ISCTE-IUL
Licenciatura em Ciência de Dados

Previsão do desempenho de estudantes ao jogar

Trabalho realizado no âmbito da Unidade Curricular Processamento de Big Data do 2º ano em 2022/2023 da Licenciatura em Ciência de Dados

André Plancha, 105289
Andre_Plancha@iscte-iul.pt
Allan Kardec Rodrigues, 103380
aksrs@iscte-iul.pt

9 Abril 2023
Versão 1.0.0

Introdução

Jo Wilder and the Capitol Case é um jogo educacional sobre a história de Wisconsin, direcionado a crianças com idades entre os 8 e os 12 anos. O jogo de aventura e investigação segue a história de Jo Wilder, que descobre histórias sobre artefactos do passado do estado, investigando objetos, e encontrando pessoas. O jogo pode ser jogado no site oficial do PBD Wisconsin Education[1]. Esta competição está a ser promovida pela plataforma kaggle[2].

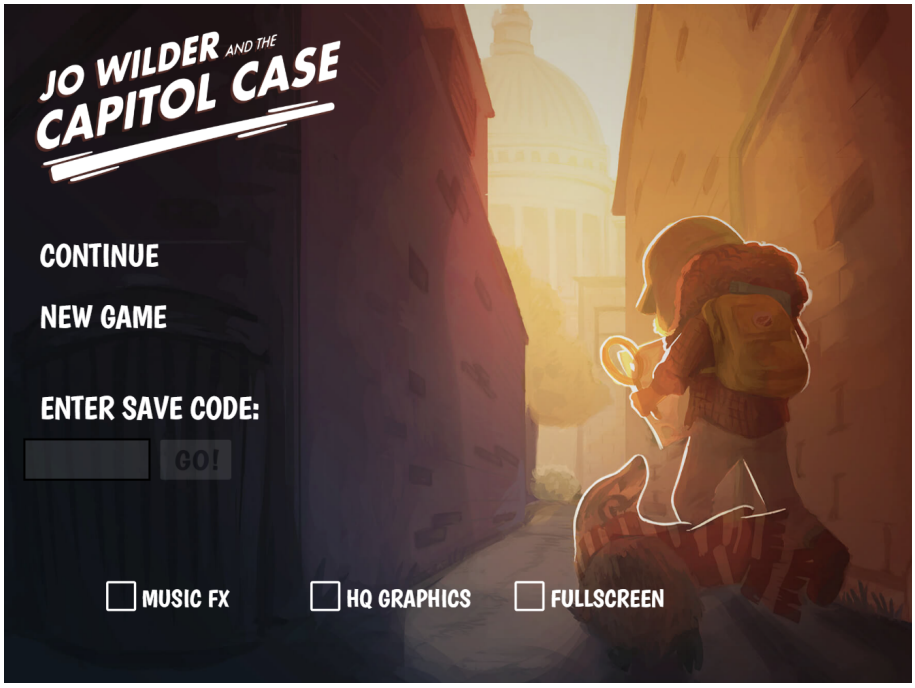

Figura 1: Imagem do menu principal

Este projeto tem como objetivo prever como os jogadores vão responder às 18 questões que o jogo apresenta, baseado na sua atividade durante o tal, implementando uma solução computacional para estudo e análise de dados de grande dimensão. Para isso, vamos usar Apache Spark™e *PySpark* para processar os dados e a biblioteca *MLlib* para a criação de um modelo de aprendizagem supervisionada.

Análise exploratória dos dados

A competição disponibiliza um conjunto de ficheiros de dados, nos quais 2 são de interesse: `train.csv` e `train_labels.csv`, elas tendo as variáveis de interesse e as respetivas respostas, respetivamente. `train.csv` contém as seguintes colunas[3], entre outras:

- `session_id`: Identificador da sessão do evento.
- `index`: Índice do evento na sessão.
- `elapsed_time`: Tempo decorrido desde o início da sessão em ms.
- `event_name`: Tipo do evento.
- `name`: Específicos do tipo de evento.

[1] `pbswisconsineducation.org/jowilder/play-the-game/`
[2] `https://www.kaggle.com/competitions/predict-student-performance-from-game-play`
[3] Mais informações sobre cada coluna está no site da competição, e no EDA em anexo

- `hover_duration`: O tempo de permanência do cursor sobre um objeto, em ms, se aplicável.
- `text`: O diálogo/monólogo que o jogador viu no evento, se aplicável
- `fqid`: O identificador único do evento.
- `fullscreen`: Se o jogador está em modo de ecrã inteiro.
- `hq`: Se o jogo está em alta qualidade.
- `music`: Se a música do jogo está ligada.
- `level_group`: O grupo de níveis a que o evento pertence.

; e `train_labels.csv` contém as seguintes colunas:

- `session_id`: Identificador da sessão do evento, em conjunto da pergunta que se pretende responder.
- `correct`: Se a pergunta está correta ou não.

Perante as colunas, como há uma incompatibilidade entre perguntas e respostas, nós decidimos criar tabela com cada sessão, características de tal (recursos) e as suas respostas, de forma usar classificação para prever as respostas.

Para criar estes recursos, foi necessário de uma análise dos tipos de eventos disponíveis. Uma análise mais aprofundada encontra-se em anexo, mas para resumir, os eventos estão divididos em 3 categorias: Exploração, exposição e revisao. Exploração são os eventos onde a personagem se move, interage com objetos opcionais, tenta entrar em zonas ainda não acedidadas, etc... Exposição são os eventos onde a personagem interage com personagens, interage com objetivos principais, e é onde se encontra o que o utilizador aprende. Revisão são os eventos de quando o utilizador está perdido e portanto precisa de apoio de qual é o próximo passo.

A única excepção a estes eventos é o evento `checkpoint`, representando quando o utilizador começa a responder às perguntas (os eventos das respostas às perguntas não se encontra na tabela). Há 3 zonas onde o utilizador responde às perguntas, e na base de dados as 3 zonas são divididas pelos `checkpoints`, e cada zona está rotulada pelo seu `level_group`.

Análise e limpeza

Nós usámos como base de limpeza a nossa análise, e a análise de erros de um utilizador do kaggle[4]. Na Tabela 1 encontra-se os problemas encontrados e as soluções que nós usámos para os resolver.

Algumas das limpezas foram úteis na criação dos recursos, enquanto que outras apenas limparam a base de dados. Nós não fizemos (nem registámos) algumas das limpezas, pois não seria benéfico para a criação dos recursos necessários.

Algo importante a notar também é que os roteiros dos jogos são diferentes para cada utilizador, entre 4 roteiros diferentes: *dry*, *nohumor*, *nosnark*, e *normal*. Um exemplo pode ser encontrado na Figura 2. A diferença entre roteiros é mínima na opinião dos autores, mas decidimos usar esta informação na mesma.

[4] `https://www.kaggle.com/code/abaojiang/eda-on-game-progress`

Tabela 1: Resolução dos problemas

Erros	Problemas	Solução	Notas
Tempo recorrido recuáva no tempo	Cálculo de tempos entre eventos pode ficar inválido	Transformar em diferenças entre os tempos e tornar zero quando negativo	Máximo desta coluna continua afetado
Saltos de `index`	O índice do evento não é sequencial	-	Suspeitamos que os eventos aconteceram na mesma, só não foram anotados
Sessões com menos ou mais de 3 checkpoints	As sessões são inválidas e dificultam análise	Foram removidas essas sessões	
Tempos de jogo demasiado elevados comparado com a maioria dos tempos	Dificulta o processo de aprendizagem	Tempos de jogo foram transformados e normalizados	

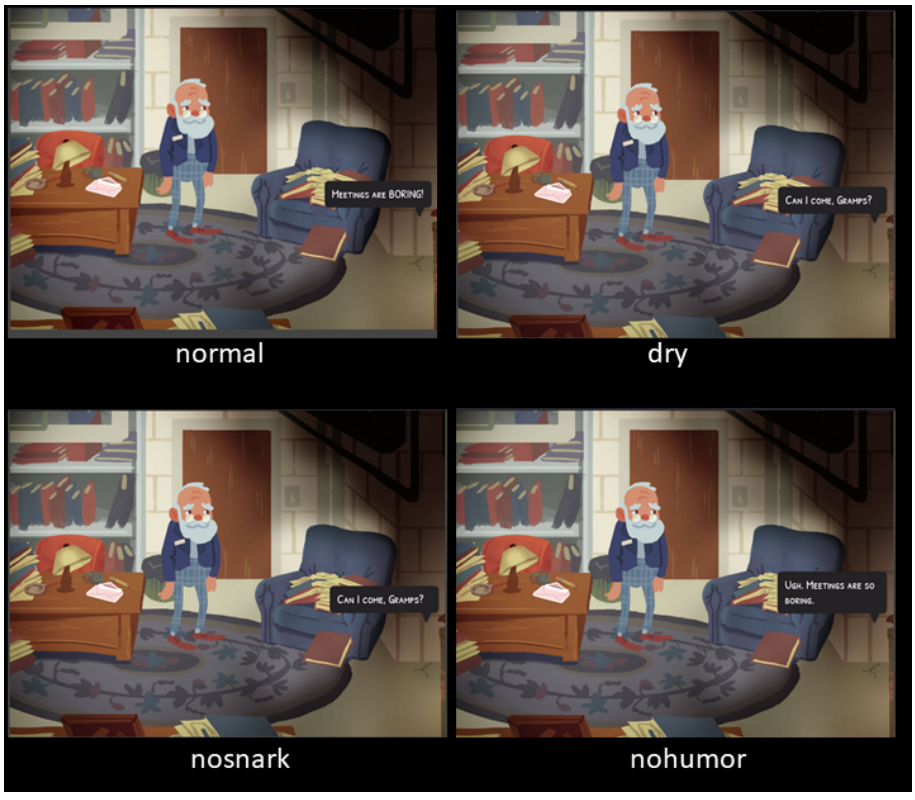

Figura 2: Diferenças entre roteiros

Processo

O processo da implementação da solução computacional pode ser descrito como caótico, sendo que como os dados não vieram com uma estrutura clara, foi necessário uma representação dos dados adaptada previamente ao análise de relações entre os dados; ou seja a criação de recursos teóricos foi executada *à priori*, e a análise de colinearidade e de relação foi feita *ad hoc*. Por outras palavras, a análise de dados foi feita ao longo do processo de criação de recursos, e a criação destes foram feitos apenas com análises comportamentais potências invés de análises estatísticas. Ainda assim, a Figura 3 pode ser visto o processo que foi seguido.

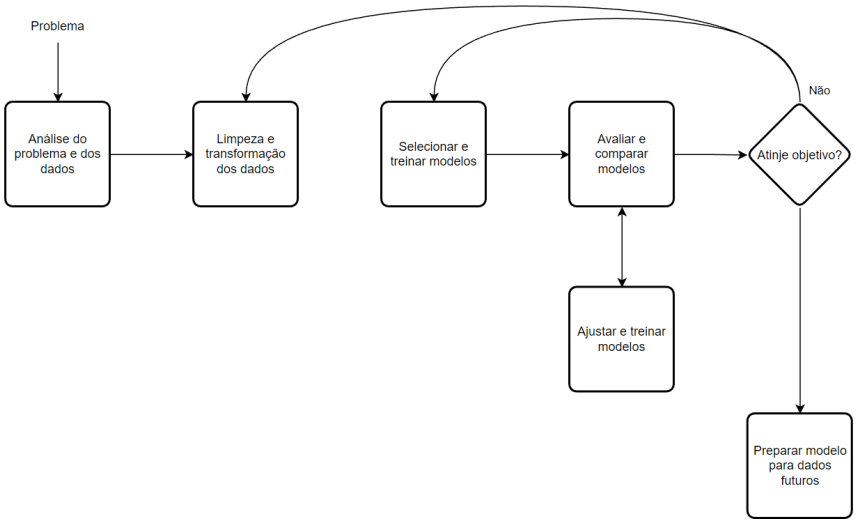

Figura 3: Processo

Recursos

As features que nós fizemos para cada sessão são entre as seguintes:

- O index máximo, que indica o número de eventos de cada sessão;
- Se a sessão esteve em tela cheia;
- Se a sessão estava em alta qualidade;
- Se a sessão tinha música ligada;
- Quantos objetivos opcionais a sessão interagiu com;
- Quantas salas inacessíveis a sessão tentou entrar em;
- Quantas vezes a sessão reviu o objetivo;
- Tempo médio de leitura do diálogo;
- Tempo médio por movimento do jogador;
- Tipo de roteiro.

Os promenores de como foram criadas encontram-se em anexo.

Exploração

TODO

Modelação

Como processo de modelação, nós usámos aprendizagem supervisionada de classificação multi-rótulo, usando como rótulos cada pergunta certa, ou seja, cada sessão podia ter entre 0 e 18 rótulos. Nós considerámos várias opções de classificação, mas decidimos que a melhor forma para o nosso projeto foi com uma transformação para um problema de classificação binária, usando o método de *binary relevance*.

De forma simplificada, para cada rótulo, um modelo binário é treinado, independente dos outros rótulo; se 1 o rótulo é incluido, se 0 não. Esta estratégia fazia sentido, sendo que os nossos rótulo já são binários, o que facilita a modelação, e como temos uma grande quantidade de rótulos, outras estratégias (como *Label Powerset*) não faziam tanto sentido. Ainda assim, pode haver outros métodos mais adequados, como por exemplo *Classifier Chains*, sendo que não deve haver uma indepenência de rótulos.

Para a análise dos nossos modelos, usámos métricas de classificação multi-rótulo, como o *subset accuracy*, *hamming loss*, e *micro-F1*. Usámos também a métrica de avaliação *F1*, estrapulada e adaptada.

Subset accuracy é uma métrica que mede a percentagem de resultados que todos os rótulos estão certos. Previmos que esta métrica não seira muito útil, sendo que 18 rótulos corretos é muito difícil do nosso modelo conseguir, especialmente com o método de *binary relevance*.

$$\uparrow subset\text{-}accuracy = \frac{1}{|Y|}\sum_{i=1}^{|Y|}\left[\hat{Y}_i = Y_i\right]$$

Hamming loss é uma métrica que mede a percentagem de rótulos que estão errados. Sendo que este mede de forma mais individual, esta métrica é mais útil e direta.

$$\downarrow hamming\text{-}loss = \frac{1}{|Y|\cdot|L|}\sum_{i=1}^{|Y|}\sum_{l=1}^{|L|}\left[\hat{Y}_{i,l} \neq Y_{i,l}\right]$$

Micro-F1 é uma métrica que mede a percentagem de rótulos que estão certos, mas penaliza mais os rótulos que estão errados. Esta métrica é útil mais útil para comparar modelos, em vez de ser útil para avaliar o modelo individualmente.

$$\uparrow micro\text{-}F_1 = \frac{2TP}{2TP + FP + FN}$$

F1 é uma métrica usada para avaliar modelos de classificação binária, e é uma média ponderada entre a precisão e a revocação. A nossa métrica adaptada faz a média dos vários F1 scores de cada modelo que compoz o algoritmo. Esta métrica é também útil para comparar modelos, pois dá igual importância a cada modelo, enquanto que a mética anterior tem em conta a quantidade de rótulos que cada X tem, no nosso caso isso equivaler às respostas certas, o que pode induzir em erro a métrica.

$$\uparrow F_1 = \frac{2}{|Y|}\sum_{i=1}^{|Y|}\frac{\left|Y_i \cap \hat{Y}_i\right|}{|Y_i| + |\hat{Y}_i|}$$

Nós começámos por treinar 4 modelos, de forma a compará-los e decidir que método usar, sendo eles regressão logística, máquna de vetores de suporte linear, árvore de decição e *random forest*, todos eles com o método de *binary relevance*. Os promenores do executado e dos parametros estão em anexo. Os resultados foram os seguintes:

Tabela 2: Resultados dos modelos

Modelo	Subset accuracy ↑	Hamming loss ↓	Micro-F1 ↑	F1 ↑
Regressão Logística	0.0222	0.2477	0.8412	0.8239
Máquina de Vetores de Suporte Linear	0.0229	0.2517	0.8424	0.8285
Árvore de Decisão	**0.0237**	**0.2440**	0.8429	0.8268
Random Forest	0.0232	0.2442	**0.8444**	**0.8295**

Como se pode observar, estes modelos não mostram muita diferença nos resultados, mesmo que a árvore de decisão tenha um resultado melhor nas métricas individuais, e o Random Forest tenha obtido um melhor resultado nas métricas de comparação. Decidimos tentar melhorá-los, tirando proveito destes resultados. Logo, treiná-mos mais 4 modelos *Random Forest*, com parâmetros diferentes, especicamente na estratégia de divisão em cada nó. Mais promenores sobre este processo estão na documentação da classe *RandomForest* [5]. Os resultados foram os seguintes:

Tabela 3: Resultados dos modelos 2

Modelo	Subset accuracy ↑	Hamming loss ↓	Micro-F1 ↑	F1 ↑
RF - all	**0.0247**	**0.2420**	**0.84481**	0.8288
RF - sqrt/log2	0.0232	0.2442	0.84442	0.8295
RF - onethird	0.0242	0.2435	0.84480	**0.8298**

Como se pode observar, as diferenças são minúsculas o suficiente que dificilmente se considera um melhor que o outro. Ainda assim, o modelo *RF* com estratégia *all* tem um resultado individual melhor, e *RF* com estratégia *onethird* tem um resultado de comparação melhor. Como os resultados não parecem melhorar tanto assim, e como temos já bons resultados com os modelos apresentados, decidimos não continuar a modelar.

Pela a análise das matrizes de confusão (em anexo), podemos observar muitas das perguntas sem positivos ou negativos. Isto é devido à própria amosra que tinhamos acesso não ter muitas respostas positivas ou negativas para as perguntas em quastão. Isto devia ter sido verificado antes de começar, e devia ter-se adaptado a forma de divisão em conjunto de treino ou teste, ou devia ter-se usado validação cruzada.

```
[[    0,   392],
 [    0, 18311]],

[[   10,  1190],
 [    0, 17503]],
```

Figura 4: exemplos de matrizes de confusão resultadas por rótulo

[5] `https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.mllib.tree.RandomForest.html`