UNIVERSIDADE DE COIMBRA

Human Behaviour Classification

Parte A e B

Gonçalo Gouveia	Joana Moura	Jorge Silva
2018277419	2018277419	2018277419

13 de dezembro de 2022

Parte A: Análise e tratamento de Outliers. Extração de informação

> "O objectivo é identificar e tratar outliers no dataset usando diferentes abordagens univariável e multivariável" e "comprimir o espaço do problema, extraindo informação característica discriminante que permita implementar soluções eficazes do problema de classificação".

A análise realizada é feita aos dados do dataset FORTH-TRACE[1].

1 Dados

Primeiramente fizemos o tratamento inicial de dados, passamos todos os dados para a framework *pandas* do *python*, que nos permitiu fazer uma análise eficiente e rápida dos dados.

Fizemos um estudo das características do dataset. Por exemplo, começamos por analisar se todos os participantes realizaram todas as atividades, e com o mesmo número de repetições (figura 1).

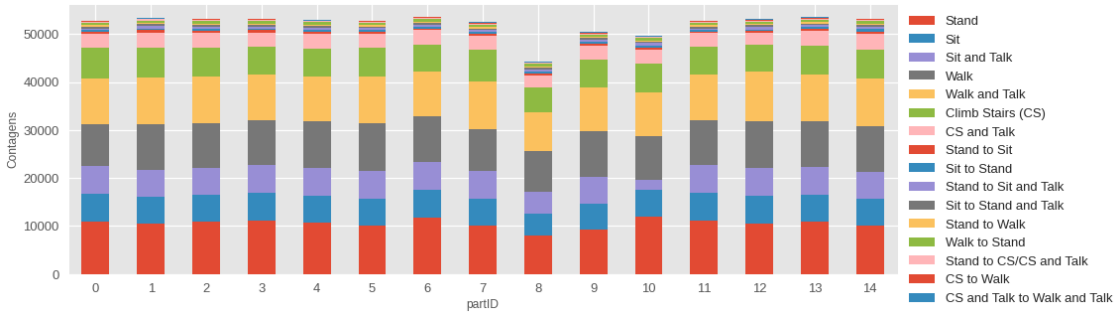


Figura 1: Contagem dos dados por atividades e por participante. Cada cor representa uma atividade. Desta maneira conseguimos ter uma perceção inicial do balanceamento dos dados. Por exemplo, o participante 8 produziu menos dados em geral. Vemos também que atividades como *Stand* ou *Text and Talk* são mais representativas do dataset do que, por exemplo, *CS to Walk*.

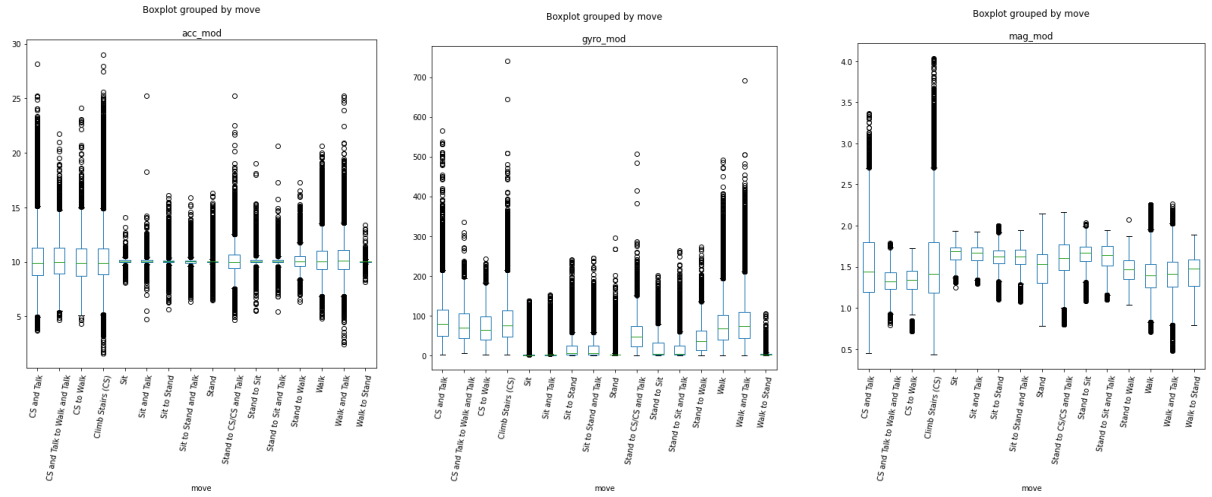
Em seguida calculamos os módulos dos valores lidos por cada sensor, da direção x, y, z

	deviceID	accX	accY	accZ	gyroX	gyroY	gyroZ	magX	magY	magZ	t	label	partID	move	acc_mod	gyro_mod	mag_mod
0	2	3.0317	9.0450	3.1739	0.76247	3.7071	-0.31384	0.57764	0.68849	1.2654	83.629	1	0	Stand	10.053699	3.797690	1.552071
1	2	2.9591	9.0436	3.1400	0.40588	4.6682	-0.10148	0.58385	0.68849	1.2741	83.160	1	0	Stand	10.020108	4.686910	1.561483
2	2	2.9465	9.0545	3.0799	0.24230	5.5225	0.34857	0.56522	0.67857	1.2675	102.690	1	0	Stand	10.007578	5.538792	1.544826
3	2	2.7898	9.0528	3.0726	0.49858	6.3463	0.61127	0.56108	0.65476	1.2610	122.220	1	0	Stand	9.958767	6.395135	1.527626
4	2	2.6924	9.0377	2.9794	0.63197	6.6972	1.00560	0.57764	0.69246	1.2675	141.750	1	0	Stand	9.889685	6.801699	1.555547
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53243	2	1.6802	9.5395	2.9636	0.83324	1.9832	0.87140	-0.38716	0.87103	1.3289	1042000.000	1	14	Stand	10.129564	2.320928	1.635409
53244	2	1.7042	9.5639	2.9504	1.03060	1.8002	0.54192	-0.37474	0.86508	1.3399	1042000.000	1	14	Stand	10.152701	2.143953	1.638330
53245	2	1.7283	9.5641	2.9496	1.05330	1.3577	0.89447	-0.35818	0.92857	1.3531	1042000.000	1	14	Stand	10.156730	1.937232	1.679707
53246	2	1.7285	9.5646	2.9737	1.03550	1.4340	0.78856	-0.36232	0.90675	1.3377	1042000.000	1	14	Stand	10.164260	1.936608	1.656174
53247	2	1.6802	9.4907	2.9646	0.77330	1.7544	1.01180	-0.38095	0.86310	1.3180	1042000.000	1	14	Stand	10.083914	2.167868	1.620860

Figura 2: Dataset inicial com cálculo de módulos respetivos

[1]link repositório: `https://github.com/spl-icsforth/FORTH_TRACE_DATASET`

Boxplot



(a) acc (b) gyro (c) mag

Figura 3: Boxplot de cada sensor para cada atividade.

Z-score

O Z-score é, para um dado evento, o número de desvios padrão que este se encontra acima ou abaixo da média da amostra. Ou seja, se uma pontuação Z for 0, isso indica que a pontuação do ponto de dados é idêntica à pontuação média. Um Z-score de 1,0 indicaria, por exemplo, um valor que é um desvio de padrão da média.

Na figura 4 apresentamos precisamente o resultado do cálculo do Z-score aos nossos dados. Relativamente aos diferentes números de k utilizados, verificamos que, para valores de k maiores, diminui o número de pontos considerados *outliers* (pontos a vermelho da figura 4). Esta constatação faz sentido e era esperada, uma vez que ao usarmos um k maior estamos a tornar mais longínquo o limite a partir do qual consideramos os pontos como *outliers*.

O aspeto que salta à vista é que para algumas atividades os pontos considerados *outliers* concentram-se (maioritariamente ou totalmente) de um lado da gama de valores possíveis. Exemplo disso são os dados registados pelo acelerámetro para por exemplo a atividade 15 ou 16, ou para qualquer atividade registada pelo giroscópio. A conclusão que tiramos daqui é que existem algumas atividades cujos dados têm uma distribuição mais parecida com uma distribuição normal, enquanto que outras atividades (as que têm os pontos vermelhos maioritariamente, ou mesmo todos, de um único lado) apresentam assimetria, ou seja, *skewness* diferente de zero. Um simples histograma dos pontos corrobora esta conclusão.

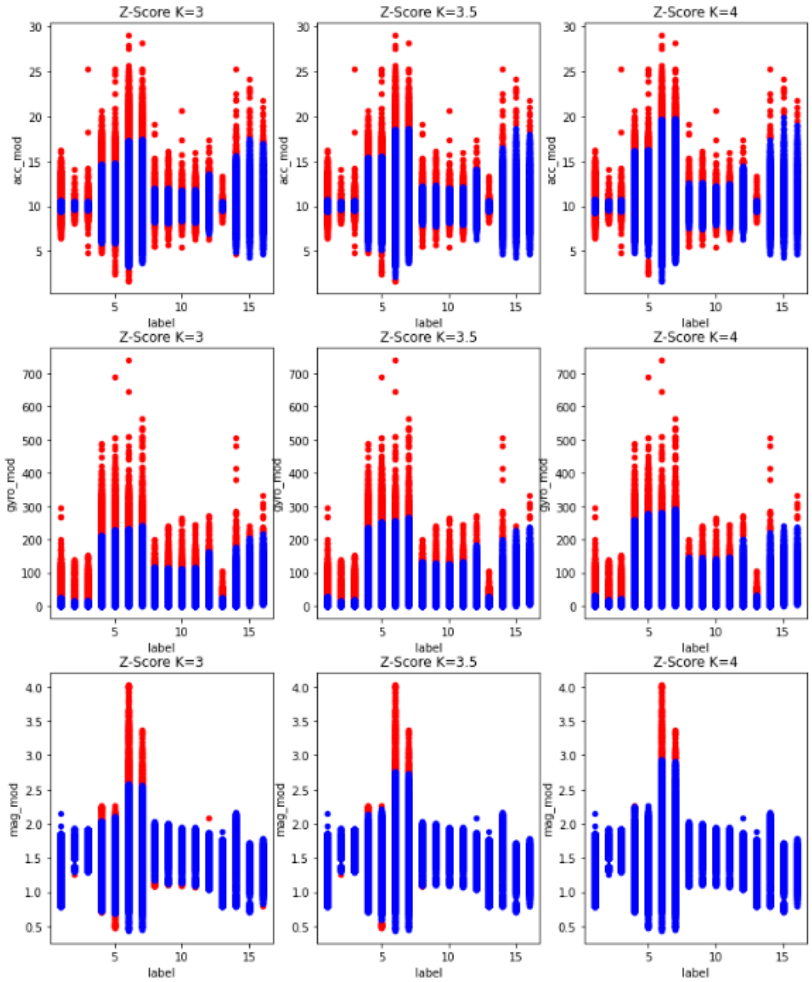


Figura 4: Representação dos valores de Z-score para as 3 *features* dos módulos (cima para baixo). Os pontos representados a vermelho são os pontos considerados *outliers*. Considerámos três valores de k, conforme pedido, k=3, 3.5, 4 (da esquerda para a direita).

Outliers

	move	densOut acc_mod	densOut gyro_mod	densOut mag_mod
0	Stand	4.195538	9.484932	0.000000
1	Sit	0.215558	6.646580	6.672780
2	Sit and Talk	0.513851	9.117459	4.766364
3	Walk	3.530950	1.620764	1.450684
4	Walk and Talk	3.477656	1.428009	1.338803
5	Climb Stairs (CS)	5.165680	1.498450	0.390559
6	CS and Talk	4.514626	2.073485	0.221507
7	Stand to Sit	15.444712	7.466947	3.485577
8	Sit to Stand	19.797585	10.777699	7.350852
9	Stand to Sit and Talk	15.324519	10.336538	0.841346
10	Sit to Stand and Talk	18.322173	11.867560	6.826637
11	Stand to Walk	10.918635	3.307087	0.026247
12	Walk to Stand	4.790026	8.254593	0.000000
13	Stand to CS/CS and Talk	14.173228	2.362205	6.666667
14	CS to Walk	4.934383	2.362205	6.666667
15	CS and Talk to Walk and Talk	4.986877	2.047244	1.574803
Total	NaN	130.305998	90.651756	48.279493

	move	densOut acc_mod	densOut gyro_mod	densOut mag_mod
0	Stand	0.841135	1.487187	0.000000
1	Sit	0.236995	0.466845	0.001191
2	Sit and Talk	0.363515	0.638308	0.000000
3	Walk	1.069790	0.910429	0.082896
4	Walk and Talk	1.069075	0.867482	0.064622
5	Climb Stairs (CS)	1.301492	0.906456	0.493515
6	CS and Talk	1.358727	1.096116	0.365372
7	Stand to Sit	1.832933	1.832933	2.524038
8	Sit to Stand	2.539062	2.183949	0.177557
9	Stand to Sit and Talk	2.223558	2.118389	0.375601
10	Sit to Stand and Talk	2.994792	2.511161	0.130208
11	Stand to Walk	1.837270	1.627297	0.026247
12	Walk to Stand	1.233596	2.467192	0.000000
13	Stand to CS/CS and Talk	1.312336	1.548556	0.000000
14	CS to Walk	1.259843	0.892388	0.000000
15	CS and Talk to Walk and Talk	1.259843	1.154856	0.104987
Total	NaN	22.733961	22.709543	4.346234

Figura 5: Resultados do cálculo das densidades de outliers para os módulos dos três vetores em análise e para cada atividade usado o boxplot (esquerda) e usando o z-score (direita). Resultados em percentagem. Os valores podem ser comparados porque usam a mesma normalização.

Pelos resultados da densidade de outliers (figura 5) verificamos que em 12 das 16 atividades o sensor de aceleração é o que apresenta predominantemente maior densidade de *outliers*, relativamente aos outros dois sensores, sendo as exceções as atividades "Stand","Sit", "Sit and Talk"e "Walk to Stand"e "CS to Walk".

Ex 3.5

Queremos começar a comparação de resultados por dizer que usamos os valores típicos nestes métodos: k=3 e IQR=1.5. Pelos resultados da figura 5 podemos compararmos quantitativamente a densidade de *outliers* dada pelos dois métodos utilizados até aqui: o *boxplot* e o Z-score. Faz sentido o Z-score dar menos outliers pelo seguinte: estamos a usar k=3, o que, se os dados seguirem uma distribuição normal, então apenas 1% são considerados *outliers*. Como na verdade algumas distribuições não são normais, e temos dados finitos, podemos explicar as percentagens que quedam um pouco maior. Nas distruibuições assimétricas, podem não haver *outliers* num dos dados (vimos isso nos plots!) enquanto que com o boxplot haverá sempre *outliers* dos dois lados do espetro.

K-means

Para cada uma das atividadades e sensores, testamos vários valores de número de clusters, sendo que os que aparecem nos gráficos são os que são os melhores valores (utilizando o método da silhueta).

Relativamente aos outliers nos clusters, estes são definidos como outliers se o número de pontos no dado cluster for inferior a um threshold estipulado (5%) do total de pontos em análise). No caso de isso acontecer, todos os pontos desse dado cluster serão considerados outliers.

Fazemos a comparação graficamente entre o K-means e o Z-score no caso de o cluster ser inferior ao dado threshold e, de um modo geral, nunca conseguimos concluir nenhum resultado em concreto, pois estamos a considerar valores fixos (Z-score com k = 3 e threshold de 5%), sendo que por vezes o K-means deteta mais outliers que o Z-score e outras vezes o contrário.

Elbow Curve

Para determinar o número ideal de clusters nos quais os dados podem ser agrupados. O Método do Cotovelo é um dos métodos mais populares para determinar esse valor ótimo de k.
Ele funciona encontrando WCSS (Within-Cluster Sum of Square), ou seja, a soma da distância quadrada entre os pontos em um cluster e o centróide do cluster.

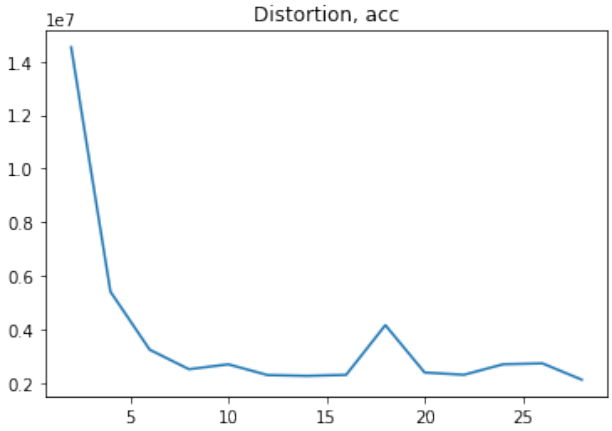


Figura 6

Silhouete Score

A fim de encontrar o número ideal de clusters em k médias, usamos a pontuação silhouete, que mede o quão semelhante a observação é ao cluster atribuído e quão diferente da observação do cluster próximo.

A pontuação da silhueta é uma métrica usada para calcular a qualidade de uma técnica de agrupamento. Seu valor varia de -1 a 1.

- 1: Significa que os clusters estão bem separados uns dos outros e claramente distintos.
- 0: Significa que os clusters são indiferentes, ou podemos dizer que a distância entre os clusters não é significativa.
- -1: Significa que os clusters são atribuídos da maneira errada.

sihouete.png

Figura 7

DBSCAN

O DBSCAN é um dos métodos de clustering mais usados porque os clusters encontrados pelo DBSCAN podem ter qualquer formato, o que pode lidar com alguns casos especiais que outros métodos não podem.[2]

O principal fato desse algoritmo é que a vizinhança de cada ponto em um cluster que está dentro de um determinado raio (R) deve ter um número mínimo de pontos (M).

[2] https://towardsdatascience.com/understanding-dbscan-and-implementation-with-python-5de75a786f9f

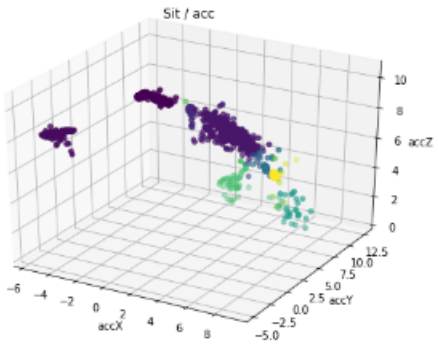


Figura 8

P.S Foi tambem feit a aanalise do metodo .DBSCAN e .dbscan percebendo as diferenças entre os mesmos, esta analise esta descrita no *colab*. De um modo geral, o DBSCAN é um melhor algoritmo de clustering que o K-Means, não é necessario definir um valor de k a priori de clusters e suporta formas irregulares de clusters (O DBSCAN não requer nenhuma forma dos clusters, mas rastreia as regiões de alta densidade).

Ex 3.8

Sem muito a dizer nesta parte, a implementação da injeção de outliers está no ficheiro de código da parte A.

Ex 3.9

Para avaliar a performance de um modelo para os nosso dados, temos de quantificar o quão bem o nosso modelo preve os dados.

Um dos parâmetro mais comuns para tal é o **mean squared error** (MSE).

Ex 3.10

Ex 4.1

Para podermos aferir que os dados seguem ou não uma distribuição normal, definimos um teste estatístico apropriado e um limiar bastante usado para o $p - value(5\% = 0.05)$, isto é, se o p-value for menor ou igual que este valor, então os dados não seguem uma distribuição normal, rejeitando assim a hipótese nula

O teste de Kolmogorov-Silmov entre os nossos dados (normalizados) e uma distribuição gaussiana (também normalizada) resultou em p-values muito inferiores ao usado como referência (0.05). A conclusão que tiramos é que os nossos dados não são bem descritos por uma distribuição normal. E, por isso, caracterizar os dados através dos valores médios e desvios-padrão não tem tanta significância estatística como podíamos pensar só de olhar para o histograma dos dados.

O Dataset foi adquirido com os participantes a realizar a mesma atividade várias vezes, o que, simplesmente pela questão da repetibilidade, seria de esperar que o teste de Kolmogorov-Silmov desse o resultado contrário ao que deu na verdade.

Uma justificação para o sucedido pode ser a variabilidade que há entre participantes, principalmente a idade e peso (ver citação em baixo).

> "Six participants with different gender, age, height, and weight are recruited to perform nine types of activities: walk forward, walk left, walk right, go upstairs, go downstairs, jump up, run, stand, and sit." [pág.2 artigo]

Para confirmar a nossa hipótese poder-se ia realizar o mesmo teste para cada participante em separado.

Ex 4.2

Os autores do artigo[3] utilizam, para a análise de dados, apenas 2 dos 3 sensores disponíveis, o acelerómetro e o giroscópio. Escrevemos o nosso código de acordo com estas características, mas facilmente é editável para incluir o terceiro sensor.

Desta forma obtemos o seguinte número de *features*:

- *features* estatísticas: 2 sensores x 3 eixos x 12 métricas + #{correlações} = 72 + 15 = 87
-

$$
\begin{array}{rrr}
\#\{\text{features físicas}\} = & \#\{MI\} & 2 \\
& +\#\{\text{SMA}\} & 1 \\
& +\#\{\text{EVA}\} & 2 \\
& +\#\{\text{CAGH}\} & 1 \\
& +\#\{\text{AVH}\} & 1 \\
& +\#\{\text{AVG}\} & 1 \\
& +\#\{\text{ARATG}\} & 1 \\
& +\#\{\text{ADF}\} & 6 \\
& +\#\{\text{ENERGY}\} & 6 \\
& +\#\{\text{AAE}\} & 1 \\
& +\#\{\text{ARE}\} & 1 \\
\hline
& = & 23
\end{array}
$$

Portanto, conseguimos identificar as 87+23=110 *features* indicadas no artigo. Do repositório com o dataset[4] obtemos a informação sobre a taxa de amostragem dos sensores: 51,2 Hz. Para comprovar que a taxa de amostragem indicada estava correta, calculámos a diferença de tempos (coluna 11) entre vetores consecutivos, cujo valor médio deu $\sim$ 20 ms (1/51.2 Hz)[5]. Tal como no artigo, vamos usar uma janela de 2 segundos.

Assim obtemos o seguinte dataset de features:

	accX_mean	accX_median	accX_std	accX_var	accX_rms	accX_avgDer	accX_skew	accX_kurtosis	accX_iqr	accX_zcr	...	gyroZ_DF	accX_ENERGY	accY_ENERGY	accZ_ENERGY	gyroX_ENERGY	gyroY_ENERGY	gyroZ_ENERGY	AAE	ARE	move
0	2.816928	2.81290	0.081657	0.006668	2.818100	-0.041974	0.104070	0.488731	0.084675	0.000000	...	0.501961	0.673458	0.399376	0.946173	215.113374	557.319074	1.586935e+02	2.019007	9.311260e+02	Stand
1	2.864813	2.86200	0.088376	0.007810	2.866162	0.099358	-0.237104	-0.098164	0.135050	0.000000	...	0.501961	0.788837	0.840539	2.375408	269.817928	571.838481	2.092227e+02	4.004784	1.050879e+03	Stand
2	2.846591	2.80950	0.347139	0.120506	2.869039	-0.600155	-0.984057	-0.323593	0.464225	0.000000	...	0.000000	12.171080	2.273113	1.044337	235.178880	1720.934265	8.297539e+02	15.488530	2.785867e+03	Stand
3	2.212950	2.09150	0.285796	0.081679	2.231149	-0.422577	0.830326	-0.620484	0.455650	0.000000	...	0.000000	8.249597	2.319877	0.751739	268.114752	1853.864081	8.761738e+02	11.321213	2.998153e+03	Stand
4	2.058188	2.05515	0.102485	0.010503	2.060713	0.079233	0.153356	0.366587	0.133325	0.000000	...	0.501961	1.060826	0.933459	0.392932	124.703677	315.283401	1.492463e+02	2.387217	5.892334e+02	Stand
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14483	-2.100210	-2.19370	0.641578	0.411623	2.195101	-0.154310	0.470383	-0.706343	0.992650	0.000000	...	50.196078	41.573873	407.773461	63.981284	22795.849415	100730.602971	1.231315e+05	513.328617	2.466580e+05	CS and Talk to Walk and Talk
14484	3.425963	4.08130	1.693263	2.867140	3.817886	-1.826497	-1.723238	2.626517	1.417625	0.501961	...	0.000000	289.581123	277.107186	110.473625	13432.853482	86110.698960	2.822261e+05	677.161934	3.817697e+05	CS and Talk to Walk and Talk
14485	2.307871	2.18690	1.030087	1.061080	2.525261	1.235324	0.010703	0.621858	0.955350	2.007843	...	49.694118	107.189052	905.170528	84.109265	33724.071772	272558.465214	1.398187e+05	1098.448845	4.461012e+05	CS and Talk to Walk and Talk
14486	1.404052	1.38710	0.719179	0.517218	1.575915	0.538254	-0.280569	0.222744	0.997815	1.003922	...	0.000000	52.239026	91.311396	135.752471	126775.273106	158173.559028	8.266789e+04	279.302893	3.676167e+05	CS and Talk to Walk and Talk
14487	2.874638	2.95255	0.902399	0.814324	3.011625	-0.606340	-0.332115	-0.940119	1.394350	0.000000	...	50.196078	82.246748	609.048544	108.330128	50826.247499	278158.421152	1.048426e+06	799.625420	1.377411e+06	CS and Talk to Walk and Talk

Figura 9: Dataset de features apresentadas na literatura e em que *'move'* é a coluna de *target*.

Apontamentos do artigo

- Variância tem boa precisão para atividade de Walking, Jogging e Hopping.
- Correlação entre os eixos dos sensores ajudam a diferenciar atividades que involvam translações numa so dimensão como Walking and Running das atividades que involvam translações em varias dimensões como Stair climbing.

Conforme iremos ver mais à frente, estas são algumas das features mais importantes.

P.S: Este dataset foi guardado num ficheiro *.csv* para ser usado posteriormente na parte B, ou sempre que for necessário. Assim evitamos voltar acorrer este bloco de código que é computacionalmente intensivo.

[3]link do artigo: `https://pdfs.semanticscholar.org/8522/ce2bfce1ab65b133e411350478183e79fae7.pdf`

[4]link repositório: `https://github.com/spl-icsforth/FORTH_TRACE_DATASET`

[5]Sublinhamos o facto de ser um valor médio. O dataset apresenta "descontinuidades"na coluna do tempo, que podem ser consequência ou de erros na aquisição ou de eliminação dos dados de um participante (o participante pode, por exemplo ter feito mal o exercício). Não tendo qualquer informação sobre este aspeto, simplificamos o problema e assumimos que foi erro de leitura, enquanto que uma análise mais conservadora seria remover esses dados da análise seguinte (janelas temporais).

Ex 4.3 e 4.4

A Principal Component Analysis (PCA) é uma técnica que transforma um conjunto de dados de grandes dimensões em menores dimensões, mantendo o máximo de informação possível.

A construção de características relevantes é conseguida pela transformação linear de variáveis correlacionadas em um número menor de variáveis não correlacionadas. Isso é feito projetando (produto escalar) os dados originais no espaço PCA reduzido usando os valores próprios da matriz de covariância, também conhecidos como componentes principais(PCs).Também é comummente utilizado para visualização de dados em duas ou três dimensões, para que os dados possam ser visualizados e explorados de forma mais fácil.

Para saber se o PCA é apropriado para um determinado conjunto de dados e quando usá-lo, é importante entender o objetivo da análise e o que se deseja obter com os dados. Se o conjunto de dados tem muitas variáveis e é difícil de visualizar ou analisar diretamente, ou se deseja melhorar o desempenho de outras técnicas de machine learning, o PCA pode ser uma boa opção. No entanto, é importante lembrar que o PCA irá reduzir a dimensionalidade dos dados, o que pode resultar em perda de informação. Por isso, é importante avaliar cuidadosamente se o PCA é apropriado para um determinado conjunto de dados e objetivo.

Para explicar 75% do nosso feature set através do PCA **são necessárias 22 dimensões** (figura 10).

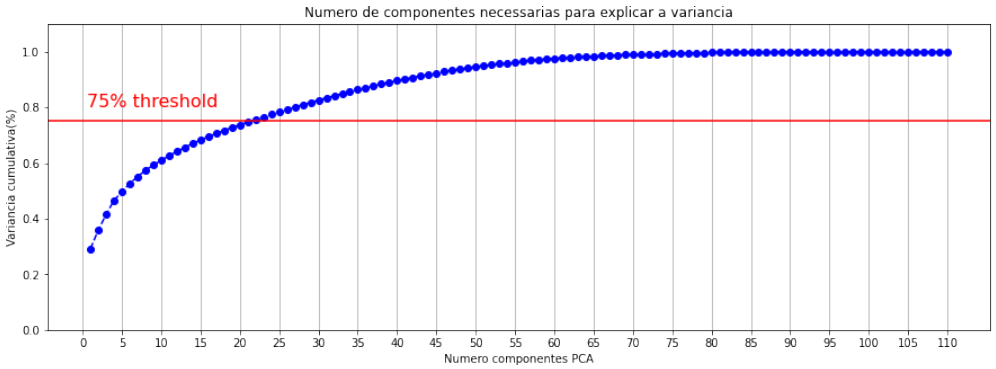


Figura 10: Evolução do explicação do feature set em função do número de componentes do PCA usadas. Uma linha vermelha marca o alvo de 75%.

Como usar o PCA

- A técnica PCA é particularmente útil no processamento de dados onde existe multicolinearidade entre as variáveis.
- O PCA pode ser usado quando as dimensões dos recursos de entrada são altas.
- O PCA também pode ser usado para redução de ruído e compactação de dados.

Desvantagens

- Perda de informação.
- Perda de intrepertabilidade.

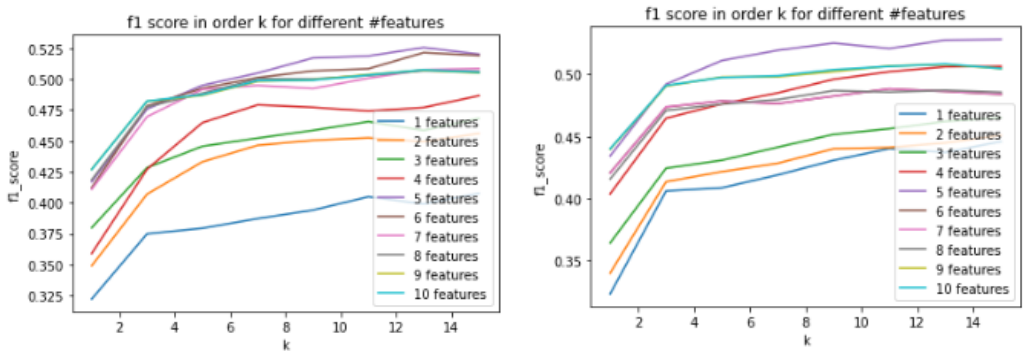
4.5 e 4.6

O **Relief F** é um algoritmo de seleção de recursos que foi desenvolvido como uma melhoria do algoritmo original Relief. É usado para selecionar recursos relevantes num conjunto de dados, o que pode ser útil em tarefas de aprendizagem, como classificação e agrupamento. Uma vantagem do Relief F é que ele é capaz

de lidar com conjuntos de dados com muitas variáveis e classes. Além disso, ele é geralmente considerado um algoritmo eficiente em termos do tempo de execução. A principal desvantagem do Relief-F é que ele não considera dependências entre variáveis e, portanto, não ajuda a remover recursos redundantes.

O **Fisher Score** é ,outro algoritmo de seleção de variáveis relevantes, baseado na ideia de que as variáveis escolhidas devem ser capazes de diferenciar claramente os dados entre as classes presentes. Uma vantagem do Fisher Score é que ele é capaz de lidar com conjuntos de dados que possuem uma grande quantidade de variáveis. No entanto, ele pode ser menos eficiente em termos de tempo de execução em comparação com outros algoritmos de seleção de recursos.

Foram implementados ambos os códigos, que depois vieram ser usados na **parte B**, embora ambos reduzam o número de *features* como esperado, e tal como se previu na literatura o **Fischer Score** demororu mais tempo a acabar de executar que o **Relief F**



(a) Relief-F (execution: 24603.8108 ms) (b) Fischer score (execution: 32976.79877 ms)

Figura 11: Plots Relief-F e Fischer Score, tempo de execução relativo (at that run time). Imagens obtidas usando os métodos respetivos na pergunta 3 da parte B do projeto.

Também se notou que após implementar os algoritmos de ReliefF e de Fisher Score para retirar as 10 melhores features para representar o dataset, não existe uma única feature em comum entre as duas abordagens.

Fisher: Features[57 59 81 84 85 107 20 53 22 75]

Relief: Features[101 99 33 100 69 45 49 37 36 35]

Concluimos que 10 features podem não ser suficiente para representar o Dataset, estando assim a perder informação relevante.

A vantagem destas abordagens é que se consegue reduzir bastante o numero de features sem perder a interpretabilidade dos dados, ao contrário do que acontece no algoritmo do PCA. Outra vantagem é que assim podemos estar a remover/ignorar features que sejam redundantes ou que adicionem pouca informação.

2 Parte B:Tarefas de Aprendizagem Computacional e Avaliação

1.

A principal diferença entre os métodos de treino-teste, treino-validação-teste e dividir dados em K-folds é a forma como os dados são divididos e utilizados para treinar e avaliar um modelo de aprendizado de máquina.

1.1.

Train-Test

No método train-test, dividimos os dados em dois conjuntos: um conjunto de treino e um conjunto de teste. O modelo é treinado usando o conjunto de treino e depois é avaliado usando o conjunto de teste. Esta abordagem é útil quando queremos avaliar o desempenho do modelo em dados diferentes dos utilizados durante o treino.

```
#Train-Test (TT)
#Função que separa os dados em 2 grupos (Train, Test) com base numa percentagem (test_size)
def Train_Test(x, y, test_size):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = test_size)

    return x_train,x_test,y_test,y_train


x_train,x_test,y_test,y_train = Train_Test(X,Y_n,0.25)
#Atribui a cada return uma variável
```

Train-Validation-Test

No método train-validation-test, dividimos os dados em três conjuntos: um conjunto de treino, um conjunto de validação e um conjunto de teste. Divide-se o conjunto de dados inicial num conjunto de treino+validação e num de teste e, de seguida, o conjunto de treino+validação é também dividido, resultando nos três conjuntos treino, validação e teste. Numa primeira fase, o modelo é treinado utilizando o conjunto de treino e depois avaliado usando o conjunto de validação, o que permitirá ajustar os parâmetros do modelo antes de uma avaliação final utilizando o conjunto de teste. Este método é útil quando queremos encontrar os melhores parâmetros para o nosso modelo.

```
#Train-Validation-Test data split
#Função que separa os dados em 3 grupos (Train, Validation, Test) com base em duas percentagens (test_size, validation_size)
def Train_Validation_Test(x, y, test_size, validation_size):
    x_train_validation, x_test, y_train_validation, y_test = train_test_split(x, y, test_size = test_size)
    #começa-se por dividir o conjunto total em duas percentagens de treino+validação e teste

    x_train, x_validation, y_train, y_validation =
    train_test_split(x_train_validation, y_train_validation, test_size = validation_size/(1-test_size))
    #depois divide-se de novo o conjunto resultante anterior em duas percentagens de treino e validação
    #a percentagem de validação será sob o conjunto resultante da primeira divisão

    return x_train, x_validation, x_test, y_train, y_validation, y_test

x_train, x_validation, x_test, y_train, y_validation, y_test = Train_Validation_Test(X,Y_n,0.15,0.15)
```

K-fold data split

O método k-fold cross-validation é um pouco diferente. Em vez de dividir os dados em conjuntos de treino, validação e teste, ele divide os dados em k secções ou partições. O modelo é treinado k vezes, cada vez usando uma partição diferente como conjunto de teste (das k secções totais) e as k-1 secções restantes como conjunto de treino. Em seguida, os resultados são agregados para avaliar o desempenho do modelo. Esse método é útil quando temos poucos dados disponíveis e queremos aproveitar ao máximo os dados que temos.

```python
#K-fold data split (para N splits)
def K_Fold(x,y,n_splits):
    kf = KFold(n_splits = n_splits, random_state = None, shuffle = True)
    result = next(kf.split(x,y), None)
    x_train = pd.DataFrame(x).iloc[result[0]] #iloc
    x_test =  pd.DataFrame(x).iloc[result[1]]
    y_train = pd.DataFrame(y[result[0]], columns = ['label'])
    y_test = pd.DataFrame(y[result[1]], columns = ['label'])
    return x_train, x_test, y_train, y_test

x_train, x_test, y_train, y_test = K_Fold(X,Y,5) # Define the split - into 5 folds (at least 2)
```

1.2. Métricas de Exatidão

Com o objetivo de avaliar a performance dos métodos de classificação, é importante estabelecer funções de cálculo de métricas apropriadas para tal. Nesse sentido, tratando-se de problemas de classificação, foram escolhidas as seguintes métricas:

- **Recall (Sensitivity)**: Rácio entre true positives e todas os casos realmente positivos (True Positives + False Negatives). Explica quantos dos casos realmente positivos o modelo de classificação previu corretamente. É dada a partir da seguinte equação:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{1}$$

- **Precision**: Rácio entre True Positives e todas as previsões positivas (True Positives + False Positives). Explica quantos dos resultados corretamente previstos são positivos. É dada a partir da seguinte equação:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2}$$

- **F1-Score**: Média harmónica da precisão e do recall. Atinge o valor máximo quando a precision é igual ao recall. É dada a partir da seguinte equação:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

- **Matriz de confusão**: A matriz de confusão é uma tabela que é usualmente utilizada para descrever a performance de um modelo de classificação sob um conjunto de dados de teste, com valores reais conhecidos.

		Predicted Values	
		Positive (1)	Negative (0)
Actual Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Tabela 1: Exemplo generalizado de uma matriz de confusão.

Pode interpretar-se a tabela anterior utilizando uma analogia de diagnóstico médico:

- **True Positive (TP)**: O modelo de classificação preveu que o caso era positivo (doente) e é verdadeiro;
- **True Negative (TN)**: O modelo de classificação preveu que o caso era negativo (saudável) e é verdadeiro;
- **False Positive (FP)**: O modelo de classificação preveu que o caso era positivo (doente), o que é falso (erro tipo 1);
- **False Negative (FN)**: O modelo de classificação preveu que o caso era negativo (saudável), o que é falso (erro tipo 2).

De seguida, apresentam-se as aplicações destes conceitos na resposta ao problema 1.2.:

```
#Classification Errors
#nota: y_test = y_real

#confusion matrix vai avaliar a accuracy do classificador
#cada classe (sentado, correr,...) conta como um indice na matriz (2*2,3*3)
#predictions (total) são todos os números da matriz somados

def plot_confusion_matrix(y_test, y_pred):
    labels = np.unique(y_test)
    column = [f'Predicted {label}' for label in labels]
    indices = [f'Actual {label}' for label in labels]
    table = pd.DataFrame(confusion_matrix(y_test,y_pred), columns = column, index = indices)

    plt.title("Confusion Matrix")

    return sns.heatmap(table,annot = True, fmt = 'd', cmap='viridis')

#plot_confusion_matrix(y_test,y_pred) - teste

from sklearn.model_selection import cross_validate
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

#Função que devolve métricas de exatidação: Recall, Precision e F1-Score
def metricas(y_true, y_pred, print_ = True):

  recall = recall_score(y_true, y_pred, average = 'weighted')
  precision = precision_score(y_true, y_pred, average = 'weighted')
  f1 = f1_score(y_true, y_pred, average = 'weighted')
  if(print_):
    print("Recall/Sensitivity Score: ", recall,"\n")
    print("Precision Score: ", precision,"\n")
    print("F1 Score: ", f1,"\n")

    return recall, precision, f1
```

Figura 12: Código de implementação de funções de métricas de exatidão para modelos de classificação.

Ex 2

2.1

Para avaliar a capacidade do classificador K Nearest Neighbours no dataset Iris, fizeram-se algumas funções com diferentes métodos de Data Splitting, com o número de vizinhos, k, constante e também para um intervalo de diferentes valores.

2.1.2.

Fazendo k variar na gama 1, 3, 5, . . . , 15, aplicaram-se os métodos de divisão de dados Train-Only, Train-Validation-Test e 10×10 Cross-Validation, utilizados também no ponto 2.1.1., desta vez com o objetivo de criar vários modelos e avaliá-los para obter o ideal. Para cada método foram obtidos os seguintes resultados:

Train Only

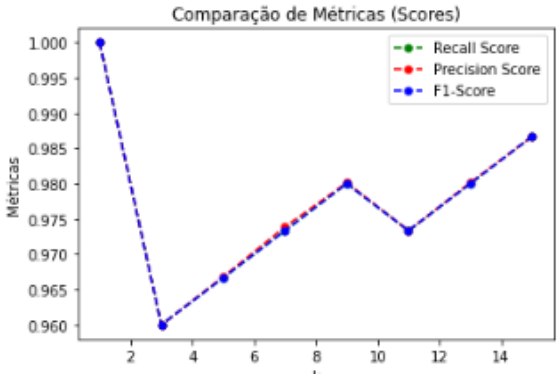


Figura 13: Valores das métricas em função do número de vizinhos k para Data Splitting Train-Only.

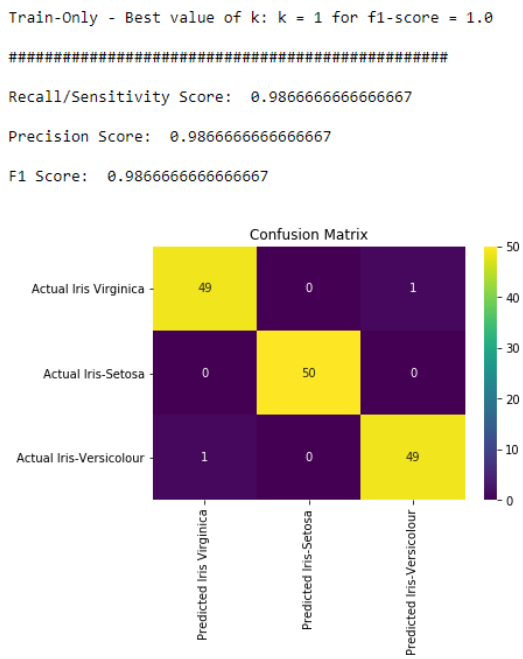


Figura 14: Matriz de confusão e métricas resultantes do modelo ideal kNN para Data Splitting Train-Only.

Train Validation Test

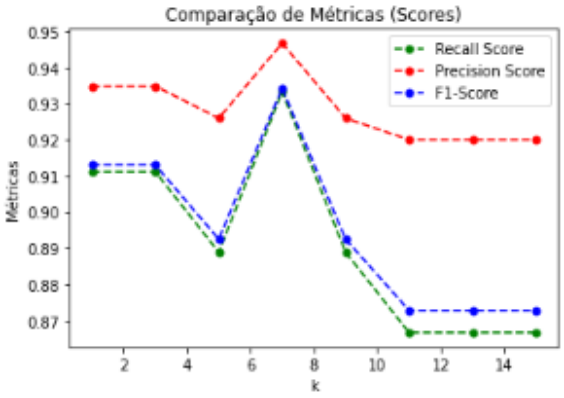


Figura 15: Valores das métricas em função do número de vizinhos k para Data Splitting Train-Validation-Test.

```
Train-Validation-Test - Best value of k: k = 7 for f1-score = 0.3398467432950191

##################################################

Recall/Sensitivity Score:  0.9555555555555556

Precision Score:  0.9623931623931624

F1 Score:  0.955925925925926
```

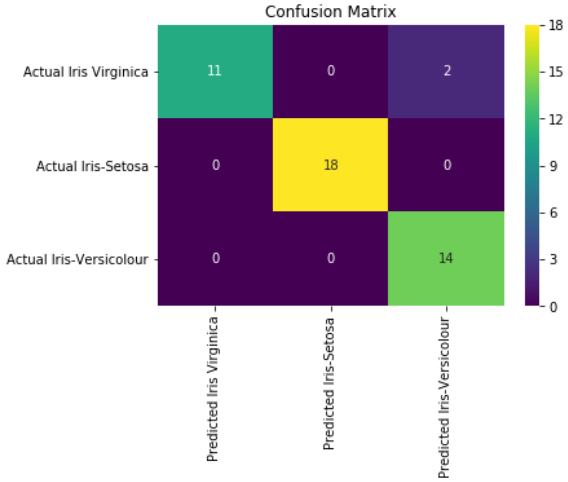


Figura 16: Matriz de confusão e métricas resultantes do modelo ideal kNN para Data Splitting Train-Only.

10×10-Fold Cross Validation

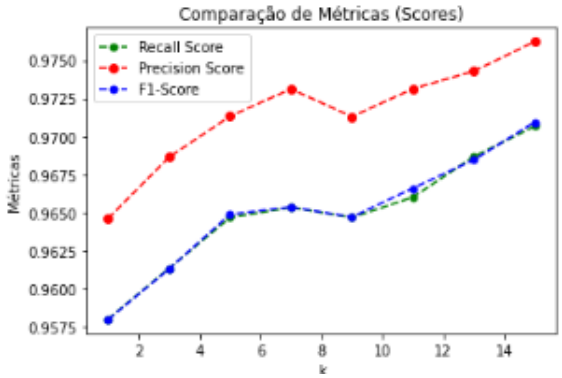


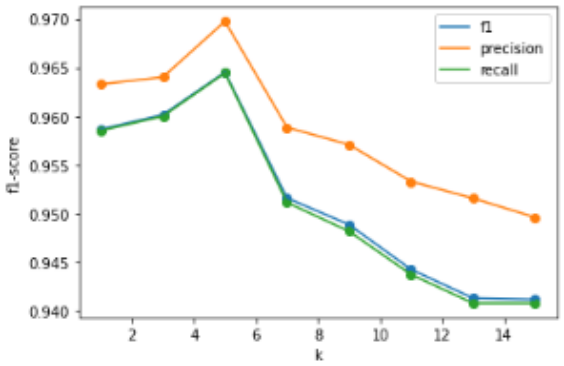
Figura 17: Valores das métricas em função do número de vizinhos k para Data Splitting 10×10 Cross-Validation.

2.1.3 Discussão

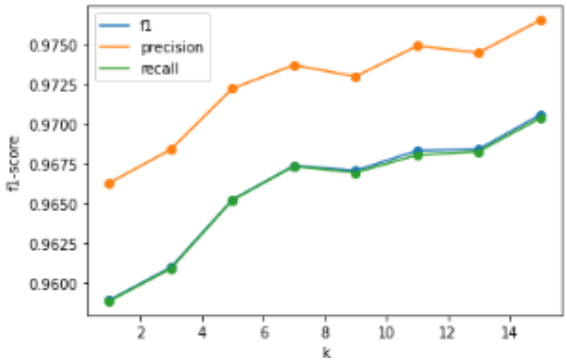
Bias : é o erro devido à diferença entre as previsões médias e os valores corretos que estamos tentando prever, quando treinamos um modelo, geralmente repetimos o processo de treinamento n vezes e cada vez que o modelo é treinado, um novo modelo é criado e pela aleatoriedade dos dados, consequentemente teremos uma variedade de previsões, então o Bias ou Viés é a distância, em geral, das previsões para o valor correto.

Variance : é o erro devido à variabilidade de uma previsão do modelo para um determinado ponto de dados. Repetindo novamente o treinamento, a variância mede o quanto as previsões do modelo variam entre diferentes realizações do modelo.

Assumindo que o Recall (accuracy) é à Bias e Precision (precisão) refere-se à Variancia, o F1-score é o meio termo de ambos. Implemenatmos um codigo que faz a media dos resulatdos obtidos para os seguintes mdoelos (25 runs). E podemos analizar o a Bias-variance.[6]



(a) Média de resultados do TVT, 25 runs e k a variar.



(b) Média de resultados do CV10-TT-KNN, 25 runs e k a variar.

Figura 18

Podemos concluir que para k demasiados baixos temos underfitting e para k demasiado altos temos overfitting

[6] https://towardsdatascience.com/tradeoffs-how-to-aim-for-the-sweet-spot-c20b40d5e6b6

2.2 & **2.3**

Relief-F

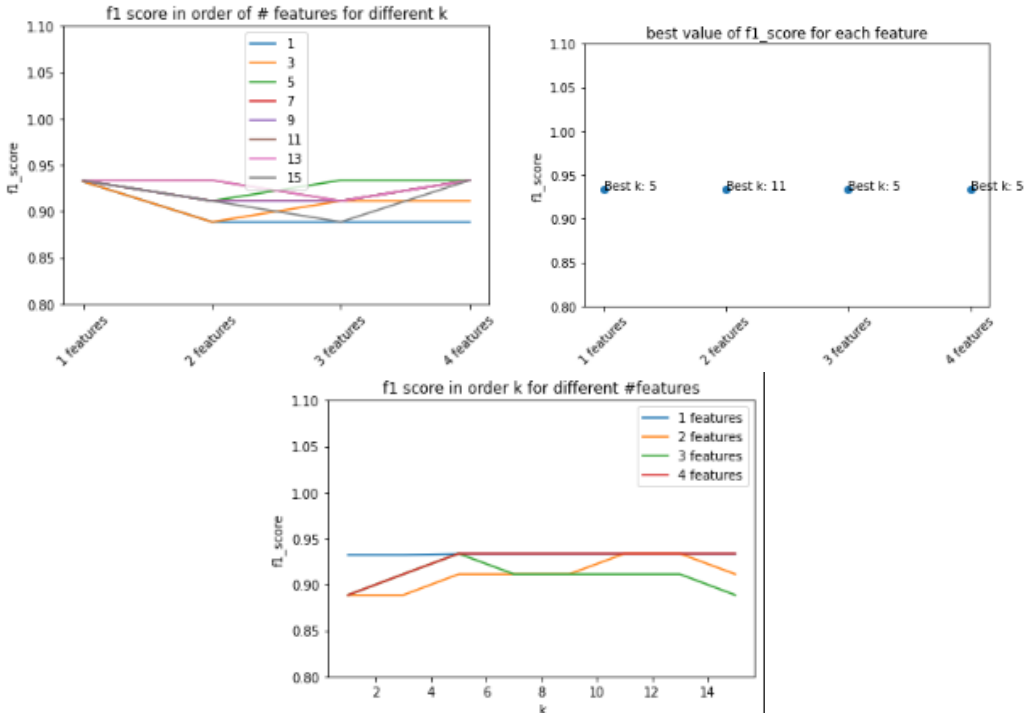


Figura 19: Plots Relief-F. Resultado médio após correr varias vezes o código(neste caso 20 vezes)

Forward Feature Selection

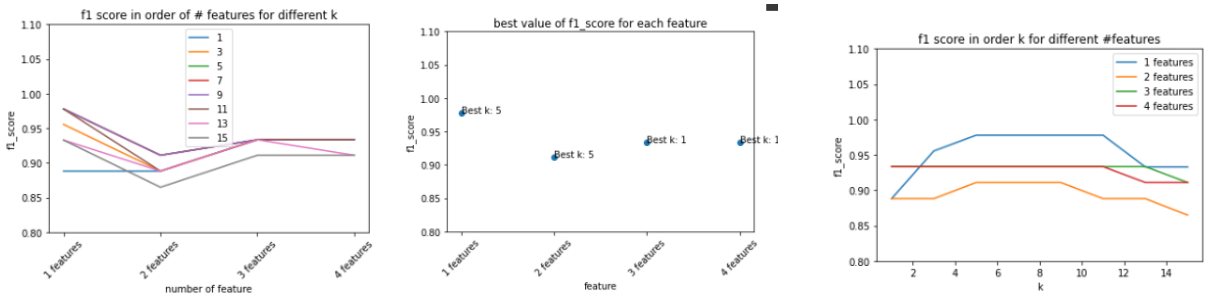


Figura 20: Plots Forward Feature Score. Resultado médio após correr varias vezes o código(neste caso 20 vezes)

Fischer Score

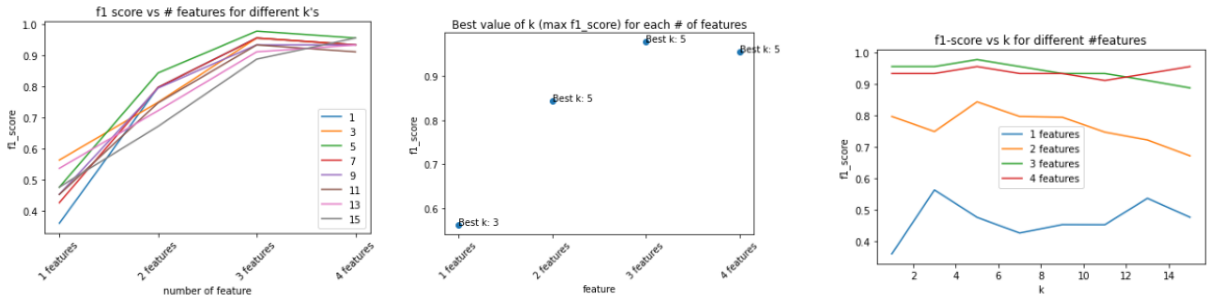


Figura 21: Plots Fischer score. Resultado médio após correr varias vezes o código(neste caso 20 vezes)

2.4

O unbalance não é significativo neste caso não é significativo. Como podemos observar obtemos resultados muito semelhantes aos obtidos na questão anterior.

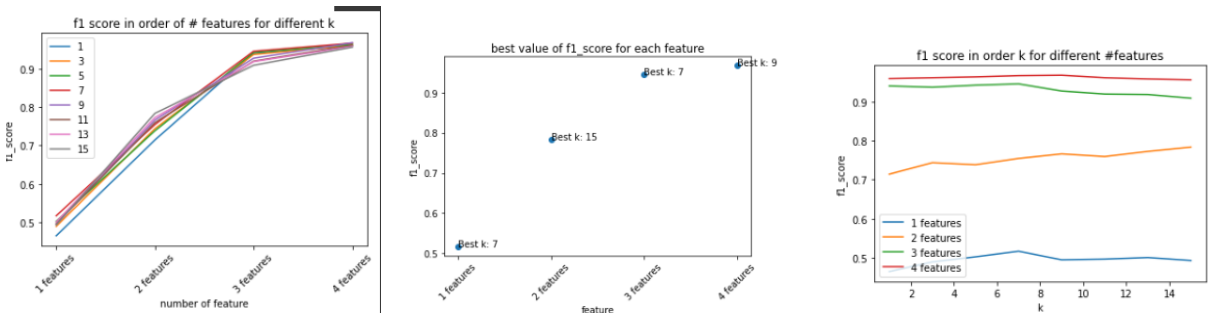


Figura 22: Plots Fischer score. Resultado médio após correr varias vezes o código(neste caso 20 vezes) para o dataset unbalanced

Ex 3

Cross validation

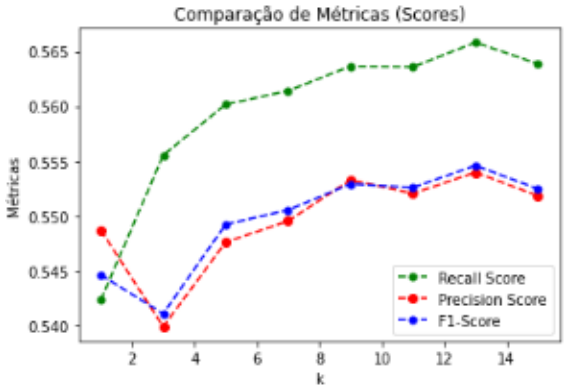


Figura 23: CV10-TT-KNN,O melhor valor de k é 13 com f1-score = 0.5497745. f1 score médio de 0.549

Train Only

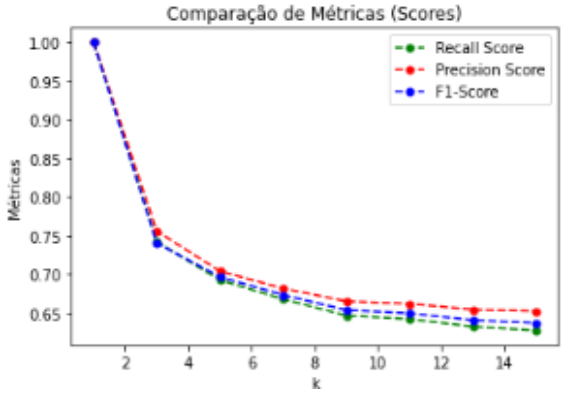


Figura 24: Train Only,O melhor valor de k é 1 com f1-score = 1. f1 score médio de 0.711

Train validation Test

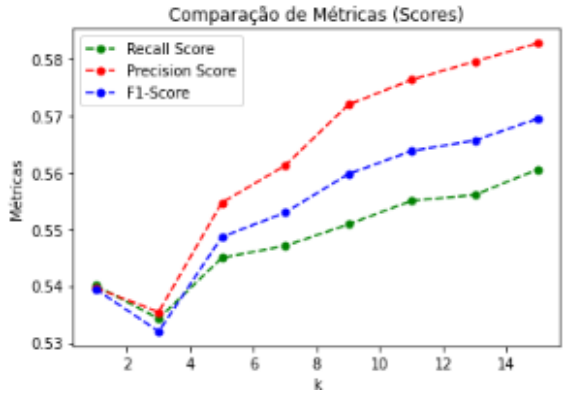


Figura 25: TVT, O melhor valor de k é 15 com f1-score = 0.554. f1 score médio de 0.549

Relief-F

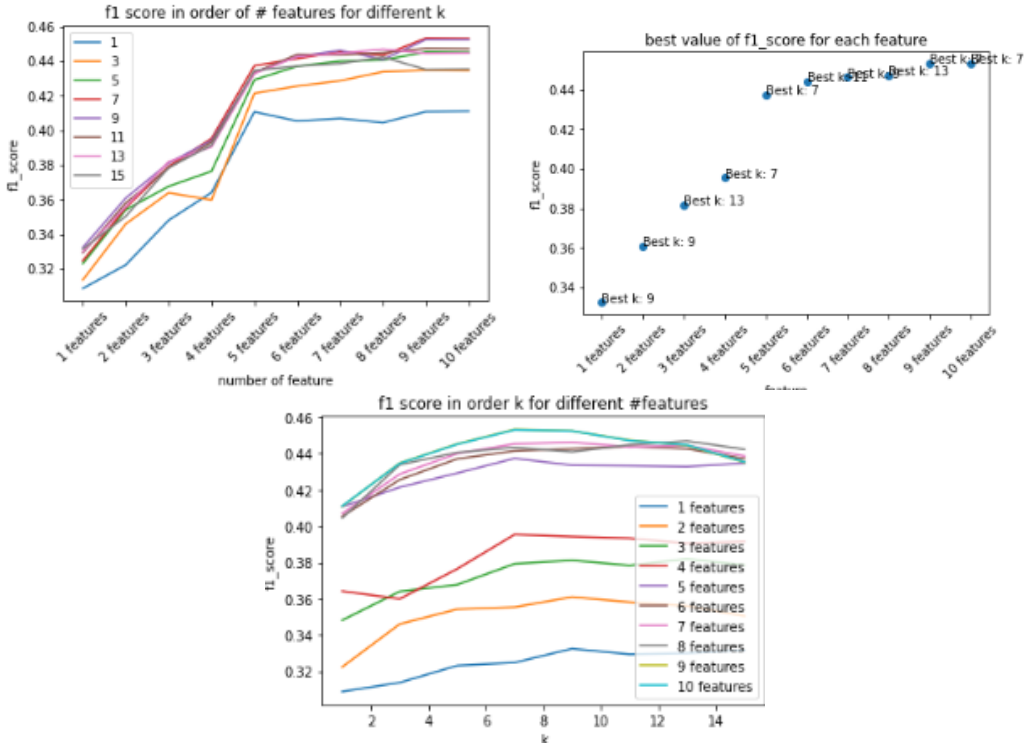


Figura 26: Plots kNN feature selection with Relief-F

Ex 4

Aprendizagem Fixa

As redes neuronais com aprendizagem fixa são treinadas de forma estática, ou seja, os seus pesos e *bias* são definidos previamente e não podem ser alterados durante o processo de aprendizagem. Isso significa que estas redes são capazes de realizar apenas tarefas de classificação, mas não são flexíveis o suficiente para se adaptar a novos conjuntos de dados ou tarefas.

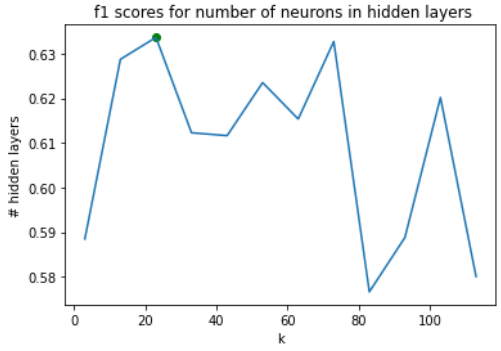


Figura 27: F1-scores para número de neurónios variável, tendo aprendizagem fixa.Neste caso obtivemos um valor médio de $F1 - score$ de 0.45

Aprendizagem variável

Por outro lado, redes neuronais com aprendizagem variável são capazes de se adaptarem a novos conjuntos de dados e tarefas de forma dinâmica, pois os seus pesos e *bias* são atualizados continuamente durante o processo de treino. Isso permite que essas redes sejam mais flexíveis e eficientes em problemas que exigem aprendizagem em tempo real, como reconhecimento de padrões e tomada de decisão.

.

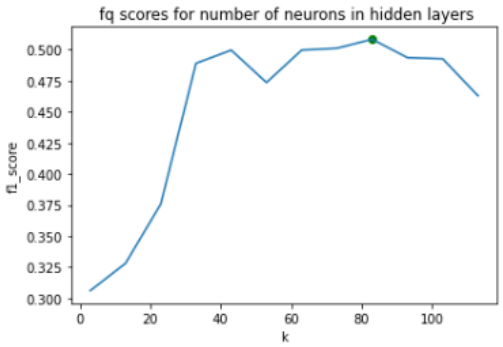


Figura 28: F1-scores para número de neurónios variável, tendo aprendizagem variável. Neste caso obtivemos um valor médio de $F1 - score$ de 0.617

Neste caso obtemos um melhor valor com mais iterações, mas por gestão de recursos apenas se conseguiu correr o código até k = 125.

Coeficiente de momento (EXTRA)

O coeficiente de momento é uma técnica utilizada para acelerar o processo de treino de redes neuronais, principalmente nas redes neuronais com aprendizagem variável. É um parâmetro adicionado à atualização dos pesos e *bias* das camadas da rede, que leva em consideração o desvio da última atualização e o desvio anterior. Isso permite que a rede se adapte mais rapidamente as novas informações e evite ficar presa em mínimos locais durante o treino.

Um dos principais problemas associados ao uso do coeficiente de momento em redes neuronais é o risco de otimização excessiva, *oevrfitting*. Isso ocorre quando o valor do coeficiente é muito alto, o que pode fazer com que a rede ignore informações importantes e se ajuste de forma exagerada aos dados de treino, levando, naturalmente a uma situação de *overfitting*. Outro problema é que o coeficiente de momento pode ser difícil de ajustar corretamente. É um parâmetro hiperbólico, o que significa que seu valor pode variar de forma muito rápida e imprevisível. Isso pode dificultar a escolha do valor ótimo para o coeficiente, o que pode afetar negativamente o desempenho da rede.

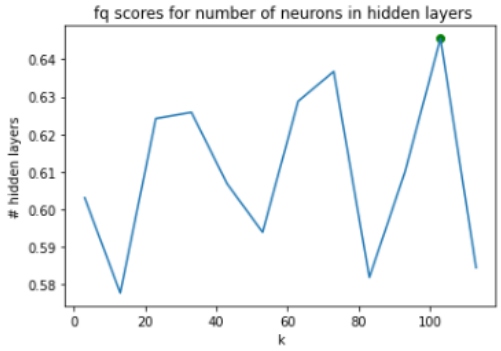


Figura 29: F1-scores para número de neurónios variável, tendo coeficiente de momento. Neste caso obtivemos um valor médio de $F1 - score$ de 0.699.

4.4

O KNN (K-Nearest Neighbors) é um método de classificação supervisionado que utiliza um algoritmo para identificar a que classe um novo dado de entrada pertence, com base em dados de treino previamente classificados. O KNN é baseado no pressuposto de que dados que estão próximos um do outro no espaço das features provavelmente pertencem à mesma classe.

Por outro lado, as redes neuronais são um tipo de modelo de machine learning que se baseiam num conjunto de algoritmos para criar um modelo que pode fazer previsões ou classificações com base em dados de treino. As redes neuronais são compostas por camadas de neurónios interconectados, que são treinados usando um algoritmo de otimização para aprender a realizar tarefas específicas.

Enquanto o KNN é um método simples que pode ser facilmente compreendido e implementado, as redes neuronais são muito mais poderosas e capazes de realizar tarefas complexas com alta precisão. No entanto, as redes neuronais também são mais difíceis de entender e requerem mais recursos computacionais para treino e inferência.

Em resumo, a principal diferença entre o KNN e as redes neuronais é que o KNN é um algoritmo de classificação simples baseado num conjunto de regras pré-definidas, enquanto as redes neuronais são modelos de machine learning mais poderosos que são treinados usando algoritmos de otimização para realizar tarefas específicas.

Observa-se que o melhor resultado das redes neuronais é descrito por 4.1), seguido de 4.3) e depois 4.2). De modo a melhorar o resultado das redes neuronais poderiamos variar a função de ativação (ReLU, SELU, ELU), poderíamos variar o número de epochs, batch sizes, o número de neurónios nas camadas escondidas e o próprio número de hidden layers. Por outro lado, o KNN é o método que obteve melhores resultados, é muito simples e requer o ajuste de apenas um hiperparâmetro (o valor de k), enquanto o treino da rede neural envolve muitos hiperparâmetros que determinam o tamanho e a estrutura da rede e o procedimento de otimização.
`https://www.tasq.ai/glossary/algorithm/`

Ex 5

Com auxilio de :
`https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6`
Implementamos uam rede neuronal simples em que aplicamos primeiramente o dataset *iris* e em seguida aplicamos as funções desenvolvidas por nos ao dataset das features humanas.
No dataset humano foi feito o estudo do comportamento da rede em função do numero de épocas de treino e do numero de neurónios na *hidden layer*.

O código e gráficos referentes a esta questão encontram-se no ficheiro .ipynb respetivo a parte B.