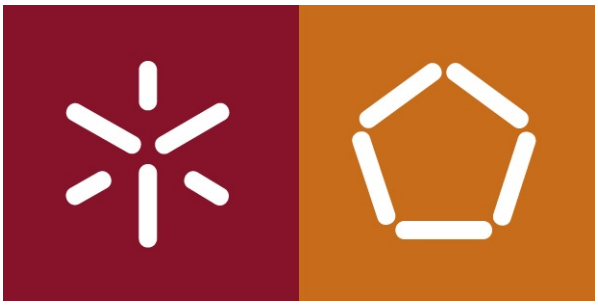
Universidade do Minho

Mestrado em Engenharia Informática



Dados e Aprendizagem Automática

Trabalho Prático de Grupo - Conceção e otimização de modelos de *Machine Learning.*

Grupo: 54	*Número*
Eduardo Teixeira	PG47166
Marco Sampaio	PG47447
Luis Ribeiro	PG46540
João Amorim	A74806

3 de Janeiro de 2022

Conteúdo

1 *Introdução*

Este trabalho prático individual foi desenvolvido no âmbito da cadeira de Dados e Aprendizagem Automática. O projeto tem como objetivo aplicar os conhecimentos lecionados ao longo deste semestre em dois casos de estudo diferentes. Conhecimentos estes desde metadologias de abordagens a problemas, exploração de dados e preparação destes até modelos de decisão e diferentes sistemas de aprendizagem . O primeiro caso passa pelo estudo e desenvolvimento de modelos de *Machine Learning* de um *dataset*, escolhido pelo grupo, de entre os que estão acessíveis a partir de fontes disponibilizados pelos docentes capazes de prever o resultados sobre o problema escolhido. O segundo passa pelo estudo de um *dataset* construído pelos docentes e pelo desenvolvimento de modelos de *Machine Learning* capaz de prever o fluxo de tráfego rodoviário. Assim, será utilizada a linguagem de programação *Python* para o desenvolvimento do conhecimento acima referido, ao longo deste relatório será descrito o processo utilizado para analisar e realizar previsões de resultados em ambos os casos de estudo.

2 Domínio, Objetivos e Metadologia

O projeto é realizado sobre dois casos de estudo diferentes uma vez que vamos analisar dois *datasets* diferentes ainda assim vamos ter o mesmo domínio, o domínio do *Data Science* que se baseia em extrair conhecimento e percepções desses mesmos dados. Embora tenhamos o mesmo domínio vamos possuir objectivos diferentes para os dois casos.

Para o primeiro caso de estudo foi escolhido pelo grupo na plataforma *Kaggle*, o *Dataset of songs in Spotify*. A ideia era realizar um projeto que estivesse relacionado com música, neste caso o problema, de classificação, passa por obter a previsão de géneros musicais consosante dados numéricos. Este géneros já são conhecidos.

Para o segundo caso, vamos analisar um *dataset* dado pelos docentes que contém dados referentes ao tráfego de veículos numa cidade portuguesa durante um período superior a 1 ano. O objectivo passa por extrair conhecimento deste de modo a prever o fluxo de tráfego rodoviário, numa determinada hora, na referida cidade.

Ambos os casos encontram-se sobre um sistema de Aprendizagem com Supervisão.

Referido o domínio e objetivos, para a realização deste projeto foi aplicado uma metadologia para *Knowledge Extraction*, ou seja esta descreve um conjunto de etapas que o desenvolvimento de um projeto de extração de conhecimento deve passar de forma a obter resultados e a atingir objetivos. No nosso caso, para ambos os datasets foi utilizada uma metadologia perto da denominada de **CRISP-DM**.

2.1 CRISP-DM

Business Understanding- Para ambos os casos definimos qual o problemas em questão e os objetivos a cumprir;

Data Understanding- Obtemos acesso aos dados e identificamos a qualidade destes para posteriomente os preparármos;

Data Preparation- Com o uso de bibiliotecas tratamos os dados, por exemplo, tratar dados em falta,incompletos e identificar quais a usar;

Modeling- Escolhemos o modelo de *Machine Learning* a usar e identificamos hiperparâmetros com várias ferramentas;

Evaluation- Comparamos resultados para diferentes modelo e efetuamos as respetivas avaliações;

Deployment- Este último ponto põe em ação os modelos, algo que **não** se realizou.

Todos estes passos são importantes e cruciais para uma interpretação e realização dos objetivos dos nossos casos. Cada um possui o peso para os resultados e previsões. Porém como está referido no último ponto **Deployment** não é realizado visto que refere por em prática cada modelo realizado, algo que neste projeto não acontece.

3 *Dataset of songs in Spotify*

3.1 Business Understanding

Detalhado na secção anterior.

3.2 Data Understanding

De forma a obter os dados e a analisar estes foi utilizada a linguagem *Python* e bibliotecas como *pandas* , *seaborn* e *missingno*. Foi feita então uma identificação da qualidade dos dados. Ou seja, que dados é que possuiam valores em falta, o tipo de cada atributo/coluna e correlações entre atributos.

```
ds_tracks.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42305 entries, 0 to 42304
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   danceability      42305 non-null  float64
 1   energy            42305 non-null  float64
 2   key               42305 non-null  int64
 3   loudness          42305 non-null  float64
 4   mode              42305 non-null  int64
 5   speechiness       42305 non-null  float64
 6   acousticness      42305 non-null  float64
 7   instrumentalness  42305 non-null  float64
 8   liveness          42305 non-null  float64
 9   valence           42305 non-null  float64
 10  tempo             42305 non-null  float64
 11  type              42305 non-null  object
 12  id                42305 non-null  object
 13  uri               42305 non-null  object
 14  track_href        42305 non-null  object
 15  analysis_url      42305 non-null  object
 16  duration_ms       42305 non-null  int64
 17  time_signature    42305 non-null  int64
 18  genre             42305 non-null  object
 19  song_name         21519 non-null  object
 20  Unnamed: 0        20780 non-null  float64
 21  title             20780 non-null  object
dtypes: float64(10), int64(4), object(8)
memory usage: 7.1+ MB
```

(a) Tipos dos atributos/colunas

```
ds_tracks.isna().sum()

danceability            0
energy                  0
key                     0
loudness                0
mode                    0
speechiness             0
acousticness            0
instrumentalness        0
liveness                0
valence                 0
tempo                   0
type                    0
id                      0
uri                     0
track_href              0
analysis_url            0
duration_ms             0
time_signature          0
genre                   0
song_name           20786
Unnamed: 0          21525
title               21525
dtype: int64
```

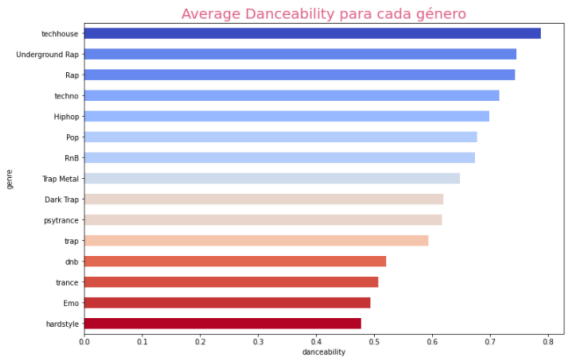
(b) Atributos com dados em falta

Figura 1: Atributos de *Dataset of Songs*

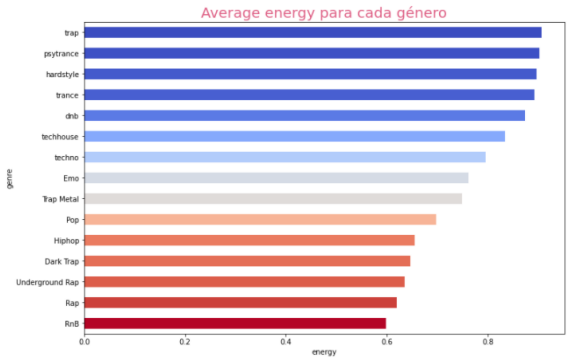
De seguida foi feita uma análise aos atributos como distribuição dos dados, identificado os tipos de género e as relação de alguns atributos com esses géneros. Foi feito também uma análise à correlação entre as colunas.

```
np.unique(ds_tracks['genre'])

array(['Dark Trap', 'Emo', 'Hiphop', 'Pop', 'Rap', 'RnB', 'Trap Metal',
       'Underground Rap', 'dnb', 'hardstyle', 'psytrance', 'techhouse',
       'techno', 'trance', 'trap'], dtype=object)
```

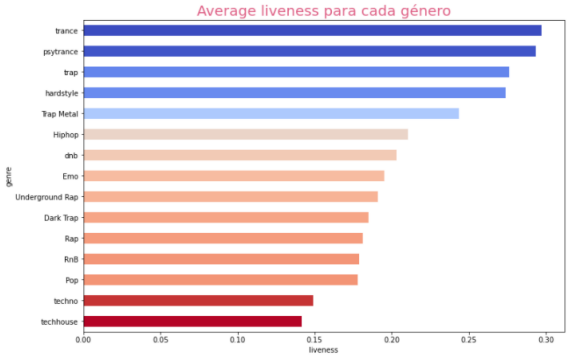
Figura 2: Os vários géneros do *dataset*



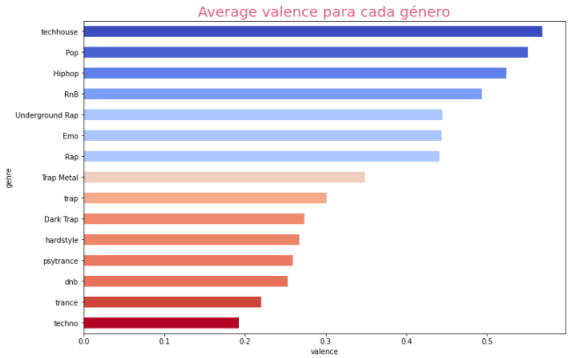
(a) Danceability por género



(b) Energy por género



(c) Liveness por género



(d) Valence por género

Figura 3: Alguns atributos e suas relações com os vários géneros

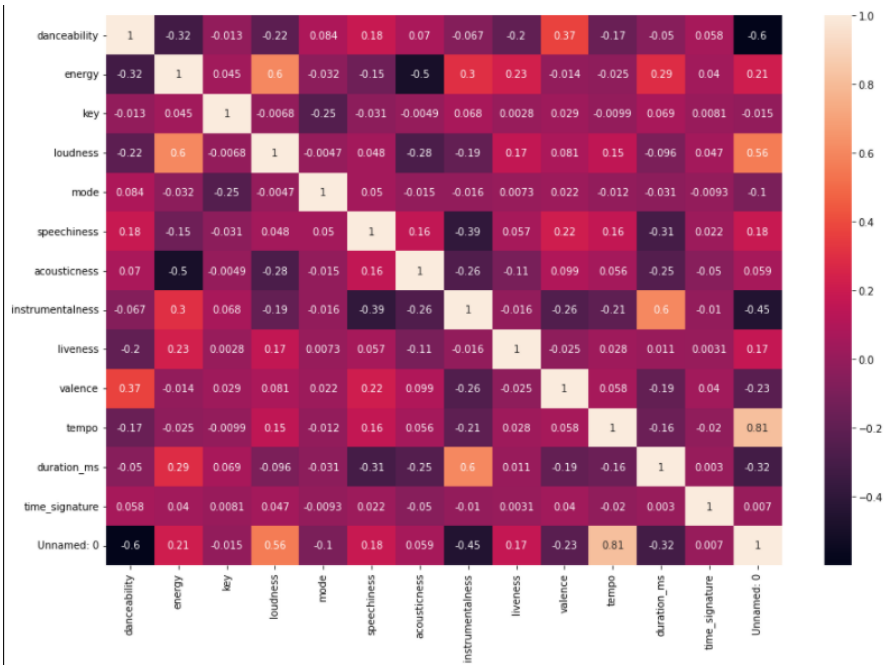


Figura 4: Correlação entre Atributos

3.3 Data Preparation

Na secção anterior foi feita uma análise aos atributos que possuiamos. Visto que grande parte deles estava no melhor tipo para ser treinado (int ou float) apenas foram tratados os valores em falta. Da análise dos dados em falta identificou-se que apenas existiam em campos que não nos interessavam por isso foi dado drop a estes.

```
# Realizamos drop às colunas e atributos não necessários, no nosso caso são maioritariamente colunas sobre links
# titulos de músicas/tracks .
ds_tracks = ds_tracks.drop(["title","Unnamed: 0","song_name","analysis_url","track_href","uri","id","type"], axis=1)
```

Figura 5: Atributos descartados do *dataset*

3.4 Modeling e Evalutation

Para a secção de *Modeling* foi utilizada a biblioteca *scikit-learn*. Nesta etapa foram realizados vários modelos e técnicas de *Machine Learning*: *Logistic Regression* ,*DecisionTree classifiers* e *RandomForest*. Foi utilizada a função de *train_test_split* uma vez que apenas possuiamos um dataset e tinhamos necessidade de separar os dados em treino e teste.

```
x= ds_tracks.drop(['genre'], axis=1)
y = ds_tracks['genre'].to_frame()

X_train,X_test,y_train,y_test=train_test_split(x,y,random_state=1)

log_m = "logistic_reg"
rand_m = "random_forest"
dec_m = "decision_tree"


models={log_m: LogisticRegression(),
        rand_m:RandomForestClassifier(random_state  = 1, max_depth = 10),
        dec_m: DecisionTreeClassifier(max_depth=10, min_samples_split=10, random_state=42) }
```

Figura 6: Modelos

Em ambos os modelos de regressão logística e de de *DecisionTree Classifier* não foi utilizado qualquer técnica de *Tuning*, apenas para *RandomForest*. O nosso X_train é constituidos pelos campos que foram tratados na preparação de dados. O y_train constitui a coluna "genre". Tendo os valores de X_train e y_train é então criado cada modelo. Obtemos previsões ao aplicar o modelo aos valores de X_test.

```
dct = models[dec_m].fit(X_train,y_train)

predictions_dct = dct.predict(X_test)

print(classification_report(y_test,predictions_dct))
```

	precision	recall	f1-score	support
Dark Trap	0.49	0.42	0.46	1158
Emo	0.52	0.53	0.52	401
Hiphop	0.44	0.39	0.41	779
Pop	0.14	0.03	0.04	113
Rap	0.91	0.32	0.47	463
RnB	0.32	0.30	0.31	494
Trap Metal	0.35	0.19	0.25	499
Underground Rap	0.43	0.65	0.52	1511
dnb	0.97	0.93	0.95	766
hardstyle	0.75	0.84	0.79	747
psytrance	0.90	0.90	0.90	720
techhouse	0.80	0.86	0.83	744
techno	0.83	0.78	0.80	732
trance	0.71	0.78	0.74	744
trap	0.75	0.71	0.73	706
accuracy			0.63	10577
macro avg	0.62	0.58	0.58	10577
weighted avg	0.64	0.63	0.62	10577

(a) *DecisionTree Classifier*

```
models[log_m].fit(X_train,y_train)

LogisticRegression()

predictions = models[log_m].predict(X_test)

print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
Dark Trap	0.29	0.01	0.02	1158
Emo	0.00	0.00	0.00	401
Hiphop	0.00	0.00	0.00	779
Pop	0.00	0.00	0.00	113
Rap	0.00	0.00	0.00	463
RnB	0.00	0.00	0.00	494
Trap Metal	0.00	0.00	0.00	499
Underground Rap	0.23	0.85	0.37	1511
dnb	0.17	0.49	0.25	766
hardstyle	0.00	0.00	0.00	747
psytrance	0.34	0.90	0.49	720
techhouse	0.17	0.05	0.07	744
techno	1.00	0.00	0.00	732
trance	0.11	0.10	0.10	744
trap	0.00	0.00	0.00	706
accuracy			0.23	10577
macro avg	0.15	0.16	0.09	10577
weighted avg	0.19	0.23	0.12	10577

(b) Regressão Logistica

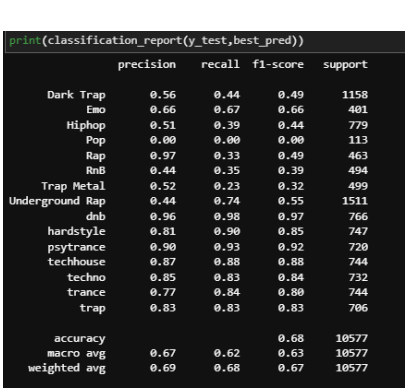
3.4.1 RandomForest Classifier

Para este modelo utilizou-se o *RandomizedSearchCV* para um dicionário com valores de **max_depth** e **n_estimators**, obtendo-se os melhores valores para cada um. Seguidamente comparou-se o modelo dos melhores parâmetros com o modelo base.

```
tunning_dict = { 'max_depth' : [10, 20, 30],
                 'n_estimators': [100,200, 400, 600]
                }

rf_random = RandomizedSearchCV(estimator = models[rand_m], param_distributions = tunning_dict, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)

rf_random.fit(X_train,y_train)

Fitting 3 folds for each of 12 candidates, totalling 36 fits

RandomizedSearchCV(cv=3,
                   estimator=RandomForestClassifier(max_depth=10,
                                                    random_state=1),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'max_depth': [10, 20, 30],
                                        'n_estimators': [100, 200, 400, 600]},
                   random_state=42, verbose=2)

rf_random.best_params_

{'n_estimators': 400, 'max_depth': 10}


rfc = models[rand_m].fit(X_train,y_train)

best_rfc = rf_random.best_estimator_
best_pred = best_rfc.predict(X_test)
```

Figura 8: *RandomizedSearchCV*

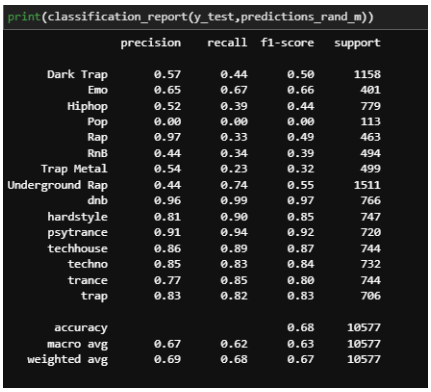
Porém os resultados entre o melhor modelo e o modelo base pouco se alteraram, obtendo o mesmo valor de *accuracy*.



```
print(classification_report(y_test,best_pred))
```

	precision	recall	f1-score	support
Dark Trap	0.56	0.44	0.49	1158
Emo	0.66	0.67	0.66	401
Hiphop	0.51	0.39	0.44	779
Pop	0.00	0.00	0.00	113
Rap	0.97	0.33	0.49	463
RnB	0.44	0.35	0.39	494
Trap Metal	0.52	0.23	0.32	499
Underground Rap	0.44	0.74	0.55	1511
dnb	0.96	0.98	0.97	766
hardstyle	0.81	0.90	0.85	747
psytrance	0.90	0.93	0.92	720
techhouse	0.87	0.88	0.88	744
techno	0.85	0.83	0.84	732
trance	0.77	0.84	0.80	744
trap	0.83	0.83	0.83	706
accuracy			0.68	10577
macro avg	0.67	0.62	0.63	10577
weighted avg	0.69	0.68	0.67	10577

(a) Modelo *best_ estimator_*



```
print(classification_report(y_test,predictions_rand_m))
```

	precision	recall	f1-score	support
Dark Trap	0.57	0.44	0.50	1158
Emo	0.65	0.67	0.66	401
Hiphop	0.52	0.39	0.44	779
Pop	0.00	0.00	0.00	113
Rap	0.97	0.33	0.49	463
RnB	0.44	0.34	0.39	494
Trap Metal	0.54	0.23	0.32	499
Underground Rap	0.44	0.74	0.55	1511
dnb	0.96	0.99	0.97	766
hardstyle	0.81	0.90	0.85	747
psytrance	0.91	0.94	0.92	720
techhouse	0.86	0.89	0.87	744
techno	0.85	0.83	0.84	732
trance	0.77	0.85	0.80	744
trap	0.83	0.82	0.83	706
accuracy			0.68	10577
macro avg	0.67	0.62	0.63	10577
weighted avg	0.69	0.68	0.67	10577

(b) Modelo Base

Figura 9: Comparação entre modelo $best_estimatoremodelobase$

3.5 Evaluation

Na seguinte tabela é possivel identificar e avaliar os modelos consoante a marca de *accuracy* que cada um obteve. De todos os modelos obtivemos melhores resultados ao usar os modelos de *RandomForestClassifier* e *DecisionTree*. Embora o *dataset* esteja um pouco não balanceado foram utilizados modelos capazes de bons resultados visto que os seus métodos obtém valores concretos.

RandomForest obtém melhores valores visto que seleciona aleatoriamente *rows* e características/variáveis específicas para construir várias *DecisionTree* e, em seguida, calcula a média dos resultados, obtendo o melhor.

Model	**Accuracy**
Logistic Regression	0.22917
DecisionTree Classifier	0.6303
RandomForestClassifier	0.6822

4 Previsão do fluxo de tráfego rodoviário

4.1 Business Understanding

É constante e comum ouvirmos falar do congestionamento do tráfego rodoviário e das suas consequências em vários fatores. A modelação deste é então um conhecido e famoso problema de características estocásticas, não lineares. A popularidade deste é demonstrada na quantidade de modelos e do potencial assinalável destes no tipo de previsões de tráfego.

Todos estes fatores considerados e importantes são representados pelo caso em estudo e com isso denotamos o objetivo deste segundo caso do projeto. Visto que este problema passa pela previsão do tráfego estamos perante um problema de **Classificação**. Para a realização deste projeto foram providenciados dois *datasets*, um para treino (treino do modelo na etapa *Modeling*) e outro para teste (obter previsões do modelo em *Modeling*).

Concluimos assim o tópico de **Business Understanding** deste caso de estudo.

4.2 Data Understanding

De forma a obter os dados e a analisar estes foi utilizada a linguagem *Python* e bibliotecas como *pandas* , *seaborn* e *missingno*. Foi feita primeiro uma análise dos campos que que o *dataset* possuia, cerca de 13. Deste 13 valores, 5 eram do tipo *object* algo que, se possível teria de ser tratado para o tipo que pretendíamos, o dos restantes 8, *float*.

```
df2.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   city_name               1500 non-null   object
 1   record_date             1500 non-null   object
 2   AVERAGE_FREE_FLOW_SPEED 1500 non-null   float64
 3   AVERAGE_TIME_DIFF       1500 non-null   float64
 4   AVERAGE_FREE_FLOW_TIME  1500 non-null   float64
 5   LUMINOSITY              1500 non-null   object
 6   AVERAGE_TEMPERATURE     1500 non-null   float64
 7   AVERAGE_ATMOSP_PRESSURE 1500 non-null   float64
 8   AVERAGE_HUMIDITY        1500 non-null   float64
 9   AVERAGE_WIND_SPEED      1500 non-null   float64
 10  AVERAGE_CLOUDINESS      901 non-null    object
 11  AVERAGE_PRECIPITATION   1500 non-null   float64
 12  AVERAGE_RAIN            140 non-null    object
dtypes: float64(8), object(5)
```

Figura 10: Tipos associados a cada coluna do *dataset*

É possível também identificar nesta tabela que possuimos valores não nulos ou nulos/*missing* e a quantidade destes para atributo/coluna. Atributos como "AVERAGE_CLOUDINESS"e "AVERAGE_RAIN"possuem valores em falta ou nulos. Era então questão de verificar se estes dados valiam a pena serem recuperáveis/tratados ou simplesmente descartados.

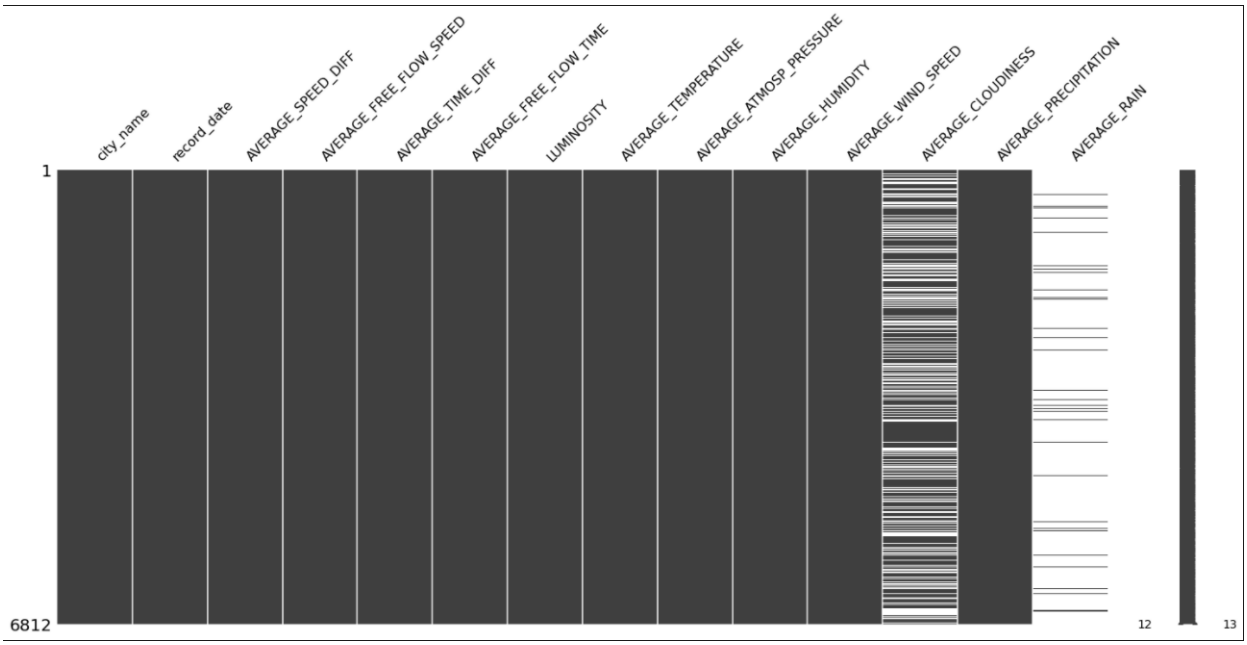


Figura 11: Valores em falta no *dataset*

Após a análise deste gráfico reparou-se que as colunas de "AVERAGE_CLOUDINESS"e "AVERAGE_RAIN"possuiam valores em falta porém podiamos tratar esses dados de forma a tentar completá-los da melhor maneira. Quanto a "AVERAGE_PRECIPITATION"reparou-se que possuia não possuia dados em falta porém todos estes eram nulos, ou seja a coluna inteira tinha valores com valor 0.0 . Estes valores não contribuem para o sucesso de qualquer previsão e deste modo foram descartados na fase posterior.

4.3 Data Preparation

Nesta etapa os dados não tratados foram preparados e processados para que seja possível utilizá-los aquando do treino para diferente modelos de *Machine Learning*. Este tratamento vai desde tratamento de dados do tipo *object*, que não possuem interesse para o nosso caso no estado atual até dados em falta ou então nulos.

Primeiramente identificaram-se colunas com valores do tipo *object* que podiam ser importantes, como os campos "record_date"e "AVERAGE_CLOUDINESS"que são campos influentes da visto da nossa perspetiva.

4.3.1 record_date

Para "record_date"foi usado *Feature Engineering* aplicado a datas, separou-se este em vários campos com valores numéricos ("MONTH", "YEAR", etc) e num campo "DAY_NAME"posteriormente descartado.

```
df.record_date = pd.to_datetime(df.record_date)
df['MONTH'] = df.record_date.dt.month
df['YEAR'] = df.record_date.dt.year
df['DAY'] = df.record_date.dt.day
df['HOUR'] = df.record_date.dt.hour
df['DAY_NAME'] = df.record_date.dt.day_name(locale='pt')
```

Figura 12: Tratamento de "record_date"

4.3.2 AVERAGE_CLOUDINESS

No campo "AVERAGE_CLOUDINESS"não só temos valores do tipo *object* como possuimos valores em falta e repetidos. Ao analisar-se constatou-se que muitos dos valores pareciam significar o mesmo e, deste modo foi dado *replace* desses pelos valores que a equipa considerou representar melhor os dados:

- 'nuvens quebrados' por 'nuvens quebradas'
- 'nublado' por 'tempo nublado'
- 'céu claro' por 'céu limpo'

```
df['AVERAGE_CLOUDINESS'].replace({'nuvens quebrados': 'nuvens quebradas'}, inplace=True)
df['AVERAGE_CLOUDINESS'].replace({'nublado': 'tempo nublado'}, inplace=True)
df['AVERAGE_CLOUDINESS'].replace({'céu claro': 'céu limpo'}, inplace=True)
```

Figura 13: Replace de "AVERAGE_CLOUDINESS"

De seguida foram tratados os dados que faltava. Primeiramente foi se buscar para cada row com valor válido (not null) o seu valor de "AVERAGE _CLOUDINESS"para podermos tratar do valores nulos consoante a data em questão. A ideia passava por realizar um script que guardasse cada dia,mês e ano de uma linha num dicionario como key e no seu value o "AVERAGE_CLOUDINESS"válido. O dicionário era uma mais valia para o tratamento de dados de "AVERAGE_RAIN"visto que do nosso ponto de vista se relacionam nuvens e precipitação. Como alguns valores em falta (NaN) possuiam a mesma data que outras rows/linhas com valores válidos, foram substituidos esse em falta pelos válidos com a data certa guardada no dicionário.

```python
from datetime import datetime, timedelta



date_cloud = {}
for index,row in df[df['AVERAGE_CLOUDINESS'].notnull()].iterrows():
    date = datetime(year=row['YEAR'], month=row['MONTH'], day=row['DAY'])

    if date in date_cloud:
        pass

    else:
        date_cloud[date]=row['AVERAGE_CLOUDINESS']
```

(a) Construção do Dicionário

```python
for index,row in df[df['AVERAGE_CLOUDINESS'].isnull()].iterrows():
    date = datetime(year=row['YEAR'], month=row['MONTH'], day=row['DAY'])
    if date in date_cloud:
        df.at[index,'AVERAGE_CLOUDINESS']=date_cloud[date]
```

(b) Substituição de NaN por válidos do Dicionário

Figura 14: Script para valores válidos de "AVERAGE_CLOUDINESS"

Por último, verificou-se que possuiamos valores de NaN para linhas que não tinham data no dicionário criado. Deste modo, o grupo decidiu complementar essa falta de valores, linha a linha do datafram com valors NaN relacionando com os campos de "AVERAGE_HUMIDITY"e "AVERAGE_TEMPERATURE", ou seja os valores escritos no dataframe estavam dependentes dos destes dois campos.

Assim conseguiamos tratar por completo os valores em falta podendo, no final usar a feature *fit Label Enconder* para transformar os dados categóricos em numéricos.

```python
for index,row in df[df['AVERAGE_CLOUDINESS'].isnull()].iterrows():
    if row['AVERAGE_TEMPERATURE'] >=20 and row['AVERAGE_HUMIDITY'] <50:
        df.at[index,'AVERAGE_CLOUDINESS']= 'céu limpo'
    elif row['AVERAGE_TEMPERATURE'] >=10 and row['AVERAGE_HUMIDITY'] >50:
        df.at[index,'AVERAGE_CLOUDINESS']= 'céu pouco nublado'
    else:
        df.at[index,'AVERAGE_CLOUDINESS']= 'nuvens quebradas'
```

Figura 15: Tratamento de "record_date"

```python
df['AVERAGE_CLOUDINESS'] = le.fit_transform(df['AVERAGE_CLOUDINESS'])
np.unique(df['AVERAGE_CLOUDINESS'])
```

Figura 16: *fit* de dados categóricos em numéricos

4.3.3 AVERAGE_RAIN

Tal como "AVERAGE_CLOUDINESS", o campo "AVERAGE_RAIN"ao ser analisado constatou-se que muitos dos valores repetidos tinham valores que podiam significar a mesma coisa e, deste modo foi dado *replace* desses pelos valores que a equipa considerou representar melhor os dados:

- 'chuva' por 'chuva moderada'
- 'chuva de intensidade pesada' e 'chuva de intensidade pesado' por 'chuva forte'
- 'chuvisco e chuva fraca' por 'chuva fraca'
- 'chuva leve' por 'chuva fraca'
- 'aguaceiros fracos' por aguaceiros

```
df['AVERAGE_RAIN'].replace({'chuva': 'chuva moderada'}, inplace=True)
df['AVERAGE_RAIN'].replace({'chuva de intensidade pesado': 'chuva forte'}, inplace=True)
df['AVERAGE_RAIN'].replace({'chuva de intensidade pesada': 'chuva forte'}, inplace=True)
df['AVERAGE_RAIN'].replace({'chuvisco e chuva fraca': 'chuva fraca'}, inplace=True)
df['AVERAGE_RAIN'].replace({'chuva leve': 'chuva fraca'}, inplace=True)
df['AVERAGE_RAIN'].replace({'aguaceiros fracos': 'aguaceiros'}, inplace=True)
```

Figura 17: Replace de "AVERAGE_RAIN"

Em seguida foi aplicada a mesma metadologia feita sobre "AVERAGE_CLOUDINESS", ou seja foi criado um dicionário que possuia cada dia,mês e ano de uma linha como key e no seu value o "AVERAGE_RAIN"válido. Seguindo os passos anteriores identicamos as linhas que possuem valor nulo em average rain mas, que pertencem a um dia que já possui valor.

```
date_rain = {}
for index,row in df[df['AVERAGE_RAIN'].notnull()].iterrows():
    date = datetime(year=row['YEAR'], month=row['MONTH'], day=row['DAY'])

    if date in date_rain:
        pass

    else:
        date_rain[date]=row['AVERAGE_RAIN']
```

(a) Construção do Dicionário

```
for index,row in df[df['AVERAGE_RAIN'].isnull()].iterrows():
    date = datetime(year=row['YEAR'], month=row['MONTH'], day=row['DAY'])
    if date in date_rain:
        df.at[index,'AVERAGE_RAIN']=date_rain[date]
```

(b) Substituição de NaN por válidos do Dicionário

Figura 18: Script para valores válidos de "AVERAGE_CLOUDINESS"

Visto que esta solução não resolveu totalmente os problemas pois ainda possuimos valores NaN o grupo decidiu completar essa falta de valores consoante valores de "AVERAGE_HUMIDITY"e de "AVERAGE_CLOUDINESS". Depois de uma análise de relações entre estes 3 campos.

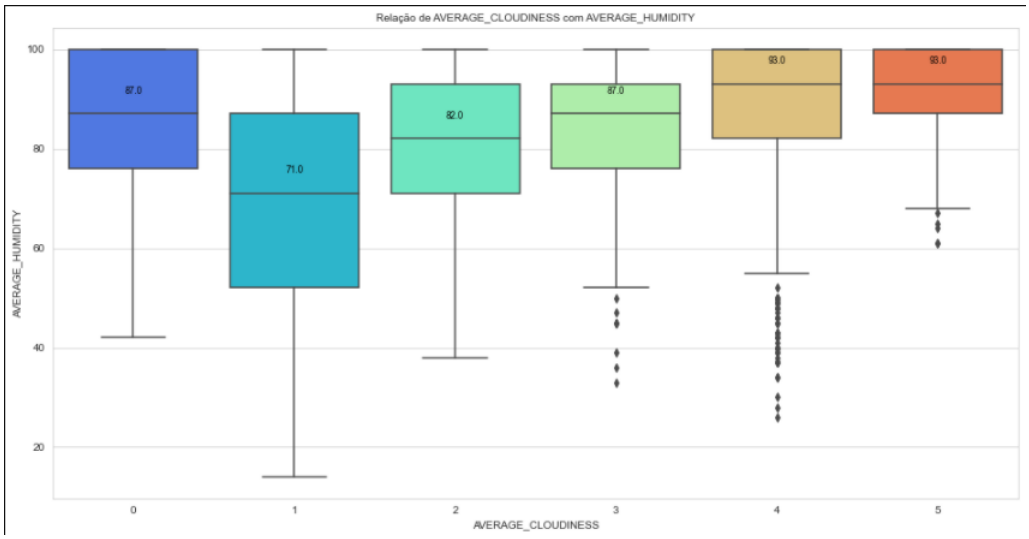


Figura 19: Relação entre "AVERAGE_CLOUDINESS"e "AVERAGE_HUMIDITY"

Labels de AVERAGE_CLOUDINESS :

0 - 'algumas nuvens'

1 - 'céu limpo'

2 - 'céu pouco nublado'

3 - 'nuvens dispersas'

4 - 'nuvens quebradas'

5 - 'tempo nublado'

Tipos de AVERAGE_RAIN :

- 'aguaceiros'
- 'chuva forte'
- 'chuva fraca'
- 'chuva moderada'
- 'chuvisco fraco'
- 'trovoada com chuva'
- 'trovoada com chuva leve'
- 'Sem chuva'*

* A variável 'sem chuva' foi adicionada pois o grupo entendeu não existir um valor de "AVERAGE_RAIN"influenciada por um tipo do campo de "AVERAGE_CLOUDINESS-'céu limpo'

Depois da análise do boxplot foi desenvolvido um script que completava os valores em falta de "AVERAGE_RAIN"influencidados por "CLOUDINESS"e "HUMIDITY".

```
for index, row in df[df['AVERAGE_RAIN'].isnull()].iterrows():

    if row['AVERAGE_CLOUDINESS'] == 0: #'algumas nuvens'
        if row['AVERAGE_HUMIDITY'] >= 87:
            df.at[index,'AVERAGE_RAIN']='chuva fraca'

        else:
            df.at[index,'AVERAGE_RAIN']='chuvisco fraco'

    if row['AVERAGE_CLOUDINESS'] == 1: #'céu limpo'
        if row['AVERAGE_HUMIDITY'] >= 71:
            df.at[index,'AVERAGE_RAIN']='chuvisco fraco'

        else:
            df.at[index,'AVERAGE_RAIN']='Sem chuva'

    if row['AVERAGE_CLOUDINESS'] == 2: #'céu pouco nublado'
        if row['AVERAGE_HUMIDITY'] >= 82:
            df.at[index,'AVERAGE_RAIN']='chuva moderada'

        else:
            df.at[index,'AVERAGE_RAIN']='chuva fraca'


    if row['AVERAGE_CLOUDINESS'] == 3: #'nuvens dispersas'
        if row['AVERAGE_HUMIDITY'] >=87:
            df.at[index,'AVERAGE_RAIN']='chuvisco fraco'

        else:
            df.at[index,'AVERAGE_RAIN']='Sem chuva'

    if row['AVERAGE_CLOUDINESS'] == 4: #'nuvens quebradas'
        if row['AVERAGE_HUMIDITY'] >= 87:
            df.at[index,'AVERAGE_RAIN']='chuvisco fraco'

        else:
            df.at[index,'AVERAGE_RAIN']='chuva fraca'

    if row['AVERAGE_CLOUDINESS'] == 5: #''tempo nublado'
        if row['AVERAGE_HUMIDITY'] >= 93:
            df.at[index,'AVERAGE_RAIN']='trovoada com chuva'

        else:
            df.at[index,'AVERAGE_RAIN']='chuva forte'
```

Figura 20: Tratamento de "AVERAGE_RAIN"

4.3.4 Dados descartados

Campos como "city_name"e "LUMINOSITY"foram descartados visto que, para o grupo não constituiam importância suficiente para os modelos a construir. Aos campos "record_date", "DAY_NAME"e "AVERAGE_PRECIPITATION"foi dado *drop* visto que: o primeiro foi tratado e referido anteriormente, o segundo foi criado a partir de "record_date"porém era uma variável categórica não preponderante e no terceiro os (explicado anteriormente em *Data Understanding*), os valores do campo serem totalmente nulos.

4.4 Modeling

Para a secção de *Modeling* foi utilizada a biblioteca *scikit-learn*. Nesta etapa foram realizados vários modelos e técnicas de *Machine Learning*: *Logistic Regression* ,*Support Vector Machine*,*Xgboost* e *RandomForest*, porém não foram usadas técnicas de *Tuning* de todas. Nos

modelos em que foi utilizado, a obtenção de melhores parâmetros foi realizado através de *GridSearch*.

A ideia desta etapa passar por treinar um modelo específico com o *dataset* de treino e fazer previsões com o *dataset* de teste. Ao contrário do que se realizou em algumas aulas, não foi utilizada a função de *train_ test_ split* uma vez que não tinhamos necessidades de separar os dados em teste e treino.

4.4.1 Logistic Regression

No modelo de regressão logística não foi utilizado qualquer técnica de *Tuning*. Para treinar este modelo foi utilizado 100% do *dataset* de training. O nosso X_train é constituidos pelos campos que foram tratados na preparação de dados. O y_train constitui a coluna que é estudada "AVERAGE_SPEED_DIFF". Tendo os valores de X_train e y_train é então criado o modelo. Obtemos previsões ao aplicar o modelo ao nosso *dataset* de teste (este também previamente tratado com a metadologia anterior).

```
logistic_model = LogisticRegression()
logistic_model.fit(X_train,y_train)
predictions = logistic_model.predict(df2)
```

Figura 21: Modelo e predict de *Logistic Regression*

4.4.2 Support Vector Machine

No modelo de *Support Vector Machine* de problemas de classificação, o SVC, para treinar este foi utilizado 100% do *dataset* de training. O nosso X_train é constituidos pelos campos que foram tratados na preparação de dados. O y_train constitui a coluna que é estudada "AVERAGE_SPEED_DIFF". Tendo os valores de X_train e y_train é então criado o modelo. Obtemos previsões ao aplicar o modelo ao nosso *dataset* de teste (este também previamente tratado com a metadologia anterior). Para esta técnica foi utilizado *GridSearch* técnica de *Tuning* para obter os melhores parâmetros nos campos $\boldsymbol{C}$, ***Gamma*** e ***kernel***.

```
from sklearn.model_selection import GridSearchCV
```

```
pwr = [10.0**(i/3.0) for i in range(-8,9)]
pwr
```

```
[0.0021544346900318843,
 0.004641588833612777,
 0.01,
 0.021544346900318832,
 0.046415888336127795,
 0.1,
 0.2154434690031884,
 0.4641588833612779,
 1.0,
 2.154434690031884,
 4.641588833612778,
 10.0,
 21.544346900318832,
 46.4158883361278,
 100.0,
 215.44346900318845,
 464.15888336127773]
```

```
param_grid = {'C': [0.001,0.01,0.1,1,10], 'gamma':[1,0.1,0.01,0.001,0.0001],'kernel':['rbf']}
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=0)
```

```
grid.fit(X_train,y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})
```

```
grid.best_params_
```

```
{'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
grid.best_estimator_
```

```
SVC(C=10, gamma=0.0001)
```

Figura 22: Melhores hyperparâmetros de SVC para X_train e y_train

```
svc_model=SVC(C=10, gamma=0.0001)
```

```
#Training and Predicting
svc_model.fit(X_train,y_train)
```

```
SVC(C=10, gamma=0.0001)
```

```
predictions = svc_model.predict(df2)
```

Figura 23: Modelo e predict de SVC

4.4.3 Xgboost

No modelo de *Xgboost* de problemas de classificação, para treinar este foi utilizado 100% do *dataset* de training. O nosso X_train é constituidos pelos campos que foram tratados na preparação de dados. O y_train constitui a coluna que é estudada "AVERAGE_SPEED_DIFF". Tendo os valores de X_train e y_train é então criado o modelo. Obtemos previsões ao aplicar o modelo ao nosso *dataset* de teste (este também previamente tratado com a metadologia anterior). Para esta técnica foi utilizado *GridSearch* técnica de *Tuning* para obter os melhores parâmetros nos campos ***n_estimators*** e ***max_depth***.

```
clf = GridSearchCV(xgb_model,
                   {'max_depth': [2,4,6],
                    'n_estimators': [50,100,200]}, verbose=1)

clf.fit(X_train,y_train)
print(clf.best_score_)
print(clf.best_params_)
```

Figura 24: Melhores hyperparâmetros de XGBClassifier para X_train e y_train

```
{'max_depth': 2, 'n_estimators': 100}

: xgb_model = xgb.XGBClassifier(max_depth=2,n_estimators=100)

: xgb_model.fit(X_train,y_train)

[17:28:28] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1
115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:soft
prob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.

: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=2, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, objective='multi:softprob', predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)

: xgb_predictions = xgb_model.predict(df2)
```

Figura 25: Modelo e predict de Xgboost

Foram realizados mais modelos de diferentes técnicas, como *DecisionTrees* e *RandomForest* porém estes são aqueles que receberam mais detalhe e preparação do grupo de trabalho. Ao realizar o *tuning* de melhores hiper parâmetros para o modelo de *RandomForest* o resultado não foi o melhor.

4.5 Evaluation

Nesta etapa comparamos os *scores* obtidos nas diferentes submissões de diferentes modelos. É de destacar que a primeira submissão foi feita com uma preparação de dados diferente da explicada e com um Modelo de *DecisionTrees*. A seguinte tabela demonstra as melhores submissões/tentativas do grupo.

Model	**Score**
Logistic Regression	0.76444
Support Vector Classifier	0.80666
XgboostClassifier	0.76000
RandomForestClassifier	0.74444

É possivel identificar o melhor resultado como sendo o modelo de *Support Vector Classifier*.

O grupo tentou realizar modelos usando Redes Neuronais Artificiais porém os valores de erro eram bastante grandes para os parâmetros de *tuning* que se estava a fornecer. Depois de várias tentativas falhadas decidiu-se não apostar nesta técnica.

DESENVOLVER MAIS

5 *Conclusões, Dificuldades e Sugestões*

No caso de estudo do *Dataset of Songs in Spotify* foi procurado sempre o estudo de várias técnicas em todas as etapas da metadologia. O estudo incluiu bastantes todo o tipo de resoluções, incluido alguma na plataforma *kaggle*. Porém os resultados ficam um bocado àquem do esperado.

Para o caso de estudo de Previsão do fluxo de tráfego rodoviário o grupo assimilou várias ideias do que realmente requer para se tratar um problema de *Machine Learning*, a etapa de preparação de dados é de altíssima importância visto que caso estes não estejam bem tratados e organizados, não interessa o tipo de modelo ou técnica a seguir. Na etapa de *Modeling* fica atravessado a tentativa de utilização de Redes Neuronais Artificiais visto que o grupo acreditava que com esta técnica se podiam ter valores melhores aos obtidos. O facto de o grupo não possuir muito conhecimento acerca de abordagens de problemas de classificação com redes pode ter contribuído, tal como o manuseamento de ferramentas do *tensorflow*. Porém acaba por ser uma das dificuldades deste projeto.

Foram aplicados bastantes conhecimentos, tanto a nível teórico como prático para a realização deste caso. O grupo acaba por ficar contente com alguns resultados mas com a ideia de que poderiam ter sido melhores, caso tivessem sido estudados com mais detalhe alguns aspetos.

6 *Referências*

Para o desenvolvimento deste trabalho foi utilizado como referência os conhecimentos adquiridos durante as aulas teóricas e praticas da UC, bem como todos os materiais disponibilizados na blackboard tal como slides e apontamentos.