma análise extensiva dos dados de ambos os *datasets*, procedemos à manipulação destes de forma a estarem prontos para serem testados com os vários modelos. Após a preparação dos dados, procedemos ao estudo dos vários modelos que poderiam ser escolhidos e escolhemos os que melhor desempenho mostraram.

Concluímos que o trabalho realizado neste projeto foi de encontro com os objetivos definidos pela equipa docente e desta forma estamos confiantes no trabalho desenvolvido.