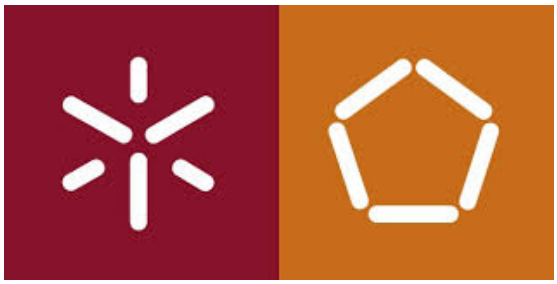

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Aprendizagem e Extração de Conhecimento

Extração de Conhecimento

Carlos José Gomes Campos a74745
José Pedro Ferreira Oliveira a78806
Ludgero da Silva Diogo pg38417
Luis Mendes a57754

5 de Dezembro de 2018

Resumo

Este documento descreve todas etapas da implementação do modelo de extração de conhecimento para prever o estado de stress de um determinado indivíduo, desenvolvido no âmbito da unidade curricular de Aprendizagem e Extração de Conhecimento, do perfil de Sistemas Inteligentes, do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

1 Introdução

Com este relatório pretendemos demonstrar o resultado da investigação e aplicação de conhecimentos, na implementação de um modelo de aprendizagem, focando em especial a parte da extração de conhecimento.

O problema relatado no contexto do desenvolvimento deste projeto prático é o do aumento de casos de esgotamento mental na sociedade, e como a utilização das técnicas de extração de conhecimento e aprendizagem são uma metodologia atrativa para o controlo dos níveis de stress em indivíduos de forma não intrusiva.

Neste trabalho pretende-se desenvolver um modelo de aprendizagem, utilizando o ambiente de desenvolvimento Python/Sklearn, tendo por base a análise de um conjunto de dados biométricos, captados ao longo da execução de exames académicos, consiga prever o nível de stress em que o aluno se encontra.

A estrutura do relatório rege-se pelas etapas que compõem o processo de extração de conhecimento. Inicialmente a na secção 2 será feita uma abordagem acerca de como foram obtidos os dados, neste caso bastante resumida, pois o dataset foi-nos fornecido. Na secção 3 será feita a representação das técnicas utilizadas para a exploração dos valores do dataset fornecido, na secção 4 serão relatados os processos de tratamento dos dados utilizados para corrigir situações críticas analisadas na secção anterior. Em 5 serão abordados os diferentes modelos de aprendizagem testados, bem como os resultados obtidos em cada um deles. Na secção 6 serão eleitos os modelos mais competentes para o problema em questão, juntamente com a respetiva justificação. Na secção 7 é descrito o processo de otimização por hiper parametrização utilizado nos modelos finais, bem como a validade da sua utilização na versão final. O relatório termina com as conclusões na secção 8 onde é feita uma reflexão acerca do trabalho realizado, bem como do trabalho futuro.

2 Aquisição dos dados

O dataset que nos foi facultado contém um conjunto de features biométricas referentes às tomadas de decisão dos estudantes universitários, obtidas no momento da realização de provas académicas.

As features presentes neste conjunto de dados são:

- *ExamID* - corresponde ao identificador do exame que estava a ser efetuado na altura da recolha dos dados
- *FinalGrade* - nota obtida pelo aluno
- *PSS_ Stress* - nível de stress a prever
- *TotalQuestions* - número total de questões do exame
- *avg_ durationperquestion* - tempo médio de resposta a uma pergunta do exame
- *avg_ tbd* (Average Time Between Decision) - tempo médio que cada aluno apresenta para tomar uma decisão
- *decision_ time_ efficiency* - tempo que um aluno demora a tomar uma resposta
- *good_ decision_ time_ efficiency* - tempo que um aluno demora a tomar uma resposta correta
- *maxduration* - duração máxima
- *median_ tbd* (Median Time Between Decision) - mediana do tempo que cada aluno demora a tomar uma decisão
- *minduration* - duração mínima
- *num_ decisions_ made* - número de decisões feitas
- *question_ enter_ count* - número de resposta submetidas
- *ratio_ decisions* - rácio entre o número de respostas dadas e número total de ações
- *ratio_ good_ decisions* - rácio entre número de decisões consideradas corretas e o número de respostas
- *totalduration* - duração total
- *variance_ tdb* - Desvio padrão do tempo entre as decisões para cada aluno

Após a análise de cada uma destas features verificamos que ExameID é uma variável categórica (o seu valor é representativo e identifica uma classe) e as restantes são quantitativas (representam qualificações ou medições).

Por fim, foi retirada a feature StudyID do dataset, porque esta é uma String que não contém informação quantitativa relacionada com os Exames nem com o nívelde Stress.

3 Visualização dos dados

Nesta secção pretende-se analisar os valores das features presentes no dataset, de maneira a que seja possível identificar padrões e situações anormais (i.e missing values, outliers) que requerem tratamento, para que não tenham, futuramente, impacto negativo no nosso modelo.

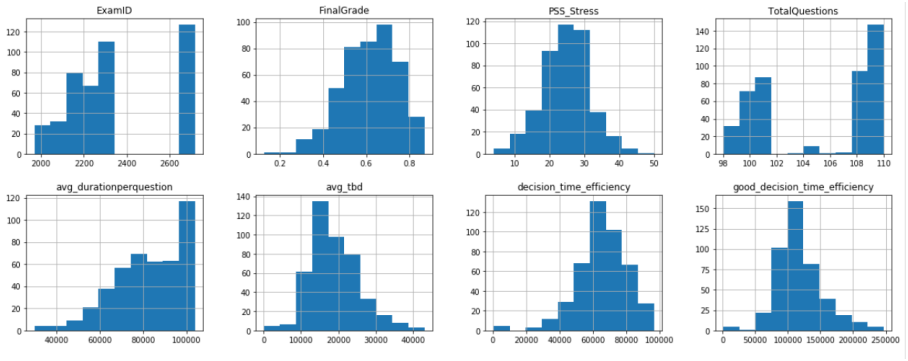

Figura 1: Exemplo da representação do primeiro conjunto de variáveis

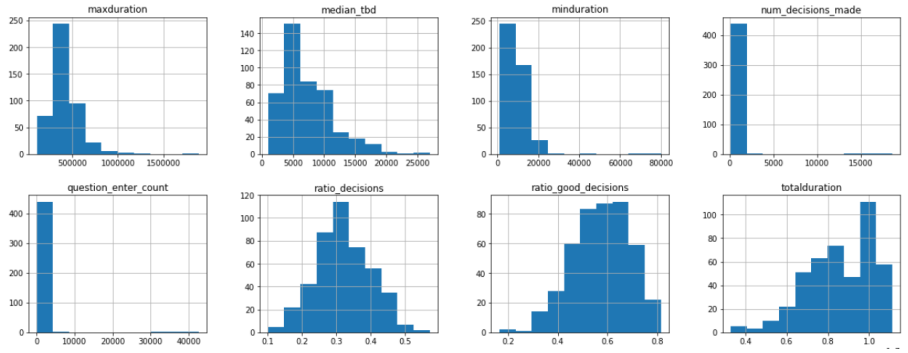

Figura 2: Exemplo da representação do segundo conjunto de variáveis

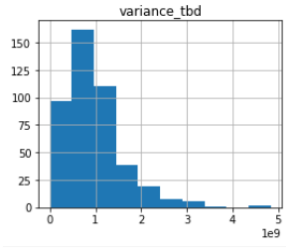

Figura 3: Exemplo da representação dos dados da variável variance_tdb

Através da análise dos histogramas das figuras 1, 2 e 3 é possível verificar a distribuição das variáveis e analisar os intervalos de valores em que existem mais ocorrências, permitindo ter um ideia dos valores médios, mínimos e máximos de cada atributo.

É também possível detetar a existência de valores atípicos, por exemplo nos gráficos referentes a *min_ duration* e *num_ decisions_ made* existe uma ocorrência de um ou dois valores muito maiores aos que têm sido registados noutras instâncias.

Também foram gerados os gráficos de dispersão de todas as variáveis, e depois da análise dos mesmos, identificamos cinco outliers em várias *Features*, como por exemplo as que estão representadas nas figuras 4 e 5. Os restantes gráficos podem ser visualizados no ficheiro *DataVizualization.py*.

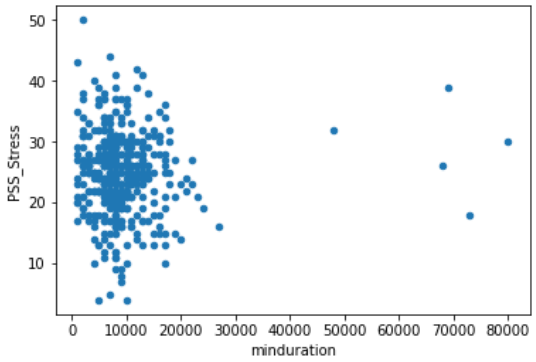

Figura 4: Distribuição da minduration relativamente ao valor de PSS

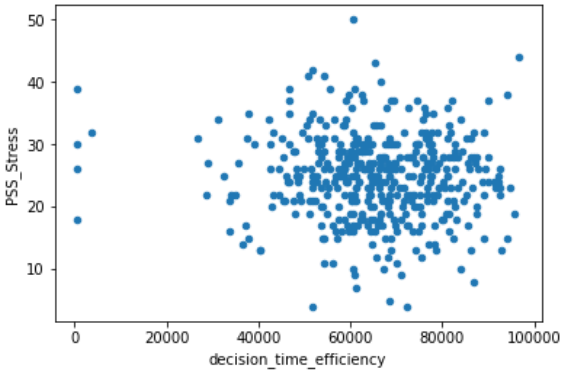

Figura 5: Distribuição da decision_time_efficiency relativamente ao valor de PSS

Para além dos gráficos acima referidos, foi utilizado uma *scatter matrix* com o objetivo de verificar possíveis relações entre a variável a prever e as restantes. Na figura 6 estão apenas representadas algumas das features, onde mais uma vez é possível verificar a presença dos cinco *outliers* e também é possível verificar um padrão, pois existe uma grande concentração de valores que dizem respeito a níveis de PSS intermédios. O gráfico completo pode ser visualizado no ficheiro *DataVizualization.py*.

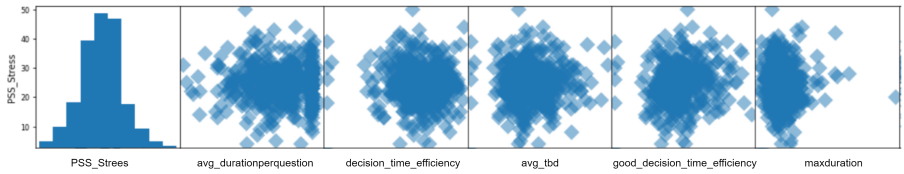

Figura 6: Distribuição de algumas Features em relação ao PSS

4 Pré-processamento dos dados

4.1 Normalização

Na secção anterior, verificamos que os valores das variáveis do dataset possuiam escalas muito diferentes, pois existiam desde valores percentuais até valores numéricos de elevada grandeza. Deste modo, numa perspetiva de melhorar o desempenho do modelo e a sua performance, decidimos normalizar os dados utilizando a função *MaxMinScaler* que mapeia os valores do dataset para números representativos entre zero e um.

```
data_scaler=sk.preprocessing.MinMaxScaler(feature_range=(0,1))
X=data_scaler.fit_transform(X)
```

4.2 Discretização

A variável a prever *PSS_Stress* varia no intervalo de zero a cinquenta e dois o que nos levou a identificar uma situação que requer análise. O dataset em questão possui apenas cerca de quatrocentas instâncias, sendo que a maior parte destas representam medidas associadas a níveis intermédios de PSS_Stress. Isto poderia influenciar a taxa de acerto do modelo, pois o número de variáveis a prever é bastante amplo.

Com o intuito de contornar este problema, o grupo optou por discretizar a variável PSS_Stress em cinco intervalos de acordo com a tabela 7.

[0,10[	1	Inexistente
[10,20[	2	Baixo
[20,30[	3	Moderado
[30,40[	4	Muito elevado
[40,52]	5	Extremo

Figura 7: Discretização da variável PSS_Stress

A discretização foi feita recorrendo à função *cut* conforme o seguinte código.

```
bins = [0,10,20,30,40,55]
group_names = [1,2,3,4,5]
df['PSS_S'] = pd.cut(df['PSS_Stress'],bins,labels=group_names)
```

4.3 Valores em falta

Verificada a situação da existência de outliers em algumas variáveis do dataset, analisamos o ficheiro csv e constatamos que nas instâncias em que ocorriam estes valores, também havia falta de informação noutros atributos, em particular na variável *median_tdb*.

384000.0	2000.0	14000.0	166	1076	0.13091	0.48795
436000.0	6000.0	11000.0	109	226	0.32733	0.74312
186000.0	NaN	68000.0	16165	34302	0.29843	0.60476
196000.0	1000.0	48000.0	2754	4603	0.3277	0.63108
148000.0	NaN	69000.0	18553	42626	0.28345	0.51194
144000.0	NaN	73000.0	17321	31838	0.32327	0.50453
138000.0	NaN	80000.0	14459	30123	0.30202	0.61401
336000.0	9000.0	9000.0	172	257	0.35102	0.59884
469000.0	8000.0	5000.0	124	234	0.33423	0.64516

Figura 8: Valores em falta no dataset original

Posto isto, o grupo decidiu que a melhor maneira de tornar consistente o conjunto de dados para o modelo, seria eliminar estas linhas, uma vez que para além de representarem valores atípicos, possuem valores em falta. Utilizamos a função *dropna()* para apagar as linhas que continham valores *NaN*.

4.4 Seleção de variáveis

Para a seleção dos atributos, usamos os quatro métodos lecionados na cadeira, nomeadamente o *SelectKBest*, o *Recursive Feature Elimination* (RFE), o *Principal Component Analysis* (PCA) e o *VarianceThreshold*. Também usamos a função *corr* que verifica a correlação entre as várias colunas. Decidimos eliminar as features que apareciam em quase todos os resultados, que foram a *ratio_decisions* e a *ratio_good_decisions*.

```
US
[1.060e+03 1.399e+00 8.056e+02 8.130e+00 8.851e+04 6.472e+04 1.130e+05
 3.686e+05 4.222e+06 6.531e+04 8.394e+04 4.103e+03 6.826e+03 7.010e-01
 1.163e+00 1.074e+07 3.089e+10]
```

Figura 9: SelectKBest eliminou a ratio_decisions e a ratio_good_decisions

```
RFE
[ True False  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True]
```

Figura 10: Recursive Feature Elimination eliminou a ratio_decisions e a FinalGrade

```
PCA
[1.000e+00 5.111e-06 4.651e-08 2.030e-09 1.677e-10 5.525e-11 4.478e-11
 2.391e-11 4.528e-12 9.279e-14 7.080e-14 3.692e-15 9.853e-17 1.207e-18
 5.815e-21 1.805e-21 6.432e-22]
```

Figura 11: Principal Component Analysis eliminou a ratio_good_decisions e a variance_tbd

```
VT
[5.985e+04 1.611e-02 4.542e+01 2.034e+01 2.320e+08 4.088e+07 1.920e+08
 1.171e+09 2.817e+10 1.714e+07 2.270e+07 1.610e+04 5.856e+04 6.210e-03
 1.299e-02 2.524e+12 4.289e+17]
```

Figura 12: VarianceThreshold eliminou a ratio_decisions e a ratio_good_decisions

```
PSS_Stress                      1.000000
good_decision_time_efficiency   0.121809
ExamID                          0.073082
num_decisions_made              0.055256
ratio_decisions                 0.050242
question_enter_count            0.049713
variance_tbd                    0.046217
avg_tbd                         0.022109
maxduration                    -0.004781
median_tbd                     -0.012591
avg_durationperquestion        -0.015174
totalduration                  -0.017025
TotalQuestions                 -0.023629
decision_time_efficiency       -0.047220
minduration                    -0.053395
FinalGrade                     -0.212334
ratio_good_decisions           -0.212547
```

Figura 13: VarianceThreshold eliminou a FinalGrade e a ratio_good_decisions

Para a visualização destes resultados, pode-se executar o ficheiro *Features-Selection.py*.

5 Treino e Validação

5.1 Support Vector Machine

Para efetuar o treino e sua validação de uma *Support Vector Machine* (SVM), começamos por dividir o dataset através de um simples *Split*, em que 80% dos dados são para treino, e os restantes 20% são para validação, e obtivemos **0.5227272727272727** de pontuação.

Depois de executar várias vezes o *SVMSplit.py*, verificamos que os resultados variavam, visto que a divisão do dataset não era estática, e isso poderia não representar com exatidão, o poder de previsão do modelo, já que numas iterações obtinha-mos valores muito altos em relação ao anterior, e noutros obtinhamos valores muito baixos.

Então foi feito um programa que se chama *SVMIterative.py* que faz o *Split* acima referido, num ciclo *while* com cem iterações, e no fim o resultado final é apenas uma média desses scores calculados em cada iteração, e obtivemos o resultado **0.5975**.

Por fim para as *SVM's*, foi usado o *KFold* como método de validação e treino, através do ficheiro *SVMKFold.py* e obtivemos o *Score* de **0.59561540582774164**.

5.2 Linear Regression

O mesmo raciocínio foi usado neste modelo, com a diferença que, o *Linear Regression* apresenta valores de previsão com casas decimais, e como os valores PSS são números inteiros, antes de calcular o score é feito o arredondamento da previsão.

Para o método *Split*, usado no programa *LRSplit.py* numa das execuções, foi obtido o score de **0.6477272727272727**. Também foi feita uma versão iterativa no *LRIterative* e foi obtido o score de **0.586590909090909**. Para o *KFold*, no programa *LRKFold*, foi obtido a pontuação de **1.0**.

5.3 K-Nearest Neighbors

Neste modelo, o primeiro passo foi tentar descobrir o melhor número de vizinhos para se usar. Para isso treinamos e validamos o dataset através do método *KFold*, em que era feita a validação do modelo com vários números de vizinhos. Esta codificação está implementada no *KNN.py*

	precision	recall	f1-score	support
1	0.00	0.00	0.00	3
2	0.20	0.21	0.21	19
3	0.62	0.69	0.65	52
4	0.29	0.15	0.20	13
5	0.00	0.00	0.00	1
avg / total	0.45	0.48	0.46	88

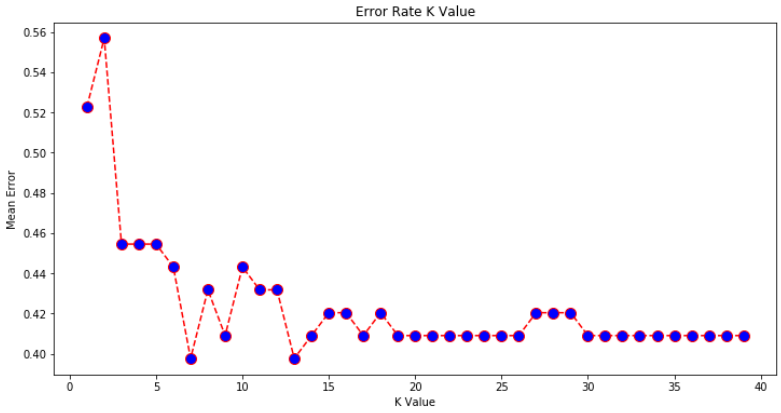

Figura 14: Resultados do KNN com K-Fold com 5 divisões

O gráfico mostra a média do erro em relação ao valor de K, sendo assim decidimos usar para este caso o valor 3, uma vez que quando se passa de 2 para 3 é quando há uma maior descida do erro.

Depois do número escolhido, passamos aos métodos de treino e validação adotados nos modelos anteriores. Atarvés do programa *KNNSplit.py* foi obtido o score de **0.42045454545454547**, com o *KNNIterative.py* foi obtido a pontuação de **0.4592045454545454** e por último com o *KNNKFold.py* teve **0.4798740693078408** de acerto nas previsões.

6 Seleção dos modelos

Dos três que foram usados neste processo de treino e validação, foi escolhido o *SVM*, visto que este ao contrário do *KNN* não depende de nenhuma variável de escolha com é o caso do número de vizinhos.

Em comparação ao *Liner Regression*, o modelo que usa as *SVM's* apresenta uma capacidade de previsão sem casas decimais, não necessitando fazer arredondamentos, e bem que no caso do *KFold* usado no *LR* é apresentado um score de 100%, o que pode causar uma falsa impressão de uma previsão prefeita, e por todos estes motivos escolhemos as *SVM's*, como modelo final.

7 Otimização por hiperparâmetros

Usando agora apenas o modelo selecionado, tentamos fazer uma otimização por hiper parametros, através da estratégia de *Grid Search*, esta estratégia está implementada no ficheiro *GridSearch.py*.

```
Best parameters set found on development set:

{'C': 1000, 'kernel': 'linear'}
```

Figura 15: Parâmetros indicados pelo Grid Search

Quando são encontrados esses parâmetros, eles são recebidos pelo modelo, e este treina e valida de novo o dataset, obtendo os resultados que são ilustrados na figura 16

	precision	recall	f1-score	support
1	0.00	0.00	0.00	6
2	0.00	0.00	0.00	73
3	0.60	1.00	0.75	211
4	0.00	0.00	0.00	57
5	0.00	0.00	0.00	5
avg / total	0.36	0.60	0.45	352

Figura 16: Resultados obtidos pelo SVM depois de parameterizado

Como se pode ver pela coluna do *recall* não existiram melhorias significativas, já que o resultado indicado não é muito melhor do que os que foram apresentados na secção 5.1

8 Conclusão

Após a realização deste projeto, o grupo adquiriu conhecimentos relativos às principais etapas de um processo de extração de conhecimento e também foi possível interagir com as principais ferramentas presentes do ambiente de desenvolvimento Python/Sklearn, aplicando-os na conceção de um modelo preditivo para a resolução de um problema real.

Ao longo do desenvolvimento do projeto as fases iniciais revelaram-se muito importantes para a conceção do projeto, pois permitiu-nos fazer uma análise cuidada do dataset a tratar e com base nisso definir estratégias de tratamento dos dados para implementação do modelo preditivo final.

É importante referir que o grupo reconhece algumas falhas em decisões tomadas em determinadas etapas do processo, nomeadamente na discretização da variável PSS_Stress foi utilizada a função *cut* que permite a divisão das classes com distribuições diferentes. Para este caso de estudo, a utilização da função *qcut* que divide as variáveis equitativamente pelos intervalos criados seria o mais sensato, pois no dataset estão representados maioritariamente medidas correspondentes a níveis de PSS médios (entre 20 e 30). Posto isto, apesar da taxa de acerto do modelo aumentar substancialmente consideramos que houve *overfitting*, sendo uma maneira de corrigir isso a utilização da função *qcut*.

Em suma, o grupo faz uma apreciação positiva do trabalho realizado, pois considera que apesar de algumas falhas o objetivo principal foi atingido com sucesso.