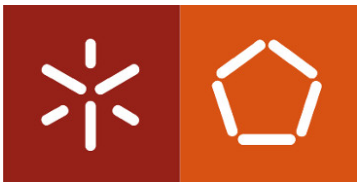


UMinho

Mestrado em Engenharia Informática

Dados e Aprendizagem Automática (2022/23)

Grupo 42

Gonçalo Braz (a93178)
Tiago Silva (a93277)
Simão Cunha (a93262)
Gonçalo Pereira (a93168)




Braga, 15 de janeiro de 2023

1. Introdução

Este relatório surge no âmbito do trabalho prático da Unidade Curricular de Dados e Aprendizagem Automática.

Iremos começar por uma breve apresentação dos objetivos que pretendemos atingir com este trabalho prático assim como a metodologia adotada ao longo do mesmo.

De seguida, iremos abordar dois casos de estudo: um proposto pela equipa docente e outro proposto pelo nosso grupo de trabalho. Em ambas as situações, iremos dar uma contextualização sobre o *dataset* em questão assim com uma exploração dos seus dados. Explicaremos, também, qual foi o tratamento efetuado a cada caso e quais foram os modelos de *machine learning* que foram aplicados, assim como uma análise crítica aos resultados desses mesmos modelos.

Terminaremos este relatório com uma secção de conclusões e eventuais alterações que melhorassem os resultados obtidos pelo grupo em ambos os casos de estudo deste trabalho prático.

2. Objetivos a atingir

O objetivo principal deste trabalho prático é elaborar um projeto de Machine Learning utilizando os modelos de aprendizagem abordados ao longo do semestre.

Neste projeto foram abordados dois casos de estudo:

- o primeiro contém dados referentes à quantidade e características dos incidentes rodoviários que ocorreram numa cidade portuguesa em 2021, mais propriamente a cidade de Guimarães. Aqui, fomos inseridos numa competição, criada pela equipa docente, com os outros grupos de trabalho para verificar quem contém o modelo com uma maior pontuação de *accuracy* na plataforma Kaggle, permitindo que a nossa equipa se esforçasse cada vez mais para atingir o primeiro lugar;
- já o segundo caso é inferir acerca da pontuação que será dada a um dado vinho, cujo *dataset* também foi obtido na mesma plataforma.

Numa 1ª fase, um dos objetivos da equipa foi o tratamento dos dados, procurando extrair conhecimento relevante no contexto de ambos os problemas, de forma a que os modelos desenvolvidos tivessem a melhor pontuação possível. Não obstante, para o primeiro *checkpoint*,

a equipa apenas desenvolveu os modelos de aprendizagem *decision tree*, *linear regression*, *logistic regression*, *redes neuronais* e *support vector machine*.

Já referente a uma 2ª fase (e a última), a equipa teve como objetivo desenvolver e otimizar modelos de *machine learning* para ambos os problemas estudados (incluindo os já desenvolvidos na primeira fase), aplicando ao longo do processo diversas métricas de avaliação, como por exemplo a *accuracy*.

3. Metodologia aplicada

Neste trabalho prático, adotamos uma metodologia de extração de conhecimento semelhante à utilizada durante as aulas práticas desta Unidade Curricular:

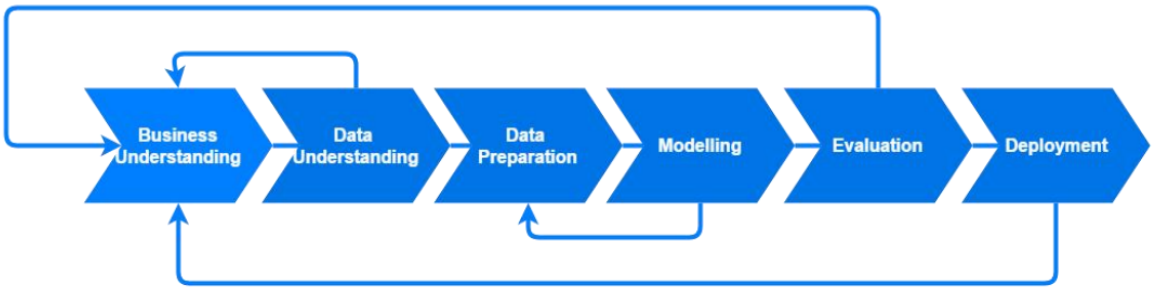


Figura 3.1: Metodologia de extração de conhecimento

Com isto, destacaremos as 5 primeiras fases - não iremos passar pela fase de *deployment* pois iremos aplicar vários modelos e verificar qual o que obtém melhor performance:

1. Compreensão do problema (*Business understanding*): perceber os objetivos do projeto e definir o problema de extração de conhecimento em ambos os casos de estudo;

2. Análise e exploração dos dados (*Data understanding*): analisar os dados de ambos os *datasets*, ou seja, verificar eventuais valores em falta, *outliers*, balanceamento dos dados, ... ;

3. Preparação dos dados (*Data preparation*): efetuar o tratamento dos dados: escolha de *features* relevantes para o problema, tratamento de *missing data* e adaptar todos os atributos com tipos de dados incompatíveis com o modelo de aprendizagem para tipos compatíveis, utilizando as técnicas correspondentes a cada um - por exemplo, transformar os dados para o tipo numérico;

4. Modelos de extração de conhecimento (*Modeling*): aplicação de vários modelos de aprendizagem com os dados previamente preparados nas etapas anteriores;

5. Avaliação dos modelos (*Evaluation*): de forma a sabermos se houve sucesso na extração do conhecimento, comparamos os modelos obtidos com os resultados que seriam esperados através de métricas como o MAE (*Mean Absolute Error*) ou a *accuracy*.

4. Dataset grupo: Avaliação de vinhos

1 Contextualização

Nesta etapa do trabalho prático, tivemos de procurar um *dataset* onde conseguíssemos extrair o máximo de conhecimento possível. Ora, como o segundo *dataset* refere-se a um problema de classificação, decidimos partir à procura de um problema de regressão. Ora, depois de alguma pesquisa, encontramos um *dataset* de avaliação de vinhos de várias regiões do mundo na plataforma *Kaggle*, podendo ser obtido aqui.

O objetivo principal é prever a pontuação dada a um vinho consoante uma série de fatores como a sua região, ano de colheita, ...

Este *dataset* contém as seguintes features:

- `Points`: número de pontos(de 1 a 100) atribuídos ao vinho;
- `Title`: título da *review* do vinho - acaba por ser o nome do vinho em questão;
- `Variety`: tipo de uvas usadas para fazer o vinho;
- `Description`: descrição do vinho sobre o sabor, cheiro, aparência, ... ;
- `Country`: país de origem do vinho;
- `Province`: província ou estado de origem do vinho;
- `Region 1`: área da província/estado de origem do vinho;
- `Region 2`: área mais específica da província/estado de origem do vinho;
- `Winery`: adega detentora do vinho;
- `Designation`: vinha da adega onde cresceram as uvas constituintes do vinho;
- `Price`: preço de uma garrafa do vinho em questão;
- `Taster Name`: nome da pessoa que provou e fez a *review* do vinho;
- `Taster Twitter Handle`: *nickname* na plataforma *Twitter* do nome da pessoa que provou e fez a *review* do vinho

#	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twit...	title	variety	winery
0	Italy	Aromas include tropical fruit, broom, brimstone and dried herb. The palate isn't overly expressive, ...	Vulkà Bianco	87		Sicily & Sardinia	Etna		Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
1	Portugal	This is ripe and fruity, a wine that is smooth while still structured. Firm tannins are filled out w...	Avidagos	87	15.0	Douro			Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
2	US	Tart and snappy, the flavors of lime flesh and rind dominate. Some green pineapple pokes through, wi...		87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainstorm
3	US	Pineapple rind, lemon pith and orange blossom start off the aromas. The palate is a bit more opulent...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore		Alexander Peartree		St. Julian 2013 Reserve Late Harvest Riesling (Lake Michigan Shore)	Riesling	St. Julian
4	US	Much like the regular bottling from 2012, this comes across as rather rough and tannic, with rustic,...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Sweet Cheeks 2012 Vintner's Reserve Wild Child Block Pinot Noir (Willamette Valley)	Pinot Noir	Sweet Cheeks

Figura 4.1: Primeiras 5 linhas deste *dataset*

2 Análise e exploração dos dados

Na fase de análise e exploração dos dados deste *dataset*, incidimos em alguns pontos que irão ser demonstrados abaixo.

O primeiro ponto foi a obtenção da lista de atributos disponíveis, que é obtida através do comando `df.info()` - os atributos obtidos são os listados na secção anterior. Observámos vários atributos do tipo `object` que devem ser transformados para valores numéricos de forma a serem suportados pelos modelos de *machine learning*, assim como eventuais atributos desnecessários.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129971 entries, 0 to 129970
Data columns (total 14 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   Unnamed: 0             129971 non-null  int64  
 1   country                129908 non-null  object 
 2   description            129971 non-null  object 
 3   designation            92506 non-null   object 
 4   points                 129971 non-null  int64  
 5   price                  120975 non-null  float64
 6   province               129908 non-null  object 
 7   region_1               108724 non-null  object 
 8   region_2               50511 non-null   object 
 9   taster_name            103727 non-null  object 
 10  taster_twitter_handle  98758 non-null   object 
 11  title                  129971 non-null  object 
 12  variety                129970 non-null  object 
 13  winery                 129971 non-null  object 
dtypes: float64(1), int64(2), object(11)
memory usage: 13.9+ MB
```

Figura 4.2: Lista de *features*

De seguida, partimos para a visualização do conteúdo do *dataset* (cuja *print* já se encontra na secção anterior) e procuramos fazer uma análise estatística dos dados iniciais. Podemos observar que, embora o atributo `points` seja de classificação que iria de 0-100, neste *dataset* apenas temos valores entre 80 e 100 e que o valor máximo de preço é muito superior à sua média, havendo a eventual necessidade de tratarmos dos *outliers*.

```
df.describe()
```

	Unnamed: 0	points	price
count	129971.000000	129971.000000	120975.000000
mean	64985.000000	88.447138	35.363389
std	37519.540256	3.039730	41.022218
min	0.000000	80.000000	4.000000
25%	32492.500000	86.000000	17.000000
50%	64985.000000	88.000000	25.000000
75%	97477.500000	91.000000	42.000000
max	129970.000000	100.000000	3300.000000

Figura 4.3: Estatística

Além disso, também procuramos saber quais são os valores únicos no *dataset* para termos uma noção de quais as *features* que devem manter-se e quais devem ser removidas.

- **Análise dos valores únicos das features**

```
for c in df:
    print(f"{c}: {df[c].unique()}")
    print(f"Quantidade: {df[c].nunique()}")
    print("----------------------------------------")
```

```
Quantidade: 425
----------------------------------------
region_1: ['Etna' nan 'Willamette Valley' ... 'Del Veneto' 'Bardolino Superiore'
 'Paestum']
Quantidade: 1229
----------------------------------------
region_2: [nan 'Willamette Valley' 'Napa' 'Sonoma' 'Central Coast' 'Oregon Other'
 'Central Valley' 'North Coast' 'Columbia Valley' 'California Other'
 'Finger Lakes' 'Sierra Foothills' 'New York Other' 'Long Island'
 'Napa-Sonoma' 'Southern Oregon' 'Washington Other' 'South Coast']
Quantidade: 17
----------------------------------------
taster_name: ['Kerin O'Keefe' 'Roger Voss' 'Paul Gregutt' 'Alexander Peartree'
 'Michael Schachner' 'Anna Lee C. Iijima' 'Virginie Boone' 'Matt Kettmann'
 nan 'Sean P. Sullivan' 'Jim Gordon' 'Joe Czerwinski'
 'Anne Krebiehl\xa0MW' 'Lauren Buzzeo' 'Mike DeSimone' 'Jeff Jenssen'
 'Susan Kostrzewa' 'Carrie Dykes' 'Fiona Adams' 'Christina Pickard']
Quantidade: 19
----------------------------------------
taster_twitter_handle: ['@kerinokeefe' '@vossroger' '@paulgwine\xa0' nan '@wineschach' '@vboone'
```

Figura 4.4: Análise dos valores únicos

Por fim, vimos qual a distribuição da *feature target* `points` para vermos se os modelos conseguem fazer as suas previsões de forma precisa, facto que se verifica analisando o gráfico.

```
df.points.hist()
```

```
<AxesSubplot:>
```

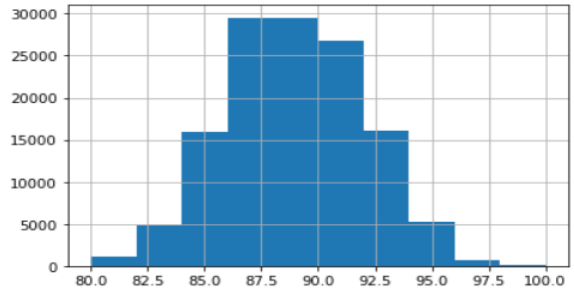


Figura 4.5: Distribuição do atributo alvo `points`

3 Tratamento dos dados

Nesta etapa, iremos preparar os dados de forma a que possam ser utilizados nos modelos de *machine learning* que aplicaremos na secção seguinte.

Em primeiro lugar, removemos quatro features:

- `Unnamed:0`: apenas serve como um ID da linha do *dataset*;
- `designation`: contém informação em falta e vários valores únicos para um atributo categórico;
- `description`: contém imensos valores únicos para um atributo categórico sem qualquer significado para os modelos a aplicar;
- `region_1`: especificação do atributo `region_1`, com vários valores únicos;
- `region_2`: mais uma especificação do atributo `region_1`, com imensos valores em falta;
- `taster_twitter_handle`: é uma outra designação para `tester_name`, pelo que é um atributo redudante;
- `winery`: contém imensos valores únicos para um atributo categórico sem qualquer significado para os modelos a aplicar mas com importância no mundo real.

```
remove_features_list = ["Unnamed: 0",'designation', "description",'region_1', "region_2", "taster_twitter_handle","winery"]
for ft in remove_features_list:
    df = df.drop(ft, axis=1)
```

Figura 4.6: Remoção de atributos

De seguida, passamos ao preenchimentos dos valores em falta das *features* `price` com o seu valor médio, `country`, `province` e `variety` com o respetivo valor mais frequente e `taster_name` com o valor `unknown`.

```
df.price.fillna(df.price.mean(),inplace =True)
df.country.fillna(str(df.country.mode()),inplace =True)
df.province.fillna(str(df.province.mode()),inplace =True)
df.taster_name.fillna('unknown',inplace =True)
df.variety.fillna(str(df.variety.mode()),inplace =True)
```

Figura 4.7: Tratamentos dos valores em falta

O próximo passo é tratar o atributo `title`. Conseguimos observar que em todas as entradas desta *feature* contêm o ano da colheita do vinho, incluindo a variedade do vinho (expressa em `variety`), a respetiva adega - que, explicada anteriormente, estava expressa no atributo `winery` - e a sua `designation`, atributos que foram removidos e cuja explicação já foi dada no relatório. Assim, de forma a retirar o ano da colheita, utilizamos a expressão regular `.*\d{4}.*` para identificarmos se todas as entradas contém, de facto, o ano, facto comprovado na imagem abaixo.

```
df[df.title.str.contains(r'.*\d{4}.*')].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 125362 entries, 0 to 129970
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   country      125362 non-null  object
 1   points       125362 non-null  int64
 2   price        125362 non-null  float64
 3   province     125362 non-null  object
 4   taster_name  125362 non-null  object
 5   title        125362 non-null  object
 6   variety      125362 non-null  object
dtypes: float64(1), int64(1), object(5)
memory usage: 7.7+ MB
```

Figura 4.8: Verificação da existência do ano

Uma vez que uma grande maioria realmente contêm o ano - apenas 4608 registos é que não contêm o ano num total de 129970 - extraímos a data e alteramos o nome da coluna para um nome mais sugestivo - `year`.

```
df.title = df.title.str.replace(r'(.|\n)*(\d{4})(.|\n)*',r'\2',regex=True)
df.title = df.title[df.title.str.contains(r'\d{4}')]
```

Mudar nome da feature para year

```
rename_map = {'title':'year'}
df.rename(columns=rename_map,inplace=True)
```

Figura 4.9: Extração do ano e mudança do nome da coluna

De seguida, passamos para o preenchimento dos valores em falta do atributo `year` com o valor mais frequente (moda) e removemos os registos duplicados.

Preencher valores nulos com a moda

```
df.year.fillna(df.year.mode().astype(int).values[0],inplace=True)
```

```
df.year = df.year.astype(int)
```

- **Remoção de duplicados**

```
df.drop_duplicates(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 97886 entries, 0 to 129970
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   country      97886 non-null  object
 1   points       97886 non-null  int64
 2   price        97886 non-null  float64
 3   province     97886 non-null  object
 4   taster_name  97886 non-null  object
 5   year         97886 non-null  int32
 6   variety      97886 non-null  object
dtypes: float64(1), int32(1), int64(1), object(4)
memory usage: 5.6+ MB
```

Figura 4.10: Preenchimento dos valores em falta de `year` e remoção de duplicados

Nesta fase, ainda temos alguns atributos do tipo *object* e, para os transformarmos em numéricos, é necessário aplicar o método de *nominal value discretization*. Neste método, iremos fazer um mapeamento de um inteiro com o valor do dado na escala do respetivo atributo.

Depois de todas estas preparações relatadas acima, temos então o seguinte *dataset* que irá estar sujeito a vários modelos de *machine learning*.

Número de valores únicos e tipo de cada feature

```
columns = df.columns.values
for c in columns:
    print(f"{c} : {df[c].nunique()}  \n   type : {df[c].dtype}")
```

```
country : 44
   type : int64
points : 21
   type : int64
price : 391
   type : float64
province : 426
   type : int64
taster_name : 20
   type : int64
year : 168
   type : int32
variety : 708
   type : int64
```

Labeling de features tipo object

```
for c in df.columns.values:
    if(df[c].dtype=='object'):
        labels = df[c].astype('category').cat.categories.tolist()
        replace_map_comp = {c : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}
        df.replace(replace_map_comp,inplace=True)

df.head()
```

	country	points	price	province	taster_name	year	variety
0	24	87	35.363389	333	10	2013	692
1	33	87	15.000000	110	16	2011	452
2	42	87	14.000000	270	15	2013	438
3	42	87	13.000000	220	1	2013	481
4	42	87	65.000000	270	15	2012	442

Figura 4.11: *Nominal value discretization* em atributos categóricos e *dataset* preparado

4 Modelos desenvolvidos

Nesta secção, iremos descrever o processo usado na aplicação dos modelos de *machine learning* para os 3 com o valor de MAE(*Mean Absolute Error*) mais pequenos. Esta fase é bastante importante onde não só se pretende obter modelos que sejam capazes de representar o *dataset* em questão, como também fazer previsões corretas a partir de dados desconhecidos. Além disto, o modelo a desenvolver não pode ser demasiado simples de forma a evitar *underfitting*, como também não deverá ser demasiado "perfeito" - funcionar muito bem para os dados de treino, mas falhar ao fazer previsões com dados que nunca viu (*overfitting*).

O primeiro passo consiste na divisão dos dados do nosso *dataset* em dados para treino (0.20%) e em dados para teste (0.80%).

De seguida, tratámos de normalizar os dados, caso os modelos requeiram os dados dessa forma, com recurso à função `MinMaxScaler`.

A seguir, aplicamos os modelos em questão (cuja demonstração irá ser demonstrada através de imagens abaixo em apenas um - Decision Tree Regressor) e, posteriormente, tratamos de fazer o seu *tunning*. Os outros modelos podem ser consultados no notebook `tarefa_grupo.ipynb`.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2022)
clf = DecisionTreeRegressor(random_state=2022)
clf.fit(x_train,y_train)
predictions_train = clf.predict(x_train)
predictions_test = clf.predict(x_test)
```

```
print(f'Train MAE {mean_absolute_error(y_train, predictions_train)}')
print(f'Test MAE {mean_absolute_error(y_test, predictions_test)}')
```

```
Train MAE 0.7477383453101132
Test MAE 2.359416058255316
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {'max_depth': [3, 4, 5, 6],
              'min_samples_leaf': [0.05, 0.1, 0.2]}

model = DecisionTreeRegressor(random_state = 2022)

grid_search = GridSearchCV(model, param_grid, cv=6)

grid_search.fit(x_train, y_train)

print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)

predictions_train = grid_search.predict(x_train)
predictions_test = grid_search.predict(x_test)
train_score = mean_absolute_error(y_train, predictions_train)
test_score = mean_absolute_error(y_test, predictions_test)
if test_score <= train_score:
    print("Model is overfitting")
    print(f"-> train: {train_score}")
    print(f"-> test: {test_score}")
else:
    print("Model is not overfitting")
    print(f"-> train: {train_score}")
    print(f"-> test: {test_score}")
```

```
Best parameters:  {'max_depth': 6, 'min_samples_leaf': 0.05}
Best score:  0.36009371646450977
Model is not overfitting
-> train: 1.9924499452489746
-> test: 2.0216642621900847
```

Figura 4.12: Aplicação e respetivo *tuning* do modelo Decision Tree

5 Resultados e análises

De forma a agregar todos os resultados de todos os modelos aplicados, elaboramos a tabela abaixo:

Modelos	MAE
Decision Tree Regressor	2.0216
Linear Regression	2.3237
Artificial Neural Network	1.9
Support Vector Regressor	2.2369
Voting (SVR + DTR + LR)	2.0846
Bagging (DTR)	1.9989
Stacking (XGB + RFR + LR)	1.8004
AdaBoost	2.0590
XGBoost	1.8034
Random Forest Regressor	1.9915

Tal como podemos observar, o modelo com os melhores resultados - com valores de erro absoluto médio - foi o Stacking, modelo incluído no Ensemble Learning. Aqui utilizamos Support Vector Regressor, Random Forest Regressor e Linear Regression. O resultado poderá variar consoante a inclusão de novos *learners*.

5. Dataset competição: Incidentes rodoviários

1 Contextualização

A modelação de incidentes rodoviários é um conhecido problema de características estocásticas, não-lineares. Tem, contudo, aparecido na literatura um conjunto de modelos que demonstram um potencial assinalável neste tipo de previsões. Com isso em consideração, foi construído um *dataset* que contém dados referentes à quantidade e características dos incidentes rodoviários que ocorreram numa cidade portuguesa em 2021, mais propriamente na cidade de Guimarães.

O objetivo principal neste caso de estudo passa é desenvolver modelos de *Machine Learning* capazes de prever o nível de incidentes rodoviários, numa determinada hora, na referida cidade. Com isto, dar-se-á ênfase ao atributo `incidents` que indica, numa escala qualitativa (*None*, *Low*, *Medium*, *High* e *Very_High*), a quantidade de incidentes rodoviários que se verificaram num determinado ponto temporal - o valor `None` significa que não existiram incidentes e o valor *Very_High* implica a existência de uma quantidade muito alta de incidentes rodoviários.

No que toca a esta parte do trabalho prático, a equipa docente criou uma competição na plataforma Kaggle onde todos os grupos concorrem uns com os outros de forma a apurar quem tem o modelo com a melhor *performance* possível.

Este *dataset* contém as seguintes *features*:

- `city_name` - nome da cidade em causa;
- `record_date` - o timestamp associado ao registo;

- `magnitude_of_delay` - magnitude do atraso provocado pelos incidentes que se verificam no `record_date` correspondente;
- `delay_in_seconds` - atraso, em segundos, provocado pelos incidentes que se verificam no record_date correspondente;
- `affected_roads` - estradas afectadas pelos incidentes que se verificam no record_date correspondente;
- `luminosity` - o nível de luminosidade que se verificava na cidade de Guimarães;
- `avg_temperature` - valor médio da temperatura para o `record_date` na cidade de Guimarães;
- `avg_atm_pressure` - valor médio da pressão atmosférica para o `record_date` na cidade de Guimarães;
- `avg_humidity` - valor médio de humidade para o `record_date` na cidade de Guimarães;
- `avg_wind_speed` - valor médio da velocidade do vento para o `record_date` na cidade de Guimarães;
- `avg_precipitation` - valor médio de precipitação para o `record_date` na cidade de Guimarães;
- `avg_rain` - avaliação qualitativa do nível de precipitação para o `record_date`e na cidade de Guimarães;
- `incidents` - indicação acerca do nível de incidentes rodoviários que se verificam no `record_date` correspondente na cidade de Guimarães.

city_name	magnitude...	delay_in_s...	affected_r...	record_date	luminosity	avg_temp...	avg_atm_p...	avg_humid...	avg_wind_...	avg_preci...	avg_rain	incidents
Guimaraes	UNDEFINED	0	,	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	0.0	Sem Chuva	None
Guimaraes	UNDEFINED	385	N101,	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	None
Guimaraes	UNDEFINED	69	,	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	0.0	Sem Chuva	Low
Guimaraes	MAJOR	2297	N101,R206,N105, N101,N101,N101, N101,N101,N101, N101,N101	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	0.0	Sem Chuva	Very_High
Guimaraes	UNDEFINED	0	N101,N101,N101, N101,N101,	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	0.0	Sem Chuva	High

Figura 5.1: Primeiras 5 linhas deste *dataset*

2 Análise e exploração dos dados

Na fase de análise e exploração dos dados deste *dataset*, incidimos em alguns pontos que irão ser demonstrados abaixo.

O primeiro ponto foi a obtenção da lista de atributos disponíveis, que é obtida através do comando `df.info()` - os atributos obtidos são os listados na secção anterior. Observámos alguns atributos do tipo `object` que devem ser transformados para valores numéricos de forma a serem suportados pelos modelos de *machine learning*: `city_name`, `magnitude_of_delay`, `affected_roads`, `record_date`, `luminosity`, `avg_rain` e `incidents`.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   city_name           5000 non-null   object
 1   magnitude_of_delay  5000 non-null   object
 2   delay_in_seconds    5000 non-null   int64
 3   affected_roads      4915 non-null   object
 4   record_date         5000 non-null   object
 5   luminosity          5000 non-null   object
 6   avg_temperature     5000 non-null   float64
 7   avg_atm_pressure    5000 non-null   float64
 8   avg_humidity        5000 non-null   float64
 9   avg_wind_speed      5000 non-null   float64
 10  avg_precipitation   5000 non-null   float64
 11  avg_rain            5000 non-null   object
 12  incidents           5000 non-null   object
dtypes: float64(5), int64(1), object(7)
memory usage: 507.9+ KB
```

Figura 5.2: Lista de *features*

De seguida, partimos para a visualização do conteúdo do *dataset* (cuja *print* já se encontra na secção anterior) e procuramos fazer uma análise estatística dos dados iniciais.

```
df.describe()
```

	delay_in_seconds	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_precipitation
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0
mean	560.567000	14.583000	1018.145000	74.455000	1.253500	0.0
std	1686.859581	4.820514	5.174372	17.204638	1.269847	0.0
min	0.000000	1.000000	997.000000	6.000000	0.000000	0.0
25%	0.000000	11.000000	1015.000000	63.000000	0.000000	0.0
50%	0.000000	14.000000	1019.000000	78.000000	1.000000	0.0
75%	234.000000	18.000000	1022.000000	90.000000	2.000000	0.0
max	31083.000000	35.000000	1032.000000	100.000000	10.000000	0.0

Figura 5.3: Estatística

Além disso, também procuramos saber quais são os valores únicos no *dataset* para termos uma noção de quais as *features* que devem manter-se e quais devem ser removidas.

```
for c in df:
    print(f"{c}: {df[c].unique()}")
    print(f"Quantidade: {df[c].nunique()}")
    print("----------------------------------------")
```

```
----------------------------------------
record_date: ['2021-03-15 23:00' '2021-12-25 18:00' '2021-03-12 15:00' ...
 '2021-03-18 03:00' '2021-11-02 06:00' '2021-12-20 02:00']
Quantidade: 5000
----------------------------------------
luminosity: ['DARK' 'LIGHT' 'LOW_LIGHT']
Quantidade: 3
----------------------------------------
avg_temperature: [12. 14. 15. 27.  9.  8. 16. 23. 11.  7. 17. 19.  6. 13. 18. 10.  4. 21.
 22. 20. 32.  5. 28. 25. 24. 29. 33. 26.  1. 31.  3. 30.  2. 34. 35.]
Quantidade: 35
----------------------------------------
avg_atm_pressure: [1013. 1007. 1025. 1028. 1020. 1015. 1026. 1012. 1017. 1018. 1030. 1016.
 1003. 1019. 1008. 1014. 1023. 1021. 1011. 1024. 1022. 1027. 1006. 1010.
 1004. 1009. 1002. 1000. 1005.  997. 1031. 1029. 1001.  998. 1032.  999.]
Quantidade: 36
----------------------------------------
avg_humidity: [ 70.  91.  64.  75.  52.  94.  87.  71.  67.  40.  76.  60.  65.  83.
  82.  61.  88.  92.  66.  56.  55.  72.  93.  90.  47.  73.  62.  77.
  79.  84.  42.  78.  46.  58.  63.  45.  59.  74.  89.  68.  81.  86.
```

Figura 5.4: Valores únicos de todas as *features*

Por fim, vimos qual a distribuição da *feature target* `incidents` para vermos se os modelos conseguem fazer as suas previsões de forma precisa, facto que se verifica analisando o gráfico.

```
df["incidents"].hist()
```

```
<AxesSubplot:>
```

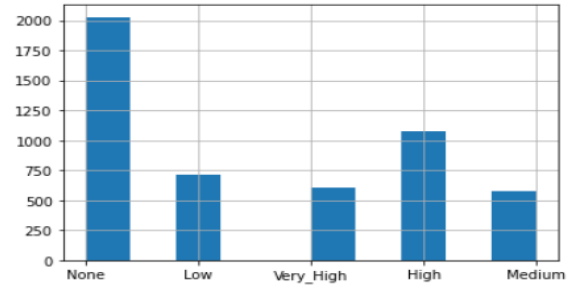


Figura 5.5: Distribuição da variável *target* `incidents`

3 Tratamento dos dados

Nesta etapa, iremos preparar os dados de forma a que possam ser utilizados nos modelos de *machine learning* que aplicaremos na secção seguinte.

Em primeiro lugar, removemos quatro features: `avg_humidity` por possuir uma forte correlação com o atributo `avg_temperature`, `avg_precipitation`, por possui informação incompleta, `city_name`, que contém apenas um valor ("Guimarães") e `magnitude_of_delay`, cuja informação já é fornecida de forma semelhante no atributo numérico `delay_in_seconds`.

Remoção da feature "avg_humidity"

```
#forte correlação com avg_temperature
df = df.drop('avg_humidity', axis=1)
df_teste = df_teste.drop('avg_humidity', axis=1)
```

Remoção da feature "avg_precipitation"

```
#Possui informação incompleta
df = df.drop('avg_precipitation', axis=1)
df_teste = df_teste.drop('avg_precipitation', axis=1)
```

Remoção da feature "magnitude_of_delay"

```
df = df.drop('magnitude_of_delay', axis=1)
df_teste = df_teste.drop('magnitude_of_delay', axis=1)
```

```
#como apenas existe um valor para "city_name", esta coluna pode ser removida
df = df.drop('city_name', axis=1)
df_teste = df_teste.drop('city_name', axis=1)
```

Figura 5.6: Remoção de `avg_humidity`, `avg_precipitation`, `city_name` e `magnitude_of_delay`

De seguida, aplicamos *feature engineering* ao atributo *record_date* com a divisão em ano, mês, dia, hora e minuto, com o intuito de melhorar o desempenho dos modelos.

Transformação da coluna *record_date* nas colunas ano, mês, dia, hora e minuto

```
df['record_date'] = pd.to_datetime(df['record_date'],format='%Y-%m-%d %H:%M', errors='coerce')
assert df['record_date'].isnull().sum() == 0,'missing record date'

df['record_date_year'] = df['record_date'].dt.year
df['record_date_month'] = df['record_date'].dt.month
df['record_date_day'] = df['record_date'].dt.day
df['record_date_hour'] = df['record_date'].dt.hour
df['record_date_minute'] = df['record_date'].dt.minute
df = df.drop('record_date', axis=1)
```

Figura 5.7: *Feature engineering* ao atributo *record_date*

Depois da análise das novas *features* criadas, observámos que existem algumas que são desnecessárias ter, tais como a antiga `record_date`, `record_date_year` - apenas contém o valor "2021"e `record_date_minute` - apenas contém o valor "0".

```
print(df["record_date_year"].unique())
print(df["record_date_minute"].unique())

df = df.drop('record_date_minute', axis=1)
df = df.drop('record_date_year', axis=1)
```

```
[2021]
[0]
```

Figura 5.8: Remoção dos atributos `record_date_year` e `record_date_minute`

O próximo passo é aplicar *nominal value discretization* aos atributos `luminosity`, `avg_rain`

e `incidents`, uma vez que todos contêm dados do tipo *object* (categórico), pelo que têm de ser transformados para o tipo numérico, existindo um mapeamento de um inteiro com o valor do dado na escala do atributo. Por exemplo, na *feature* `avg_rain`, o valor 0 irá responder ao valor `Sem Chuva`, o valor 1 irá responder ao valor `chuva fraca`, o valor 3 irá responder ao valor `chuva moderada` e o o valor 4 irá responder ao valor `chuva forte`.

Figura 5.9: *Nominal value discretization* aos atributos `luminosity`, `avg_rain` e `incidents`

De seguida, sabendo que o *dataset* contém o atributo `affected_roads`, temos de converter os seus dados para o tipo numérico. Ora, em primeiro lugar, temos de tratar os seus *missing values*. pelo que a equipa decidiu substituí-los por `","`. Com vista a aplicação do método de *one-hot encoding*, usamos a expressão regular `([a-zA-Z]+\d+(\s*-\s*[a-zA-Z]*\d+)?)` para a captura de texto das diferentes ruas. Depois, na criação dos novos atributos, colocamos o valor "1"caso essa rua era uma das que constava na célula de *affected_roads* ou "0"caso se verifique o oposto.

Figura 5.10: *One-hot encoding* no atributo `affected_roads`

Por último, seguindo a sugestão da docente no *checkpoint* realizado, adicionamos um novo atributo que se refere ao dia da semana de cada entrada do *dataset*, com o intuito de observar se esta *feature* ajuda nos modelos.

Adição de feature "dayOfWeek" e respetiva transformação em valores numéricos

```
import datetime

def getDayofWeek(df):
    mes = int (df['record_date_month'])
    dia = int (df['record_date_day'])
    intDay = datetime.date(year=2021, month=mes, day=dia).weekday()
    days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
    return days[intDay]

df['dayOfWeek'] = df.apply(getDayofWeek, axis=1)
df_teste['dayOfWeek'] = df_teste.apply(getDayofWeek, axis=1)

df.head()
```

```
replace_map = {'dayOfWeek': {'Monday':1 ,'Tuesday': 2, 'Wednesday': 3, 'Thursday': 4, 'Friday': 5, 'Saturday': 6, 'Sunday': 7}}
df.replace(replace_map,inplace=True)
df_teste.replace(replace_map,inplace=True)
df.head()
```

Figura 5.11: Criação do atributo `dayOfWeek`

Em suma, depois de toda a preparação relatada, ficamos com o seguinte conjuntos de *features*:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   delay_in_seconds    5000 non-null   int64
 1   luminosity          5000 non-null   int64
 2   avg_temperature     5000 non-null   float64
 3   avg_atm_pressure    5000 non-null   float64
 4   avg_wind_speed      5000 non-null   float64
 5   avg_rain            5000 non-null   int64
 6   incidents           5000 non-null   int64
 7   record_date_month   5000 non-null   int64
 8   record_date_day     5000 non-null   int64
 9   record_date_hour    5000 non-null   int64
 10  N101                5000 non-null   int64
 11  R206                5000 non-null   int64
 12  N105                5000 non-null   int64
 13  N206                5000 non-null   int64
 14  N309                5000 non-null   int64
 15  IC5                 5000 non-null   int64
 16  N310                5000 non-null   int64
 17  N207-4              5000 non-null   int64
 18  IC5 - N206          5000 non-null   int64
 19  EM579               5000 non-null   int64
 20  dayOfWeek           5000 non-null   int64
dtypes: float64(3), int64(18)
memory usage: 820.4 KB
```

Figura 5.12: Lista de *features* finais

4 Modelos desenvolvidos

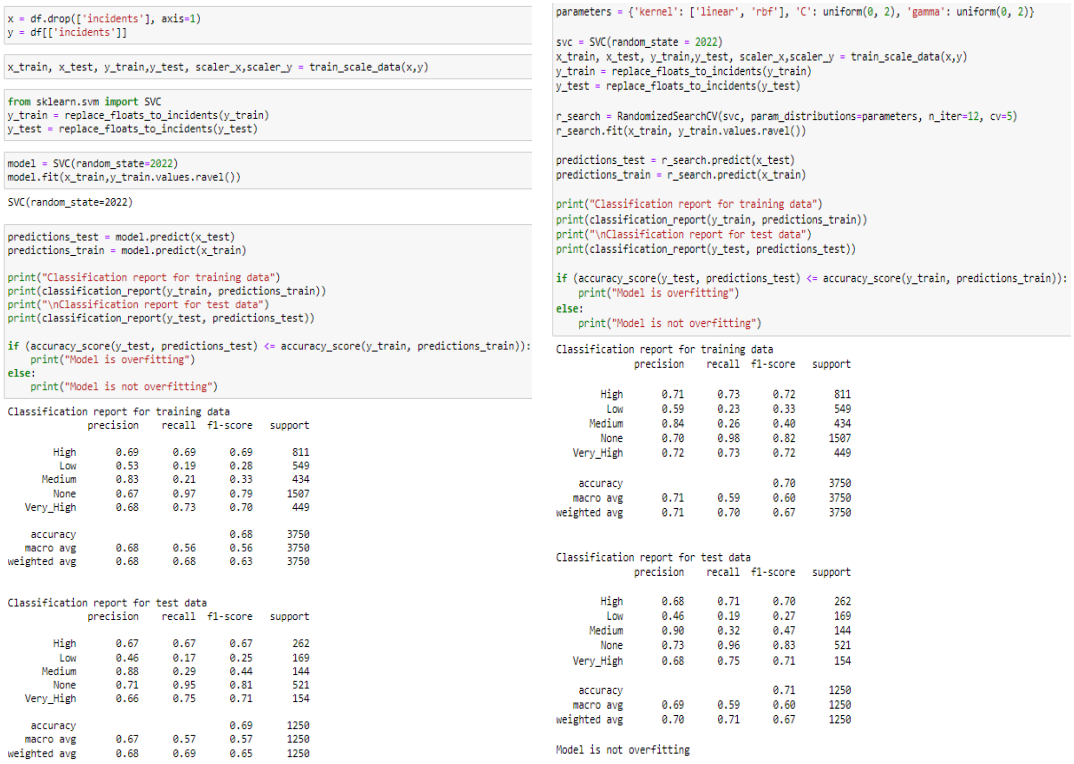
Nesta secção, iremos descrever o processo usado na aplicação dos modelos de *machine learning* para os 3 com o valor de *accuracy* mais pequenos. Esta fase é bastante importante onde não só se pretende obter modelos que sejam capazes de representar o *dataset* em questão, como também fazer previsões corretas a partir de dados desconhecidos. Além disto, o modelo a desenvolver não pode ser demasiado simples de forma a evitar *underfitting*, como também

não deverá ser demasiado "perfeito" - funcionar muito bem para os dados de treino, mas falhar ao fazer previsões com dados que nunca viu (*overfitting*).

O primeiro passo consiste na divisão dos dados do nosso *dataset* em dados para treino (0.25%) e em dados para teste (0.75%).

De seguida, tratámos de normalizar os dados, caso os modelos requeiram os dados dessa forma, com recurso à função `MinMaxScaler`.

A seguir, aplicamos os modelos em questão (cuja demonstração irá ser demonstrada através de imagens abaixo em apenas um) e, posteriormente, tratamos de fazer o seu *tunning*. Os outros modelos podem ser consultados no notebook `tarefa_kaggle.ipynb`.



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

```
x_train, x_test, y_train,y_test, scaler_x,scaler_y = train_scale_data(x,y)
```

```
from sklearn.svm import SVC
y_train = replace_floats_to_incidents(y_train)
y_test = replace_floats_to_incidents(y_test)
```

```
model = SVC(random_state=2022)
model.fit(x_train,y_train.values.ravel())
```

```
SVC(random_state=2022)
```

```
predictions_test = model.predict(x_test)
predictions_train = model.predict(x_train)

print("Classification report for training data")
print(classification_report(y_train, predictions_train))
print("\nClassification report for test data")
print(classification_report(y_test, predictions_test))

if (accuracy_score(y_test, predictions_test) <= accuracy_score(y_train, predictions_train)):
    print("Model is overfitting")
else:
    print("Model is not overfitting")
```

```
Classification report for training data
              precision    recall  f1-score   support

        High       0.69      0.69      0.69       811
         Low       0.53      0.19      0.28       549
      Medium       0.83      0.21      0.33       434
        None       0.67      0.97      0.79      1507
   Very_High       0.68      0.73      0.70       449

    accuracy                           0.68      3750
   macro avg       0.68      0.56      0.56      3750
weighted avg       0.68      0.68      0.63      3750


Classification report for test data
              precision    recall  f1-score   support

        High       0.67      0.67      0.67       262
         Low       0.46      0.17      0.25       169
      Medium       0.88      0.29      0.44       144
        None       0.71      0.95      0.81       521
   Very_High       0.66      0.75      0.71       154

    accuracy                           0.69      1250
   macro avg       0.67      0.57      0.57      1250
weighted avg       0.68      0.69      0.65      1250
```

```
parameters = {'kernel': ['linear', 'rbf'], 'C': uniform(0, 2), 'gamma': uniform(0, 2)}

svc = SVC(random_state = 2022)
x_train, x_test, y_train,y_test, scaler_x,scaler_y = train_scale_data(x,y)
y_train = replace_floats_to_incidents(y_train)
y_test = replace_floats_to_incidents(y_test)

r_search = RandomizedSearchCV(svc, param_distributions=parameters, n_iter=12, cv=5)
r_search.fit(x_train, y_train.values.ravel())

predictions_test = r_search.predict(x_test)
predictions_train = r_search.predict(x_train)

print("Classification report for training data")
print(classification_report(y_train, predictions_train))
print("\nClassification report for test data")
print(classification_report(y_test, predictions_test))

if (accuracy_score(y_test, predictions_test) <= accuracy_score(y_train, predictions_train)):
    print("Model is overfitting")
else:
    print("Model is not overfitting")
```

```
Classification report for training data
              precision    recall  f1-score   support

        High       0.71      0.73      0.72       811
         Low       0.59      0.23      0.33       549
      Medium       0.84      0.26      0.40       434
        None       0.70      0.98      0.82      1507
   Very_High       0.72      0.73      0.72       449

    accuracy                           0.70      3750
   macro avg       0.71      0.59      0.60      3750
weighted avg       0.71      0.70      0.67      3750


Classification report for test data
              precision    recall  f1-score   support

        High       0.68      0.71      0.70       262
         Low       0.46      0.19      0.27       169
      Medium       0.90      0.32      0.47       144
        None       0.73      0.96      0.83       521
   Very_High       0.68      0.75      0.71       154

    accuracy                           0.71      1250
   macro avg       0.69      0.59      0.60      1250
weighted avg       0.70      0.71      0.67      1250

Model is not overfitting
```

Figura 5.13: Aplicação e respetivo *tuning* do modelo Support Vector Machine

5 Resultados e análises

De forma a agregar todos os resultados de todos os modelos aplicados, elaboramos a tabela abaixo:

Modelos	Accuracy
Decision Tree Classifier	0.77
Logistic Regression	0.69
Support Vector Machine	0.73
K-Means	0.47
K-Medoids	0.73
Support Vector Machine	0.21
Voting (SVR + DTR + LR)	0.85
Bagging	0.77
Stacking (XGB + RFR + LR)	0.92
AdaBoost	0.81
XGBoost	0.88
Random Forest Classifier	0.81

Tal como podemos observar, o modelo com os melhores resultados - com valores de erro absoluto médio - foi o Stacking, modelo incluído no Ensemble Learning. Aqui utilizamos XGBoost, Random Forest Regressor e Linear Regression. O resultado poderá variar consoante a inclusão de novos *learners*.

Outro ponto que podemos esclarecer é o facto de os algoritmos de *clustering* aplicados - `k-means` e `k-medoids` - não serem aplicados para *datasets* com vários atributos (no nosso caso temos 20), o que explica a tão baixa taxa de acerto.

6. Conclusão e trabalho futuro

A realização deste projeto permitiu consolidar os conceitos relativos ao desenvolvimento de um projeto de Machine Learning utilizando modelos de aprendizagem, lecionados ao longo da unidade curricular.

Algumas das dificuldades que o grupo sentiu foi o facto de haver modelos em que o *tuning* não levou a um aumento da *accuracy* ou à diminuição do MAE como desejado. Além disso, também tivemos alguma dificuldade a correr as redes neuronais porque sobrecarregavam muito o GPU das nossas máquinas, havendo situações onde parava a execução, não permitindo a obtenção do *score*.

Algum trabalho futuro para este trabalho prático poderá passar pela inclusão de novos modelos ou até a melhoria dos já existentes.