Universidade do Minho
Escola de Engenharia

Unidade Curricular de

Dados e Aprendizagem Automática

Ano Letivo de 2025/2026

Beatriz Peixoto (pg59996)
Diogo Miranda (pg60001)
Martim Félix (pg58753)
Sandra Cerqueira (pg60016)

Janeiro 2026

D A A

Índice

Resumo

Este trabalho, desenvolvido no âmbito da unidade curricular **Dados e Aprendizagem Automática**, teve como objetivo o desenvolvimento e a otimização de modelos de *Machine Learning* para a previsão do nível de trânsito rodoviário na cidade do Porto. A variável alvo do problema é a ***average_speed_diff***, categorizada nos níveis *None*, *Low*, *Medium*, *High* e *Very High*. Para esse efeito, foram explorados e preparados os conjuntos de dados de treino e teste disponibilizados na plataforma *Kaggle*, que incluem variáveis temporais, de tráfego e meteorológicas.

1 Metodologia

Ao longo do desenvolvimento deste projeto foi adotada uma metodologia baseada no pipeline de *Machine Learning* utilizado ao longo das aulas de DAA. Esta metodologia encontra-se estruturada em cinco fases principais, ilustradas na *Figura 1*.

Figura 1-Metodologia pipeline de Machine Learning

- **Compreensão do Problema**
  Nesta fase procede-se à definição dos objetivos do projeto e à formulação do problema de extração de conhecimento a resolver.
- **Análise e Exploração dos Dados**
  Tem como objetivo a análise do conjunto de dados disponibilizado, incluindo a identificação de valores em falta ou inconsistentes, relações entre atributos, presença de *outliers* e avaliação da qualidade dos dados.
- **Tratamento dos Dados**
  Nesta etapa são selecionados os atributos relevantes, tratadas entradas com valores em falta ou inconsistentes e realizadas transformações necessárias para garantir a compatibilidade dos dados com os modelos de aprendizagem.
- **Modelação**
  Consiste na seleção, treino e ajuste dos modelos de Machine Learning considerados mais adequados ao problema em estudo, com base nos dados previamente preparados.
- **Avaliação do Modelo**
  Por fim, o desempenho dos modelos desenvolvidos é avaliado através da comparação entre os resultados obtidos, permitindo aferir a qualidade da extração de conhecimento.

2 Definição do problema

2.1 Contexto

A modelação do tráfego rodoviário constitui um problema complexo, caracterizado por uma natureza estocástica e não linear, sendo fortemente influenciado por diversos fatores externos. Entre estes fatores destacam-se as condições meteorológicas, a hora do dia, o dia da semana e a sazonalidade, os quais afetam significativamente os padrões de circulação rodoviária. A capacidade de prever o nível de trânsito assume um papel fundamental no apoio à gestão urbana, à mobilidade sustentável e à tomada de decisões por parte de entidades responsáveis pelo planeamento do tráfego. Neste contexto, o presente trabalho tem como foco a cidade do Porto, recorrendo a dados recolhidos entre julho de 2018 e outubro de 2019, com o objetivo de prever a gravidade do tráfego rodoviário num determinado instante temporal.

2.2 Formulação do problema

O problema em estudo pode ser formulado como um problema de aprendizagem supervisionada, mais especificamente de **classificação multiclasse**. A partir de um conjunto de variáveis explicativas, que incluem informação temporal, características do tráfego e dados meteorológicos, pretende-se prever a variável alvo '***average_speed_diff***'.

Esta variável representa a diferença entre a velocidade máxima teórica em condições de tráfego livre e a velocidade efetivamente observada, sendo posteriormente categorizada em cinco classes distintas: ***None***, ***Low***, ***Medium***, ***High*** e ***Very High***. Assim, o objetivo consiste em determinar, para cada observação, a classe que melhor representa o nível de congestionamento do tráfego rodoviário.

2.3 Objetivos do trabalho

O objetivo principal consiste no desenvolvimento e otimização de modelos de *Machine Learning* para a previsão do nível de tráfego rodoviário na cidade do Porto. De forma específica, pretende-se:

- Explorar e analisar os dados disponibilizados;
- Preparar e transformar os dados através de técnicas de pré-processamento;
- Desenvolver, otimizar e comparar diferentes modelos de *Machine Learning*;
- Avaliar criticamente os resultados e selecionar o modelo com melhor desempenho.

3 Ingestão de Dados

3.1 Fonte dos dados

Os dados foram disponibilizados no Kaggle da competição associada ao projeto, sob a forma de um conjunto de treino (training_data.csv) e um conjunto de teste (test_data.csv). Os dados disponibilizados representam aproximadamente 30% do total, sendo o restante disponibilizado após o encerramento da competição.

3.2 Particularidade da leitura

Durante a ingestão dos dados, verificou-se que, ao contrário do comum nas aulas práticas, o ficheiro não se encontrava codificado em UTF-8, tendo sido lido com a codificação Latin-1. Adicionalmente, foi necessário ter em consideração que a categoria None constitui uma classe válida da variável alvo, pelo que se ajustou a configuração de leitura de forma a evitar a sua interpretação como valor ausente:

```
train_df = pd.read_csv("training_data.csv", encoding="latin-1",keep_default_na=False, na_values=['NULL', ''])
test_df = pd.read_csv("test_data.csv", encoding="latin-1",keep_default_na=False, na_values=['NULL', ''])
```

Figura 2- Código usado na leitura dos csv

3.3 Inspeção inicial

Antes de iniciar a preparação dos dados, realizámos uma inspeção inicial ao *dataset* de treino e sumariámos na tabela no *Anexo 1* todas as variáveis disponíveis. Constatámos ainda que o *dataset* de treino contém 14 atributos e 6812 registos, enquanto o *dataset* de teste apresenta 13 atributos e 1500 registos.

4 Preparação dos Dados

De seguida iremos explicar como procedemos à preparação dos dados. Este processo foi feito de forma sequencial e estruturada, começamos com uma análise exploratória, seguida do tratamento de valores em falta e *outliers*, da remoção de atributos irrelevantes e, por fim, da criação de novas *features*. Tudo isto teve como objetivo melhorar a qualidade dos dados e potenciar o desempenho dos modelos de aprendizagem automática.

4.1 Análise Exploratória (EDA)

Começámos então por fazer a análise exploratória que teve como objetivo compreender a estrutura do conjunto de dados fornecido, identificar padrões relevantes e detetar eventuais problemas de qualidade, como valores em falta ou observações atípicas.

Tipo de dados dos atributos

Com recurso ao método *info* foi possível obter alguma informação inicial como o tipo de dados ou o número de valores não nulos em cada atributo.

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6812 entries, 0 to 6811
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   city_name               6812 non-null   object
 1   record_date             6812 non-null   object
 2   AVERAGE_SPEED_DIFF      6812 non-null   object
 3   AVERAGE_FREE_FLOW_SPEED 6812 non-null   float64
 4   AVERAGE_TIME_DIFF       6812 non-null   float64
 5   AVERAGE_FREE_FLOW_TIME  6812 non-null   float64
 6   LUMINOSITY              6812 non-null   object
 7   AVERAGE_TEMPERATURE     6812 non-null   float64
 8   AVERAGE_ATMOSP_PRESSURE 6812 non-null   float64
 9   AVERAGE_HUMIDITY        6812 non-null   float64
 10  AVERAGE_WIND_SPEED      6812 non-null   float64
 11  AVERAGE_CLOUDINESS      4130 non-null   object
 12  AVERAGE_PRECIPITATION   6812 non-null   float64
 13  AVERAGE_RAIN            563 non-null    object
dtypes: float64(8), object(6)
memory usage: 745.2+ KB
```

Figura 3-Análise inicial dos dados

A partir destes resultados, verificou-se que algumas colunas eram do tipo *object*, tornando necessário o seu tratamento posterior para que pudessem ser utilizadas pelos modelos de aprendizagem.

Análise das variáveis numéricas

De seguida**,** foi realizada uma análise estatística descritiva das variáveis numéricas, tal como mostrado na tabela abaixo.

```
train_df.describe()
```

	AVERAGE_FREE_FLOW_SPEED	AVERAGE_TIME_DIFF	AVERAGE_FREE_FLOW_TIME	AVERAGE_TEMPERATURE	AVERAGE_ATMOSP_PRESSURE	AVERAGE_HUMIDITY	AVERAGE_WIND_SPEED	AVERAGE_PRECIPITATION
count	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.0
mean	40.661010	25.637111	81.143952	16.193482	1017.388139	80.084190	3.058573	0.0
std	4.119023	33.510507	8.294401	5.163492	5.751061	18.238863	2.138421	0.0
min	30.500000	0.000000	46.400000	0.000000	985.000000	14.000000	0.000000	0.0
25%	37.600000	2.275000	75.400000	13.000000	1015.000000	69.750000	1.000000	0.0
50%	40.700000	12.200000	82.400000	16.000000	1017.000000	83.000000	3.000000	0.0
75%	43.500000	36.200000	87.400000	19.000000	1021.000000	93.000000	4.000000	0.0
max	55.900000	296.500000	112.000000	35.000000	1033.000000	100.000000	14.000000	0.0

Figura 4-Estatísticas dos dados

A análise descritiva no conjunto de treino mostra que as variáveis associadas ao trânsito como a *'AVERAGE_FREE_FLOW_SPEED'* e a *'AVERAGE_FREE_FLOW_TIME'* apresentam um menor desvio-padrão e intervalo interquartil, ou seja, uma menor variabilidade refletindo o comportamento base do trânsito. Já a *'AVERAGE_TIME_DIFF'* apresenta maior dispersão, o que indica a ocorrência de episódios de congestionamento. As variáveis meteorológicas como a temperatura, a humidade e a pressão atmosférica apresentam distribuições coerentes com o clima da cidade do porto. Além disso, a variável *'AVERAGE_PRECIPITATION'* não apresenta variabilidade e é constante igual a 0, pelo que não é relevante do ponto de vista preditivo.

Para complementar esta análise estatística e permitir uma interpretação mais intuitiva das distribuições, foram gerados histogramas para cada variável numérica.

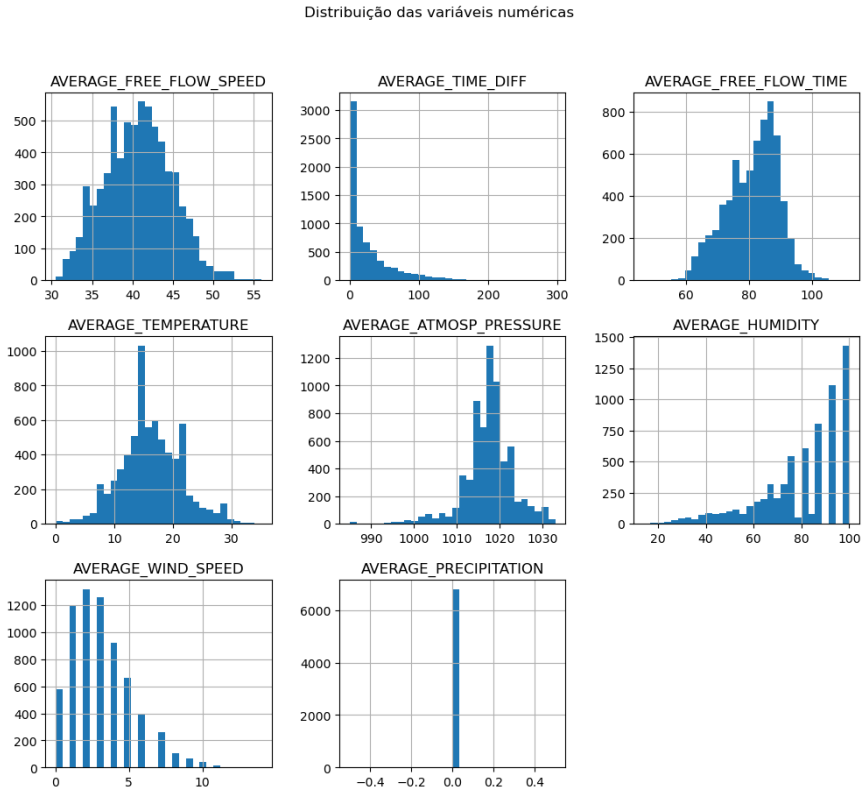

Figura 5-Distribuição dos atributos

Os histogramas confirmam visualmente as conclusões obtidas na análise estatística. Observaram-se variáveis com distribuições aproximadamente normais como a *'AVERAGE_FREE_FLOW_SPEED'*, *'AVERAGE_TEMPERATURE'*,

'AVERAGE_ATMOSP_PRESSURE' e *'AVERAGE_FREE_FLOW_TIME'*; variáveis com distribuições assimétricas como a *'AVERAGE_TIME_DIFF'*, *'AVERAGE_WIND_SPEED'* e *'AVERAGE_HUMIDITY'*; uma variável com uma distribuição quase constante que é o caso da *'AVERAGE_PRECIPITATION'* que apresenta um único valor.

Análise de correlação entre variáveis

Após a caracterização individual das variáveis, analisaram-se as relações entre as variáveis numéricas através de uma matriz de correlação, apresentada na figura abaixo.

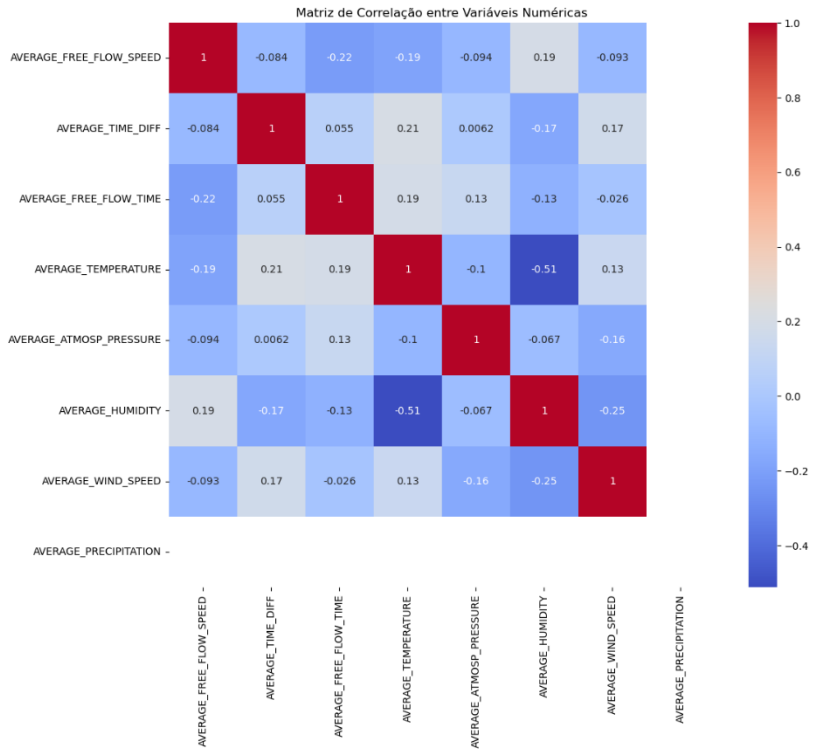

Figura 6-Matriz de correlação entre variáveis numéricas

É possível destacar uma correlação negativa moderada entre as variáveis *'AVERAGE_TEMPERATURE'* e *'AVERAGE_HUMIDITY'*, o que é esperado para variáveis meteorológicas. Já as restantes variáveis apresentam correlações mais fracas, não se observando relações lineares muito fortes. Além disso, a variável *'AVERAGE_PRECIPITATION'* aparece na matriz, mas não apresenta valores de correlação uma vez que tem valor constante igual a 0, pelo que não contribui para a análise de relações entre variáveis.

Análise de variáveis categóricas

Paralelamente à análise das variáveis numéricas, procedeu-se à análise das variáveis categóricas. Esta análise revelou que '*city_name'* apresenta um único valor (Porto). A variável *'AVERAGE_CLOUDINESS'* contém vários valores redundantes, semanticamente equivalentes, tal como mostrado na imagem abaixo.

```
train_df['AVERAGE_CLOUDINESS'].value_counts(dropna=False)
```

```
AVERAGE_CLOUDINESS
NaN                 2682
céu claro           1582
céu pouco nublado    516
nuvens dispersas     459
nuvens quebrados     448
algumas nuvens       422
nuvens quebradas     416
céu limpo            153
tempo nublado         67
nublado               67
Name: count, dtype: int64
```

Figura 7-Valores da variável 'AVERAGE_CLOUDINESS'

Adicionalmente, analisou-se a distribuição da variável alvo, verificando-se que esta apresenta uma distribuição desequilibrada, em que a classe *None* é a mais frequente enquanto a classe *Very_High* é minoritária, algo que será tido em conta no desenvolvimento dos modelos de *Machine Learning*.

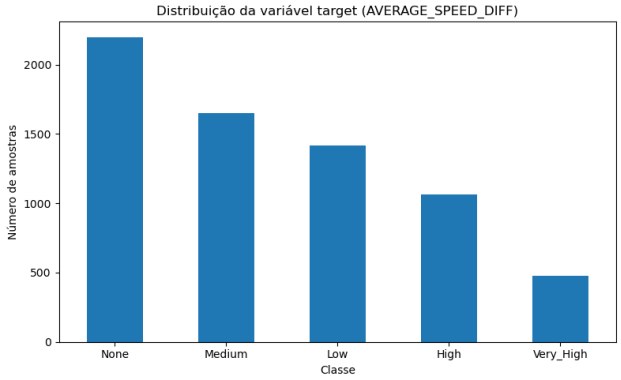

Figura 8-Distribuição da variável alvo

Análise de valores em falta

Tendo identificado os principais padrões e distribuições, foi então analisada a existência de valores em falta no conjunto de dados. Constatámos a sua presença nas colunas *'AVERAGE_CLOUDINESS'* e *'AVERAGE_RAIN'*. Posteriormente, determinamos a percentagem de *missing values* na coluna *'AVERAGE_RAIN'* que é de aproximadamente 91.7%.

```
train_df.isna().sum()
city_name                   0
record_date                 0
AVERAGE_SPEED_DIFF          0
AVERAGE_FREE_FLOW_SPEED     0
AVERAGE_TIME_DIFF           0
AVERAGE_FREE_FLOW_TIME      0
LUMINOSITY                  0
AVERAGE_TEMPERATURE         0
AVERAGE_ATMOSP_PRESSURE     0
AVERAGE_HUMIDITY            0
AVERAGE_WIND_SPEED          0
AVERAGE_CLOUDINESS       2682
AVERAGE_PRECIPITATION       0
AVERAGE_RAIN             6249
dtype: int64
```

Figura 9-Análise dos valores em falta

```
missing_percent = train_df['AVERAGE_RAIN'].isnull().mean() * 100
print(missing_percent)
```

Figura 10-Cálculo da percentagem de missing values

```
sns.heatmap(train_df.isnull(), cbar=False)
plt.title("Mapa de Missing Values")
plt.show()
```

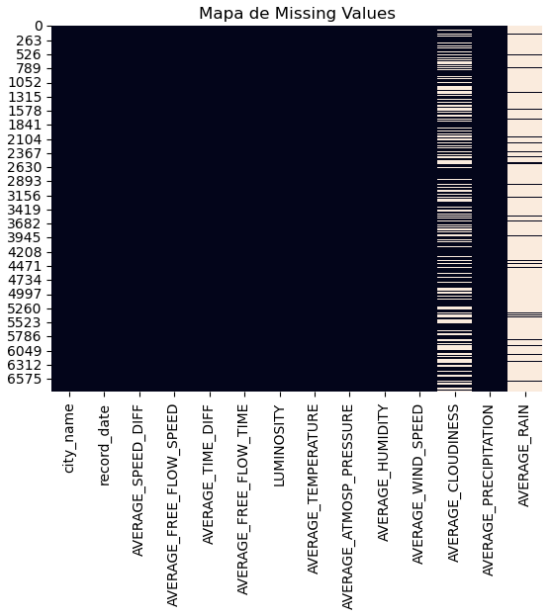

Figura 11-Heatmap de valores em falta

Análise de valores atípicos (*outliers*)

Por fim, no âmbito da análise exploratória, foi avaliada a presença de valores atípicos através de *box plots*. Nas imagens abaixo são apresentados os gráficos para as variáveis *'AVERAGE_FREE_FLOW_SPEED'* e *'AVERAGE_TIME_DIFF'*, onde é possível ver a existência de valores atípicos.

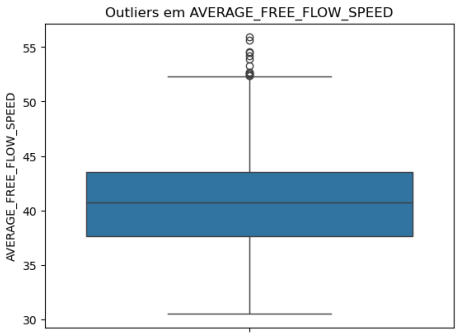

Figura 12-Boxplot variável 'AVERAGE_FREE_FLOW_SPEED'

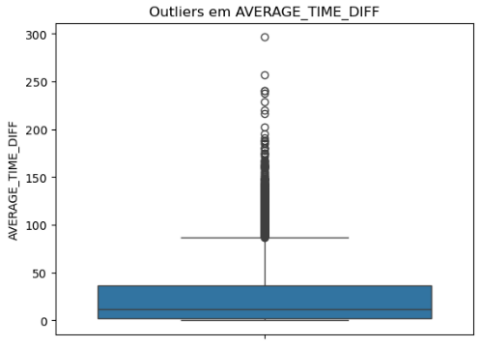

Figura 13-Boxplot variável 'AVERAGE_TIME_DIFF'

4.2 Tratamento de *missing values*

Com base nas conclusões da análise exploratória, o tratamento dos valores em falta foi definido individualmente para cada variável, tendo em conta o seu potencial impacto na modelação. A variável '***AVERAGE_RAIN'***, apresentava uma elevada percentagem de valores em falta e, numa fase inicial, chegou a ser removida do conjunto de dados. Contudo, reconsideramos a nossa escolha, e optamos por assumir que os valores em falta correspondem a **situações de ausência de chuva**. Desta forma, conseguimos preservar informação contextual relevante associada às condições meteorológicas. Com a variável sendo reconstruída como uma variável ordinal, sendo ela a *'RAIN_INTENSITY'*, que representa diferentes níveis de intensidade de chuva numa escala de 0 a 4. Com 0 correspondendo à ausência de chuva, enquanto valores mais elevados representam intensidades maiores de precipitação.

Relativamente à variável ***'AVERAGE_CLOUDINESS'***, os valores em falta foram tratados como não havendo nuvens, desta forma foram imputados com o valor 0. Desta forma, conseguimos evitar a perda de observações e garantimos a consistência da variável.

4.3 Tratamento de *outliers*

Apesar da identificação de valores atípicos na fase exploratória, optou-se por não aplicar qualquer tratamento de *outliers*, uma vez que os modelos utilizados são robustos a este tipo de valores e estes podem representar situações reais de trânsito. A sua remoção ou transformação poderia, assim, comprometer o desempenho do modelo.

4.4 Remoção de Features

Após o tratamento dos valores em falta e a avaliação dos *outliers*, procedeu-se à remoção de *features* consideradas irrelevantes ou redundantes, com o objetivo de simplificar o modelo e evitar ruído desnecessário.

- **Tratamento da '*AVERAGE_RAIN*':** Conforme definido no tratamento de *missing values*, a informação da variável '*AVERAGE_RAIN'* foi incorporada na *feature* ordinal *'RAIN_INTENSITY'*, levando à remoção da variável original por redundância.
- **Variáveis de Variância Nula:** As *features* '*city_name'* e *'AVERAGE_PRECIPITATION'* foram removidas por apresentarem variância nula, assumindo um único valor em todo o conjunto de dados. A ausência de variabilidade implica que estas variáveis não possuem poder discriminativo, não contribuindo para a capacidade preditiva do modelo.

4.5 *Feature engineering*

Numa fase final da preparação dos dados, foram aplicadas várias técnicas de *feature engineering*, com o objetivo de enriquecer o conjunto de dados e permitir que os modelos captassem padrões temporais, meteorológicos e contextuais mais complexos

4.5.1 Conversão e data engineering sobre 'record_date'

A variável temporal *record_date*, que era inicialmente categórica, foi convertida para o formato *datetime* de modo a permitir a extração de informação temporal relevante. Deste modo, a partir desta variável foram derivadas novas variáveis como o ano, o mês, o dia, a hora, o minuto, o segundo e também o dia da semana.

```
train_df['record_date'] = pd.to_datetime(train_df['record_date'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
train_df['record_date_year'] = train_df['record_date'].dt.year
train_df['record_date_month'] = train_df['record_date'].dt.month
train_df['record_date_day'] = train_df['record_date'].dt.day
train_df['record_date_hour'] = train_df['record_date'].dt.hour
train_df['record_date_minute'] = train_df['record_date'].dt.minute
train_df['record_date_seconds'] = train_df['record_date'].dt.second

# Day of week (0 = Monday, 6 = Sunday)
train_df['day_of_week'] = train_df['record_date'].dt.dayofweek
```

Figura 14-Criação de novas features a partir da data

Através destas novas variáveis derivadas verificamos através do comando *nunique()* que as variáveis *'record_date_minute'* e *'record_date_seconds'* não apresentavam variação relevante, pelo que foram removidas. Depois de extrair a informação relevante, a variável *'record_date'* também foi removida.

4.5.2 Encoding das variáveis categóricas

A codificação das variáveis categóricas foi realizada tendo em conta a sua natureza ordinal e o significado real das categorias, de forma a preservar relações de ordem relevantes para o modelo. A variável '*AVERAGE_CLOUDINESS'* foi tratada como uma variável ordinal, uma vez que as suas categorias representam níveis crescentes de nebulosidade. As diferentes descrições textuais foram mapeadas para uma escala discreta crescente, de modo a refletir a progressão desde o céu limpo até céu totalmente nublado.

```
cloud_map = {
    "céu claro": 1,
    "céu limpo": 1,
    "céu pouco nublado": 2,
    "algumas nuvens": 2,
    "nuvens dispersas": 2,
    "nuvens quebrados": 3,
    "nuvens quebradas": 3,
    "tempo nublado": 4,
    "nublado": 4
}
```

Figura 15-Maping da variável AVERAGE_CLOUDINESS

A variável *'LUMINOSITY'* foi da mesma forma convertida para uma escala ordinal, uma vez que as suas categorias representam níveis crescentes de luminosidade.

```
luminosity_mapping = {
    'DARK': 0,
    'LOW_LIGHT': 1,
    'LIGHT': 2
}
```

Figura 16-Maping da variável LUMINOSITY

Por fim, a variável alvo '*AVERAGE_SPEED_DIFF'*, originalmente representada por categorias ordenadas, foi codificada de forma ordinal numa escala de 0 a 4, correspondendo a níveis crescentes de impacto na diferença de velocidade. Desta forma, garantimos que a variável é compatível com algoritmos de aprendizagem automática que requerem variáveis numéricas.

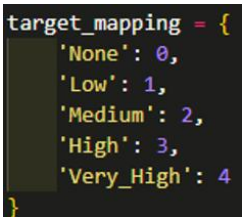

Figura 17-Maping da variável AVERAGE_SPEED_DIF

4.5.3 Data engineering de features metereológicas (Dew Point)

Para tentar capturar a interação entre a temperatura e a humidade, calculou-se o Ponto de Orvalho (*DEW_POINT*) através da biblioteca *MetPy*, uma medida que representa a temperatura de saturação do ar. Para assegurar a estabilidade numérica e mitigar a influência de *outliers* ou erros de sensor, as variáveis de entrada foram previamente restringidas a intervalos físicos plausíveis, a temperatura a [−20,45] ºC e a humidade a [1,100] % , sendo o resultado arredondado a uma casa decimal para reduzir o ruído.

Complementarmente, foram derivadas duas variáveis para enriquecer a modelação: a *TEMP_MINUS_DP*, que quantifica a depressão do ponto de orvalho e serve como *proxy* para o grau de secura do ar, e a *DEW_POINT_BIN* uma discretização ordinal que, ao agrupar os valores contínuos, reduz a sensibilidade do modelo a pequenas oscilações ou mudanças na distribuição dos dados. Para tal, a variável foi mapeada em cinco níveis correspondentes aos intervalos [−50,0], [0,5], [5,10], [10,15] e (15,50], privilegiando assim a capacidade de generalização.

```
bins = [-50, 0, 5, 10, 15, 50]
labels = [0, 1, 2, 3, 4]

df["DEW_POINT_BIN"] = pd.cut(df["DEW_POINT"], bins=bins, labels=labels, include_lowest=True).astype("int64")
return df
```

Figura 18-Discretização do Ponto de Orvalho

4.5.4 Hora de Ponta

Foram definidas como horas de ponta os períodos [7, 8, 9, 17, 18, 19, 20], com base em estudos do Metro do Porto (*https://www.metrodoporto.pt/uploads/document/file/399/3038_-_Estudo_de_Procura_Linha_Amarela_20170911.pdf*). A partir desta definição, foram criadas duas variáveis: hora_de_ponta (binária) e tipo_hora_de_ponta (ordinal), que classifica a intensidade da hora de ponta em três níveis (0: fora, 1: menos acentuada, 2: mais acentuada).

4.5.5 Fim de semana e feriados

Com base na variável *'record_date'* e no atributo derivado *'day_of_week'* foi criada a variável binária *'fim_de_semana'* que identifica se um dia corresponde a um sábado ou domingo. Esta nova variável permite distinguir padrões de trânsito diferentes entre dias úteis e fins de semana.

Além disso, foi criada uma variável binária *'is_holiday'* que identifica se uma data corresponde a um feriado na cidade do porto. Esta variável tem como objetivo capturar alterações e padrões típicos de dias de feriado no trânsito. Para isso foi criada a lista de todos os feriados municipais, que se encontra no Anexo 2.

```
def e_fim_de_semana(row):
    weekday = int(row["day_of_week"])

    if weekday in (5,6):
        return 1
    return 0
```

Figura 20-Criação da variável fim_de_semana

```
def is_holiday(row):

    year = int(row['record_date_year'])
    month = int(row['record_date_month'])
    day = int(row['record_date_day'])

    date = pd.Timestamp(year=year, month=month, day=day)
    return 1 if date in holidays_pt else 0
```

Figura 19-Criação da variável is_holiday

4.5.5 Eventos no Porto

Com o objetivo de captar o impacto de eventos na mobilidade urbana, foi introduzida uma variável binária, a ***is_event_day***, relacionada com a ocorrência de eventos relevantes na cidade do Porto.

```python
def load_event_dates(file_path):
    event_dates = pd.read_csv(file_path, header=None, names=['date', 'event'])
    event_dates['date'] = pd.to_datetime(event_dates['date'], format='%Y-%m-%d', errors='coerce')
    return event_dates

event_dates = load_event_dates('eventos.csv')

def is_event_day(row, event_dates):
    year = int(row['record_date_year'])
    month = int(row['record_date_month'])
    day = int(row['record_date_day'])

    date = pd.Timestamp(year=year, month=month, day=day)
    return 1 if date in event_dates['date'].values else 0

train_df['is_event_day'] = train_df.apply(is_event_day, axis=1, event_dates=event_dates)
print(train_df['is_event_day'].value_counts())

test_df['is_event_day'] = test_df.apply(is_event_day, axis=1, event_dates=event_dates)
print(test_df['is_event_day'].value_counts())
```

Figura 21-Criação da variável is_event_day

Foram considerados eventos de grande afluência, nomeadamente jogos do F.C. Porto, festivais de música, celebrações académicas, eventos culturais e festivais tradicionais, por terem a capacidade de influenciar significativamente os padrões de tráfego, as datas desses eventos estão datadas no ficheiro ***eventos.csv***.

```
2019-05-04, jogo
2019-05-18, jogo
2019-06-05, jogo
2019-06-09, jogo
2019-08-13, jogo
2019-08-17, jogo
2019-09-01, jogo
2019-09-19, jogo
2019-09-22, jogo
2019-09-25, jogo
2018-06-07, NOS Primavera Sound (festival)
2018-06-08, NOS Primavera Sound (festival)
2018-06-09, NOS Primavera Sound (festival)
2019-06-06, NOS Primavera Sound (festival)
2019-06-07, NOS Primavera Sound (festival)
2019-06-08, NOS Primavera Sound (festival)
2018-06-23, Festa de São João do Porto
2019-06-23, Festa de São João do Porto
2019-05-05, Queima das Fitas do Porto  (início)
2019-05-06, Queima das Fitas do Porto
2019-05-07, Queima das Fitas do Porto
2019-05-08, Queima das Fitas do Porto
2019-05-09, Queima das Fitas do Porto
2019-05-10, Queima das Fitas do Porto
2019-05-11, Queima das Fitas do Porto (fim)
2019-09-06, Feira do Livro do Porto (início)
2019-09-07, Feira do Livro do Porto
2019-09-08, Feira do Livro do Porto
2019-09-09, Feira do Livro do Porto
2019-09-10, Feira do Livro do Porto
```

Figura 22-Excerto do eventos.csv

4.5.6 Afluência a Supermercados

Foi criada a variável '***supermarket_peak_level'***, inferida a partir do dia da semana e da hora, classificando a afluência em três níveis: 0 (baixa), 1 (moderada) e 2 (forte), de acordo com a informação contida no seguinte link:https://www.sibsanalytics.com/wp-content/uploads/2024/05/5-anos-SIBS-Analytics-PT.pdf

4.5.7 *Encoding* Cíclico

Para preservar a natureza periódica das variáveis temporais e evitar descontinuidades artificiais (por exemplo, entre 23h e 0h, ou entre domingo e segunda-feira), aplicou-se ***encoding* cíclico** às componentes *record_date_hour*, *day_of_week* e *record_date_month*. Cada componente foi transformada em duas novas variáveis, através das funções seno e cosseno, segundo a parametrização $\sin(2\pi x/P)$ e $\cos(2\pi x/P)$, onde P representa o período da variável (24 para a hora, 7 para o dia da semana e 12 para o mês). Desta forma, foram criadas as features *hour_sin*, *hour_cos*, *dow_sin*, *dow_cos*, *month_sin* e *month_cos*, permitindo que o modelo capture padrões sazonais e de periodicidade (diária, semanal e anual) de forma contínua e geometricamente consistente.

```python
def add_cyclical(df: pd.DataFrame) -> pd.DataFrame:
    # hour (0-23)
    df["hour_sin"] = np.sin(2 * np.pi * df["record_date_hour"] / 24)
    df["hour_cos"] = np.cos(2 * np.pi * df["record_date_hour"] / 24)
```

Figura 23-Encoding Ciclico da Hora

4.5.8 *Congestion* Ratio

Com o objetivo de capturar o grau de congestionamento do tráfego de forma normalizada e comparável entre diferentes contextos, foi criada a variável *congestion_ratio*, definida como o quociente entre o tempo médio observado e o tempo em condições de fluxo livre:

$$\text{congestion\_ratio} = \frac{\text{AVERAGE\_TIME\_DIFF}}{\text{AVERAGE\_FREE\_FLOW\_TIME} + \varepsilon}$$

onde $\varepsilon = 10^{-9}$ foi introduzido para garantir divisão segura e evitar instabilidades numéricas em casos de denominador nulo ou muito pequeno. Nesta definição:

- *'AVERAGE_TIME_DIFF'* corresponde ao tempo adicional face ao cenário sem trânsito;
- *'AVERAGE_FREE_FLOW_TIME'* corresponde ao tempo esperado sem congestionamento.

Sendo um indicador adimensional, valores mais elevados traduzem maior degradação do desempenho do tráfego face ao cenário ideal:

- $\approx \mathbf{0}$: fluxo normal, sem atraso relevante;
- $> \mathbf{1}$: aumento significativo do tempo de viagem, funcionando como indicador direto de trânsito/congestionamento;
- $<\mathbf{1}$: possível presença de medições anómalas ou ruído (por exemplo, leituras demasiado otimistas).

5 *Data Segregation*

5.1 Divisão em treino e validação

Numa fase inicial do projeto, os dados foram divididos em conjuntos de treino, validação e teste, com o objetivo de permitir uma avaliação preliminar do desempenho dos modelos e apoiar a seleção inicial de algoritmos e hiperparâmetros. Esta abordagem possibilitou uma primeira análise da capacidade de generalização dos modelos desenvolvidos.

5.2 Modelo final

Numa fase posterior do projeto, e com o objetivo de obter uma avaliação mais robusta e fiável do desempenho dos modelos, foi adotada uma estratégia de validação cruzada *(cross-validation)*. Esta metodologia permitiu utilizar de forma mais eficiente os dados disponíveis, reduzindo a dependência de uma única partição dos dados e conduzindo à seleção do modelo final com melhor desempenho médio.

6 Treino dos modelos

Após a fase de exploração e preparação dos dados, procedeu-se ao desenvolvimento dos modelos de previsão do nível de tráfego rodoviário. Esta etapa teve como principal objetivo construir modelos com boa capacidade de generalização, assegurando um equilíbrio adequado entre complexidade e desempenho. Para esse efeito, foi seguida uma abordagem **incremental e estruturada**, permitindo avaliar progressivamente o impacto das diferentes decisões tomadas ao longo do processo de modelação. Atendendo ao desbalanceamento da variável alvo, tentámos aplicar a técnica de SMOTE, contudo, não observamos melhorias consistentes no desempenho dos modelos, razão pela qual esta abordagem não foi integrada no pipeline final.

Numa fase inicial, recorreu-se a uma abordagem de *hold-out validation* e a modelos de base, nomeadamente **Decision Tree** e **Random Forest**, utilizando exclusivamente os atributos originais do dataset, com *feature engineering* reduzido. Esta etapa teve como objetivo estabelecer uma referência inicial de desempenho. Seguidamente, foi aplicado **Grid Search** a estes modelos,

permitindo a otimização dos principais hiperparâmetros. Posteriormente, foram introduzidos novos atributos resultantes de *feature engineering*, como o *dew point*, a identificação de horas de ponta e variáveis relacionadas com a precipitação, os quais foram novamente avaliados com os modelos anteriores. Numa fase posterior, foram explorados modelos mais complexos, nomeadamente **XGBoost** e **redes neuronais**, utilizando o conjunto de atributos enriquecido. Por fim, os modelos mais promissores foram avaliados através de **validação cruzada**, permitindo uma seleção final mais robusta e fundamentada.

6.1 Modelos testados

6.1.1 *Decision Tree*

Tal como referido anteriormente, decidimos recorrer a uma *Decision Tree* para resolver o problema, visto que este modelo foi bastante abordado nas aulas da unidade curricular. Estes modelos estão entre os métodos mais usados em *Machine Learning*. Estas árvores são treinadas de acordo com um conjunto de dados de treino e posteriormente, outros exemplos serão classificados de acordo com essa mesma árvore. Em Python, foi usado o modelo ***DecisionTreeClassifier*** da biblioteca ***Scikit-learn.*** Assim sendo, definiu-se um modelo ao qual se deu o nome de ***dt_model***. Inicialmente fizemos um modelo simples tal como se vê na imagem seguinte:

```
dt_model = DecisionTreeClassifier(random_state=42, class_weight="balanced", criterion="gini")
```

Figura 24-Modelo da Decision Tree

Internamente obtivemos uma *accuraçy* média de ≈ 0.7358 e um desvio padrão de ≈ 0.0197. Já no *Kaggle* obtivemos uma *accuracy* de 0.74888. Embora estes resultados sejam aceitáveis, verificou-se margem para melhoria, o que motivou a otimização dos hiperparâmetros do modelo.

A otimização foi realizada através do **GridSearchCV**, recorrendo a validação cruzada estratificada (*StratifiedKFold* com 5 *folds*), de forma a preservar a distribuição original das classes. Foram considerados hiperparâmetros estruturais da árvore, nomeadamente: *max_depth*, *min_samples_split*, *min_samples_leaf* e o *criterion*. A métrica utilizada para otimização foi a *accuracy*, em concordância com a métrica adotada na competição.

Após a execução do *grid search*, foi selecionada a melhor combinação de hiperparâmetros, dando origem ao modelo final de **Decision Tree**. Os valores obtidos foram: *criterion* = **entropy**, *max_depth* = **5**, *min_samples_leaf* = **10** e *min_samples_split* = **2**, correspondendo a uma *accuracy* interna de **0.7660**. Este modelo apresentou um desempenho superior e mais consistente, tendo alcançado uma *accuracy* de **0.8222** no Kaggle. Estes resultados confirmam que a otimização de hiperparâmetros contribuiu de forma significativa para a melhoria do desempenho do modelo.

6.1.2 *Random Forest*

Após explorarmos as *Decision Trees*, uma técnica que permite criar modelos preditivos simples, surge o *ensemble Random Forest*, que pode ser visto como uma evolução dessas árvores de decisão. Em Python, foi utilizado o modelo **RandomForestClassifier** da biblioteca *Scikit-learn*. Deste modo, fez-se o seguinte modelo simples chamado **rf_model**:

```
rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    class_weight=class_weights,
    max_features= "sqrt",
    min_samples_leaf= 2,
    min_samples_split= 2,
    n_jobs=-1
)
```

Figura 25-Modelo da Random Forest

Internamente obtivemos uma *accuraçy* média de ≈ 0.8006 e um desvio padrão de ≈ 0.0098. Já no *Kaggle* obtivemos uma *accuracy* de 0.80444. À semelhança do que foi feito com a Decision Tree, aplicamos o **GridSearchCV**, recorrendo a validação cruzada estratificada com 5 *folds*. Foram considerados hiperparâmetros relevantes do modelo, nomeadamente o *n_estimators*,

a *max_depth*, o *min_samples_split*, o *min_samples_leaf* e o *max_features*. Atendendo ao desbalanceamento das classes, a métrica utilizada na otimização foi o **F1-weighted**. Após a execução do *grid search*, foi selecionada a melhor combinação de hiperparâmetros, correspondendo a **n_estimators = 300**, **max_depth = None**, **min_samples_leaf = 3**, **min_samples_split = 2** e **max_features = 'sqrt'**, tendo sido obtido um valor de **F1-weighted de 0.8032** na validação interna. O modelo final apresentou ainda uma *accuracy* média de **0.8020** na validação cruzada, com um desvio-padrão de **0.0088**, evidenciando um desempenho estável e consistente. Este processo deu origem ao modelo final de Random Forest, que alcançou ainda uma accuracy de 0.80666 no Kaggle.

6.1.3 *XGBoost*

Após a aplicação de modelos baseados em árvores e ensembles como o *Random Forest*, foi também explorado o algoritmo ***XGBoost*** da biblioteca ***Scikit-learn***, um método de boosting muito utilizado em problemas de classificação supervisionada devido à sua elevada capacidade preditiva e eficiência computacional. Em Python, recorreu-se à implementação *XGBClassifier* da biblioteca *XGBoost*.

Numa primeira abordagem, foi implementada uma versão mais simples do modelo, recorrendo a validação cruzada estratificada com 5 *folds*, de forma a preservar a distribuição original das classes.

O modelo simples de *XGBoost* foi configurado com hiperparâmetros conservadores, nomeadamente *max_depth* = 5, *n_estimators* = 100 e *learning_rate* = 0.1. Os resultados obtidos na validação cruzada indicaram um *F1-score weighted* médio de aproximadamente **0.8055**, com um desvio padrão de cerca de **0.0093**, evidenciando um desempenho estável entre os diferentes folds. Após o treino do modelo com a totalidade dos dados de treino, foi gerado o ficheiro de submissão para o *Kaggle*, onde se obteve uma *accuracy* de **0.82444**, um dos melhores resultados entre os modelos testados.

Posteriormente, foi desenvolvida uma versão mais avançada deste modelo, incorporando um conjunto mais alargado de funcionalidades, como *early stopping* e otimização de hiperparâmetros através de *RandomizedSearchCV*.

Esta versão apresentou um desempenho superior no *Kaggle*, obtendo uma *accuracy* de **0.83777**. No entanto, importa referir que o modelo selecionado para a fase final do projeto não corresponde ao que obteve a melhor performance no dataset privado. Em particular, o modelo escolhido mantém a base do *XGBoost* utilizado, mas incorpora mais *feature engineering*, o que motivou a sua seleção por apresentar um melhor equilíbrio global de generalização e robustez, apesar de não maximizar a *accuracy* no conjunto privado.

Embora esta versão mais complexa tenha permitido uma exploração mais aprofundada do espaço de hiperparâmetros e uma afinação mais cuidada do modelo, a sua descrição detalhada será apresentada numa secção posterior do relatório, onde é analisado o Modelo Candidato, selecionado como a solução final adotada no projeto.

6.1.4 Rede neuronal

Para abordar o problema de classificação, foi também explorada uma **Rede Neuronal Artificial do tipo Multilayer Perceptron (MLP)**, implementada recorrendo à biblioteca **PyTorch.** Este modelo permite capturar relações não lineares entre as variáveis de entrada, constituindo uma alternativa mais flexível aos modelos baseados em árvores.

A arquitetura definida é composta por **duas camadas ocultas**, com 64 e 32 neurónios, respetivamente, utilizando a função de ativação **ReLU**. A camada de saída é constituída por 5 neurónios, correspondentes às classes da variável alvo, sendo utilizada a função de perda **CrossEntropyLoss**, que aplica internamente a função *softmax*. De forma a reduzir o risco de *overfitting*, posteriormente aplicamos a regularização através de **Dropout** com taxa de 0.1 em ambas as camadas ocultas. Antes do treino, as variáveis de entrada foram normalizadas através de *StandardScaler*, de forma a garantir estabilidade no processo de aprendizagem da rede.

O desempenho do modelo foi avaliado recorrendo a **validação cruzada com 5 folds (K-Fold)**. Em cada fold, a rede foi treinada durante 9 épocas, sendo avaliadas métricas como **accuracy**,

precision, **recall** e **F1-score**, todas calculadas de forma ponderada. Os resultados médios obtidos na fase de validação cruzada indicam uma accuracy média de **0.7772 ± 0.0115**, uma precisão média de **0.7775**, um recall médio de **0.7772** e um F1-score médio de **0.7757**, revelando um desempenho consistente entre os diferentes folds.
Adicionalmente, foi analisada a curva média de *loss* de treino e de validação ao longo das épocas, conforme ilustrado na figura seguinte:

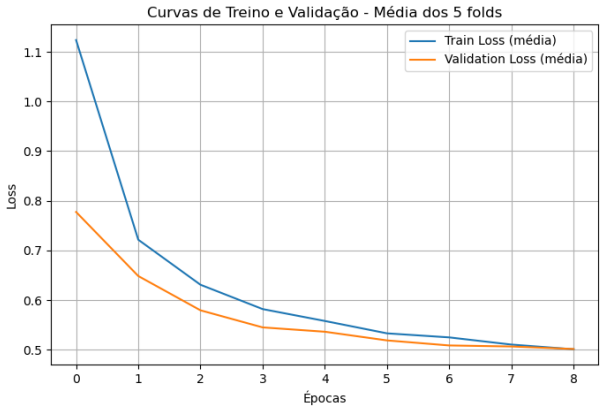

Figura 26-Curva de loss por epoch

Observa-se uma diminuição acentuada da *loss* nas primeiras épocas, seguida de uma redução mais gradual, indicando uma aprendizagem eficaz dos padrões iniciais. As curvas de treino e validação mantêm-se próximas ao longo do processo, sem divergências significativas, o que sugere um comportamento estável do modelo e ausência de sinais evidentes de *overfitting*. Após a validação cruzada, o modelo foi treinado utilizando a totalidade do conjunto de treino. No Kaggle este modelo teve um desempenho muito inferior que os restantes modelos obtendo uma *accuracy* de 0.76000, este baixo desempenho pode estar associado ao facto de este ser um modelo que tem melhor desempenho quando o volume de dados é grande, algo que não se verifica no nosso projeto.

6.1.5 *Ensembles*

Por fim, exploramos diversas abordagens de ensemble learning, onde foram utilizadas técnicas de *Bagging, Boosting, Voting* e *Stacking*.
Foram treinados vários modelos, entre eles, *Decision Tree, Random Forest, XGBoost*, todos configurados com os mesmos hiperparâmetros mencionados nos pontos anteriores.
No caso do *Bagging*, este foi implementado utilizando uma *Decision Tree* como estimador base, conforme ilustrado abaixo.

```
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(
        class_weight=class_weight,
        random_state=RSEED
    ),
    n_estimators=300,
    bootstrap=True,
    random_state=RSEED,
    n_jobs=-1
)
```

Figura 27-Modelo do Ensemble Bagging

Outro método aplicado foi o *Voting*, que combinou os modelos *Decision Tree* e *Random Forest* através de *hard voting*, onde a classe final é definida pela maioria dos votos, tal como se observa na seguinte figura.

```
voting = VotingClassifier(
    estimators=[
        ("dt", dt),
        ("rf", rf)
    ],
    voting="hard"
)
```

Figura 28-Modelo do Ensemble Voting

Por fim, recorremos ao *Stacking*, que integra *Decision Tree*, *Random Forest* e *XGBoost*, cujas previsões servem de entrada para um modelo final baseado em Regressão Logística, ilustrado na figura abaixo.

```
stacking = StackingClassifier(
    estimators=[
        ("dt", dt),
        ("rf", rf),
        ("xgb", xgb)
    ],
    final_estimator=LogisticRegression(class_weight="balanced",  random_state=RSEED),
    cv=5,
    n_jobs=-1
)
```

Figura 29-Modelo do Ensemble Stacking

Para avaliar e comparar o desempenho dos diferentes modelos, utilizámos a função *evaluate_cv*, que aplica validação cruzada. Esta função devolve o *F1-score*, que é uma métrica particularmente adequada para cenários com classes desbalanceadas, como é o nosso caso, bem como o respetivo desvio padrão. Desta forma, foi possível analisar o desempenho de cada modelo, obtendo os seguintes resultados:

```
RESULTADOS FINAIS:

          Model   F1_mean    F1_std
5      Stacking  0.808568  0.011112
2       XGBoost  0.807250  0.009937
1  RandomForest  0.801091  0.008708
3       Bagging  0.794096  0.009363
4        Voting  0.788538  0.013841
0  DecisionTree  0.736431  0.012859
```

Figura 30-Resultados finais dos modelos Ensemble

A partir da imagem apresentada, verificamos que o modelo com melhor desempenho foi o *Stacking*, alcançando um F1-score médio de 0.808568 e um desvio padrão de 0.0111112, resultando numa pontuação de **0.82222** no *Kaggle*.

6.2 Melhores modelos encontrados

De forma a resumir o trabalho desenvolvido, nesta secção elaboramos a tabela que se encontra no *Anexo 3*, onde apresentamos os melhores resultados obtidos dentro de todos os modelos testados ao longo do projeto. Para cada modelo foram indicados os hiperparâmetros finais, bem como as respetivas métricas de desempenho interno e a *accuracy* obtida no *Kaggle*, permitindo uma visão consolidada dos modelos desenvolvidos.

A análise comparativa destes modelos, presentes na tabela, evidencia que as abordagens baseadas em árvores e métodos ensemble, em particular o *XGBoost* e o *stacking*, apresentam um desempenho superior tanto na validação interna como na avaliação no *Kaggle*, reforçando a adequação destes modelos ao nosso problema. O *XGBoost* destacou-se pelo melhor compromisso entre capacidade de generalização e desempenho preditivo, enquanto o modelo de *stacking* beneficiou da combinação de diferentes classificadores, traduzindo-se em resultados competitivos. Por outro lado, a rede neuronal apresentou um desempenho inferior, o que poderá estar relacionado com a sua maior sensibilidade à quantidade de dados disponíveis e à necessidade de uma melhor afinação dos hiperparâmetros.

7. Avaliação do modelo escolhido para final

7.1 Justificação da escolha do modelo candidato

Com base na análise comparativa apresentada na secção 6.2 e nos resultados consolidados no *Anexo 3*, o modelo *XGBoost* foi selecionado como modelo final do projeto, após uma comparação detalhada com os restantes modelos testados. A decisão não se baseou exclusivamente na métrica de *accuracy*, mas sim num conjunto de critérios que inclui o desempenho global, a estabilidade na validação interna e a capacidade de generalização. Em termos objetivos, este modelo apresentou o melhor compromisso entre desempenho interno e externo, registando a maior *accuracy* no *Kaggle (0.83777)* entre todas as abordagens testadas, bem como um dos valores mais elevados na avaliação interna (*accuracy* de *0.8137*).

7.2 Funcionamento do modelo e estratégia de treino

O modelo candidato foi implementado recorrendo à classe *XGBClassifier* da biblioteca *xgboost.* O código segue um pipeline bem definido, iniciando-se com a separação explícita entre variáveis explicativas e variável alvo.

Numa fase inicial, os dados de treino foram divididos em subconjuntos de treino e validação de forma estratificada, garantindo que a distribuição das classes se mantém consistente. Esta separação é utilizada para treinar um modelo baseline de *XGBoost*, com um número elevado de estimadores, cujo principal objetivo foi servir como ponto de referência para avaliar o comportamento do modelo antes do processo de otimização. O desempenho foi acompanhado através da métrica *multi-class log loss* no conjunto de validação, sendo aplicado *early stopping* para interromper o treino quando deixa de existir melhoria significativa.

A comparação entre o desempenho obtido no conjunto de treino e no conjunto de validação permitiu-nos verificar a coerência dos resultados e identificar potenciais sinais de *overfitting*.

Posto isto, passamos à otimização dos hiperparâmetros através de *RandomizedSearchCV*, utilizando validação cruzada estratificada com 5 *folds*. A pesquisa foi limitada a um conjunto reduzido de hiperparâmetros com maior impacto no desempenho, nomeadamente a profundidade das árvores, o número de estimadores, a taxa de aprendizagem e o peso mínimo das observações nos nós.

Com base nos melhores hiperparâmetros identificados, o modelo final foi treinado utilizando a totalidade do conjunto de treino, de forma a tirar partido de toda a informação disponível. Por fim, foram geradas as previsões para o conjunto de teste e construído automaticamente o ficheiro de submissão no formato compatível com o do *Kaggle*.

7.3 Resultados locais

A avaliação local do modelo candidato revelou um desempenho consistente ao longo das diferentes fases de treino. Numa primeira etapa, o modelo *baseline* treinado com *early stopping* obteve uma *accuracy* de validação de 0.8158, servindo como ponto de referência para avaliar o comportamento do modelo. Após o processo de otimização dos hiperparâmetros, foi alcançada uma *accuracy* média de 0.8137, um valor muito próximo do registado na *baseline*.

A proximidade entre estes resultados indica que o modelo apresentou um comportamento estável em diferentes partições dos dados, não se verificando sinais relevantes de *overfitting*. Desta forma, os resultados obtidos confirmam que a configuração final do modelo conseguiu manter um bom equilíbrio entre desempenho e capacidade de generalização.

7.4 Resultados no *Kaggle* do modelo final

No *Kaggle*, o modelo candidato obteve uma *accuracy* de **0.83777** no *dataset* público, correspondendo ao melhor desempenho alcançado na fase final de testes. No dataset privado, a *accuracy* foi de **0.81238**, o que representa uma descida natural quando o modelo é avaliado num subconjunto totalmente independente e não observado durante o desenvolvimento. Ainda assim, este resultado traduziu-se num desempenho global muito positivo: alcançámos o 5.º lugar no ranking do dataset privado, tendo subido 19 posições relativamente à classificação no dataset público.

Esta diferença entre *public* e *private* é expectável e reflete a variabilidade entre subconjuntos de dados, bem como o facto de o dataset privado ser uma avaliação mais exigente da generalização. Assim, apesar da redução da *accuracy*, o desempenho no dataset privado confirma que o modelo mantém uma boa capacidade de generalização e apresenta robustez em condições mais realistas de teste, o que valida a escolha do *XGBoost* como solução final para o nosso projeto.

8. Conclusões e Trabalhos futuros

Tal como referido ao longo do relatório, foram testados diversos modelos de *Machine Learning* com o objetivo de identificar a abordagem com melhor desempenho na previsão do nível de tráfego rodoviário. Após a análise comparativa dos resultados obtidos, concluiu-se que o modelo baseado em *XGBoost* foi aquele que apresentou os resultados mais consistentes e promissores, tanto na avaliação interna como na avaliação no *Kaggle*.

No âmbito da competição no *Kaggle*, foi possível avaliar o modelo desenvolvido recorrendo a dados aos quais este nunca teve acesso. Após a submissão das previsões, o modelo obteve uma accuracy de **0.83777** no leaderboard público e **0.81238** no leaderboard privado, alcançando o **5.º lugar** no ranking privado e **subindo 19 posições** relativamente à classificação no conjunto público. Estes resultados permitem concluir que o modelo apresenta uma boa capacidade de generalização, sendo a diferença observada entre os dois leaderboards indicativa de um comportamento estável e sem sinais evidentes de *overfitting*.

Assim, considera-se que o trabalho desenvolvido produziu resultados sólidos e alinhados com os objetivos definidos. Ainda assim, reconhece-se que existem possibilidades de melhoria, nomeadamente através da experimentação de novos modelos, de uma otimização mais aprofundada dos hiperparâmetros, da exploração de abordagens alternativas na fase de preparação dos dados e de um melhor *tracking* de resultados já testados ao longo do projeto.

9. Referencias Bibliográficas

https://scikit-learn.org/stable/supervised_learning.html
https://www.metrodoporto.pt/uploads/document/file/399/3038__Estudo_de_Procura_Linha_Amarela_20170911.pdf
https://fne.pt/pt/noticias/go/acontece-calendario-escolar-2018-2019
https://www.sibsanalytics.com/wp-content/uploads/2024/05/5-anos-SIBS-Analytics-PT.pdf
https://en.wikipedia.org/wiki/NOS_Primavera_Sound_2018
https://www.festivais.pt/Festivais-2019/NOS-Primavera-Sound-2019.html
https://www.coolture.pt/event/queima-das-fitas-do-porto-2019/
https://en.wikipedia.org/wiki/Festa_de_S%C3%A3o_Jo%C3%A3o_do_Porto
https://www.up.pt/portuguesuporto/2019/09/04/feira-do-livro-do-porto/
https://www.transfermarkt.pt/fc-porto/spielplan/verein/720/saison_id/2020
https://www.fpf.pt/pt/selecoes/futebol-masculino/selecao-a/jogos
https://www.introducingporto.com/public-holidays

10 Anexos

Anexo 1

Atributo	Tipo	Descrição
city_name	Categórica/String	Nome da cidade onde os dados foram registados
record_date	Categórica	Data e hora do registo, ou seja, o timestamp do registo
average_speed_diff	Categórica	Diferença entre a velocidade em cenários sem trânsito e a velocidade que realmente se verifica. Valores mais altos indicam trânsito mais lento.
average_free_flow_speed	Numérica	Velocidade média máxima que os carros podem atingir em cenários sem trânsito.

average_time_diff	Numérica	Diferença média do tempo que se demora a percorrer um determinado conjunto de ruas em relação ao tempo de viagem sem trânsito. Valores altos indicam que se demora mais tempo.
average_free_flow_time	Numérica	Tempo médio que se demora a percorrer um determinado conjunto de ruas sem trânsito.
luminosity	Categórica	Nível de luminosidade que se verificava no Porto.
average_temperature	Numérica	Valor médio da temperatura na cidade do Porto para a data e hora registada.
average_atmosp_pressure	Numérica	Valor médio da pressão atmosférica, na cidade do Porto, para a data e hora registada.
average_humidity	Numérica	Valor médio da humidade, na cidade do Porto, para a data e hora registada.
average_wind_speed	Numérica	Valor médio da velocidade do vento, na cidade do Porto, para a data e hora registada.
average_cloudiness	Categórica	Valor médio da percentagem de nuvens, na cidade do Porto, para a data e hora registada.
average_precipitation	Numérica	Valor médio da precipitação, na cidade do Porto, para a data e hora registada.
average_rain	Categórica	Avaliação qualitativa do nível de precipitação para a data e hora registada, na cidade do Porto.

Tabela 1- Tabela com as variáveis do dataset

Anexo 2

```
holidays_pt = pd.to_datetime([
    '2018-01-01', # Ano Novo
    '2018-03-30', # Sexta-feira Santa
    '2018-04-01', # Páscoa
    '2018-04-25', # Dia da Liberdade
    '2018-05-01', # Dia do Trabalhador
    '2018-05-31', # Corpo de Deus
    '2018-06-10', # Dia de Portugal
    '2018-06-24', # São João
    '2018-08-15', # Assunção de Nossa Senhora
    '2018-10-05', # Implantação da República
    '2018-11-01', # Dia de Todos os Santos
    '2018-12-01', # Restauração da Independência
    '2018-12-08', # Imaculada Conceição
    '2018-12-25', # Natal
    '2019-01-01', # Ano Novo
    '2019-04-19', # Sexta-feira Santa
    '2019-04-21', # Páscoa
    '2019-04-25', # Dia da Liberdade
    '2019-05-01', # Dia do Trabalhador
    '2019-06-10', # Dia de Portugal
    '2019-06-20', # Corpo de Deus
    '2019-06-24', # São João
    '2019-08-15', # Assunção de Nossa Senhora
    '2019-10-05', # Implantação da República
    '2019-11-01', # Dia de Todos os Santos
    '2019-12-01', # Restauração da Independência
    '2019-12-08', # Imaculada Conceição
    '2019-12-25', # Natal
    ])
```

Figura 31-Lista de feriados na cidade do Porto

Anexo 3

Modelo	*Arquitetura*	*Hiperparâmetros*	*Score Interno*	*Accuracy Kaggle*
Decision Tree	---------	criterion=entropy max_depth=5 min_samples_leaf=10 min_samples_split= 2	Accuracy: 0.7660	0.82222
Random Forest	---------	max_depth=None max_features=sqrt min_samples_leaf=3 min_samples_split=2 n_estimators=300	Accuracy: 0.8020	0.80666
XGBoost	---------	reg_alpha=0 n_estimators=700 min_child_weight=3 max_depth=3 learning_rate= 0.03	Accuracy: 0.8137	0.83777
Rede neuronal	Duas camadas ocultas com 64 e 32 neurónios respetivamente 9 epochs		Accuracy: 0.7772	0.76000
Stacking	---------	estimators=[("dt", dt), ("rf", rf),("xgb", xgb)] final_estimator= LogisticRegression(class_weight="balanced", random_state=RSEED), cv=5, n_jobs=-1	F1_score: 0.8086	0.82222

Tabela 2- Resumo de resultados dos melhores modelos