FCTUC – DEI
Mestrado em Engenharia Informática

Project Assignment

Chasing Phishing URLs

Reconhecimento de Padrões

Vasco Rodrigues Nr.º 2024169097

10/05/2025

Índice

Índice Figuras

1. Introdução

Atualmente, enquanto navegamos na internet, estamos em constante perigo e sujeitos a tentativas de roubo da nossa informação pessoal sem a nossa permissão. Por esta razão, o trabalho proposto e realizado no âmbito da cadeira de Reconhecimento de Padrões apresenta uma grande importância para assegurar a nossa segurança. O objetivo do mesmo é a identificação de *URLs* maliciosos através de um modelo de classificação.

Para garantir que esta classificação seja a melhor possível, dados os dados disponíveis, serão realizadas diversas operações, como *feature selection*, *cleaning* e *normalization of data*, e *dimensionality reduction*. Além disso, serão utilizados diversos modelos, tendo os seus resultados comparados entre si, de forma a garantir que seja escolhido o modelo que melhor se comporta em dados nunca antes vistos.

Dentro das operações de *feature selection*, serão utilizados os testes como *Kruskal-Wallis*, que permitirá verificar como as características se relacionam com a variável *target*, podendo assim escolher aquelas que são mais discriminativas e excluir as que menos ou ROC Curves para verificar se as features são realmente. Será também verificada a matriz de correlação para identificar dados altamente correlacionados e, consequentemente, remover características redundantes.

Já nas operações de *dimensionality reduction*, será testada a utilização de *PCA* (*Principal Component Analysis*), que procura representar os dados num novo eixo, de forma a obter uma melhor representação e uma mais fácil discriminação dos dados.

Os modelos que serão utilizados são: *Minimum Distance Classifier*, tanto com *Euclidean Distance Discriminant* como com *Mahalanobis Distance Discriminant*, *Fisher LDA* (*Linear Discriminant Analysis*), *Bayes Classifier*, *K-NearestNeighbors* e *Support Vector Machine* (Tanto com kernel linear e não linear). Ambos os modelos serão treinados e testados no mesmo *dataset*, utilizando *cross-validation*, para garantir resultados de treino/teste mais coerentes. As métricas utilizadas para avaliar estes modelos serão: *sensitivity*, *specificity*, *F1-score* e *ROC Curves*. A *accuracy* não será considerada, pois, como é referido no enunciado do próprio projeto, este é um *dataset* desequilibrado (*imbalanced dataset*), e a *accuracy*, no seu cálculo, não tem em consideração este facto.

Por fim o modelo que apresentar melhores resultados terá o *dataset* dividido em dois onde na primeira metade ele será treinado e testado para estes dados e por fim na segunda metade do *dataset* será feita a sua validação, simulando assim um cenário real podendo verificar como o modelo se comportaria.

2. Implementação

Antes da especificação da implementação dos modelos é importante mencionar que as características categóricas e binárias presentes no *dataset* foram totalmente removidas isto porque os modelos dados até agora apenas lidam com dados contínuos e são próprios para esse tipo de dados.

2.1. Normalização

O processo de normalização consiste na conversão dos valores de dados contínuos para uma escala comum, de forma a garantir que todas as características contribuam de forma equilibrada para o processo de classificação.

Neste trabalho foram considerados dois métodos de normalização: *min-max* e *z-score*.

2.1.1 Min-max

O método *min-max* transforma os dados de forma que estes se situem no intervalo 0 e 1. Para isso, utiliza o valor mínimo e máximo de cada característica. O processo é o seguinte:

1. Subtrai-se o valor mínimo da característica a cada valor dos dados.
2. Divide-se o resultado pela diferença entre o valor máximo e o mínimo da característica.

A normalização *min-max* é expressa pela fórmula:

$$x_{normalizado} = \frac{x - \min(X)}{\max(X) - \min(X)}$$

Onde x é o valor original, min(X) é o valor mínimo da característica, e max(X) é valor máximo.

Neste método *outliers* podem ter uma grande influência, pois eles afetam diretamente os valores de mínimo e máximo

2.1.2 Z-score

O método *z-score*, transforma os dados de forma que tenham uma média de 0 e um desvio padrão de 1. Para isso, utiliza a média e o desvio padrão de cada característica. O processo é o seguinte:

1. Subtrai-se a média da característica a cada valor dos dados.
2. Divide-se o resultado pelo desvio padrão da característica.

A normalização *z-score* é expressa pela fórmula:

$$x_{normalizado} = \frac{x - \mu}{\sigma}$$

Onde x é o valor original, μ é a média da característica e σ é o desvio padrão.

Este método é mais robusto à presença de *outliers*, pois a média e o desvio padrão são menos sensíveis a valores extremos do que o mínimo e o máximo. No entanto, assume que os dados seguem uma distribuição aproximadamente normal, o que nem sempre é verdade.

2.2. Kruskal-Wallis

Este teste permite verificar quais características são mais discriminativas em relação a uma dada classe, ou seja, quais *features* apresentam diferenças significativas nas suas distribuições entre as diferentes classes.

Para a sua implementação foi utilizada o módulo *kruskal* da biblioteca *scipy* para o cálculo do *H-value*, no entanto o seu cálculo é feito da seguinte forma:

1. É selecionado uma *feature;*
2. Para cada classe do target são atribuídos *ranks* por ordem crescente em relação aos valores da *feature* atual;
3. É calculada a média do *rank* para cada classe da variável target*;*
4. É calculada a média de todos os *ranks*;
5. É calculado o *H-value* que tem a seguinte formula

$$H = \frac{12}{n(n+1)} \sum_{i=1}^{c} n_i \, (R_i - \bar{R})^2$$

Onde n = número total de observações, c é o número de classes da variável target e ni é o tamanho de cada grupo.

A *feature* que obtiver o *H-value* maior é aquela que melhor discrimina os dados. Isto pode então ser utilizado para *feature selection*.

Como este teste apenas verifica o poder discriminativo de uma característica é necessário descartar *features* que sejam altamente correlacionadas para isso foi feito o uso da matriz de correlação e serão removidas aquelas que apresentam valores muito altos entre si novamente, *feature selection*

2.3. ROC Curves

As ROC curves são utilizadas para avaliar o desempenho de um modelo de classificação. Elas mostram a relação entre:

- *Sensibilidade*
- 1-*Especificidade*

Quanto mais a curva se aproxima do canto superior esquerdo, melhor o modelo distingue as classes, *Figura 2-1*.

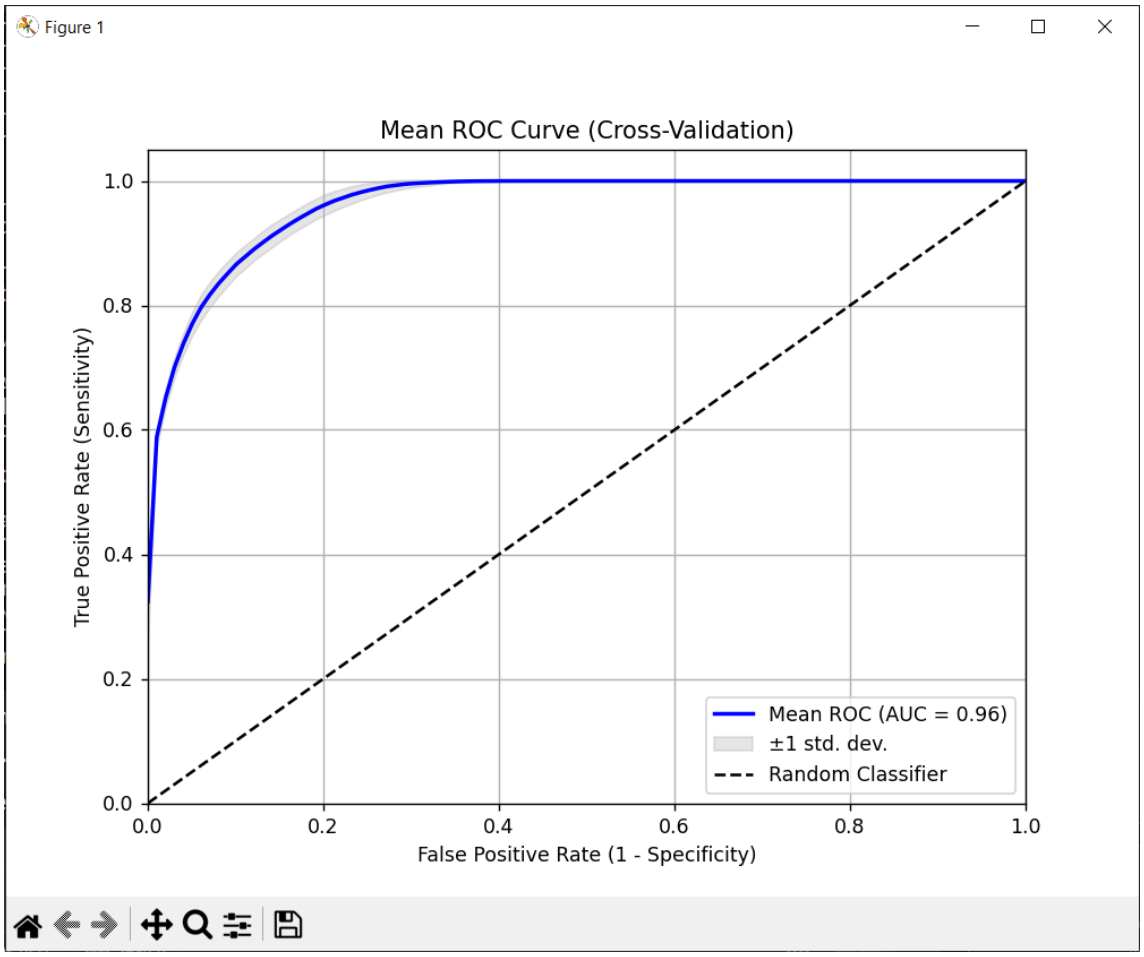


Figura 2-1 - Exemplo Roc Curve

Esta curva é gerada ao ajustar o *threshold* do modelo, entre 0 e 1 para obter os diferentes valores de especificidade e sensibilidade. Tendo esta curva será possível calcular a AUC (Area under the curve) que quanto maior for esta área melhor será a capacidade do modelo de discriminar a variável target.

Utilizando apenas uma *feature* como "preditor" será possível calcular o quão determinante esta é podendo assim fazer a respetiva *feature selection.*

2.4. PCA

O *Principal Component Analysis* (*PCA*) tem como objetivo principal descobrir um novo conjunto reduzido de *features* com menos redundância, com perda mínima de informação, esta preservação de informação é medida em termos de variância. Ele projeta os dados em direções onde os dados apresentam maior variação.

Para isto o PCA segue os seguintes Passos:

1. Normalização dos dados para ter média zero e desvio padrão de 1. Para garantir que a variância seja comparável entre as características
2. Cálculo da matriz de covariação.

3. Cálculo dos *eigenvectors* e *eigenvalues*, pois estes representam as direções no espaço dos dados e a quantidade de variância capturada por cada componente principal, respetivamente.
4. Ordenação dos *eigenvectors* e *eigenvalues* em ordem decrescente
5. Seleção dos componentes principais, foram implementados dois métodos:
    a. *Kaiser criterium*: Selecionar o eigenvalue antes do plot cair para baixo de 1, *Figura 2-2*
    b. *Scree test*: Selecionar o eigenvalue que estabilizou a curva, *Figura 2-2*

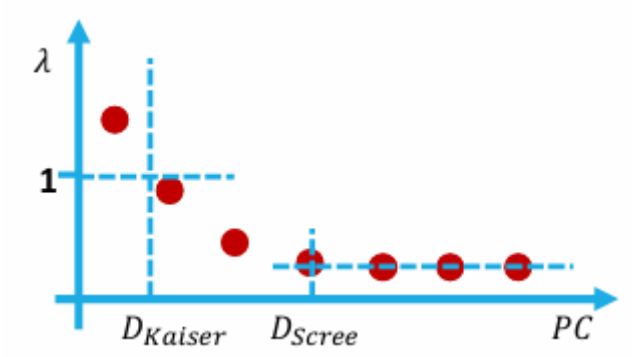


Figura 2-2 - Demonstração de kaiser criterium e scree test

Desta forma serão selecionadas um novo conjunto reduzido de *feature* que se insere no âmbito da *feature reduction/dimensionality reduction*.

2.5. Minimum Distance Classifier

Neste classificador uma amostra é classificada como a classe cuja distribuição conhecida ou estimada mais se assemelha com amostra em questão. Esta medida de semelhança é a distância no espaço de distribuição, ou seja, calcula-se a distância entre a amostra e as representações das diferentes classes (usualmente a médias dos dados delas), atribuindo-se a amostra à classe que estiver mais "próxima".

Este tipo de classificador baseia-se na ideia de que as amostras de uma mesma classe tendem a estar próximas umas das outras no espaço de características, enquanto amostras de classes diferentes tendem a estar mais afastadas. Para isto, é necessário definir métricas de distância. Entre várias existentes as mais comuns são *Euclidean Distance* e *Mahalanobis Distance*:

- *Euclidean Distance*: Mede a distância em “linha reta” entre dois pontos no espaço. E por consequência considera que as distribuições das classes são esféricas.
- *Mahalanobis Distance*: Leva em consideração a correlação entre as características e as variações nas diferentes direções do espaço. Esta é particularmente útil quando as distribuições das classes não são esféricas ou têm escalas diferentes

2.5.1 Euclidean distance discriminant

Como mencionado anteriormente este tipo de discriminante considera que as classes e as suas distribuições são esféricas pela forma como é calculada a sua distância. A distância euclidiana mede a distância "em linha reta" entre dois pontos no espaço de características, sem considerar a correlação entre as variáveis ou as diferenças de escala.

Esta abordagem pode não ser a melhor opção quando as distribuições das classes são alongadas ou têm orientações diferentes, pois ignora a estrutura de covariância dos dados.

Tendo em conta esta informação para a sua implementação é necessário:

1. É calculada a média de cada classe da variável *target* com base nos dados
2. É feita a classificação de novos pontos calculando a distância euclidiana entre as médias da classe. A distância euclidiana é dada pela fórmula:
    a. $distancia = \sqrt{\sum(x_1 - x_2)^2}$

2.5.2 Mahalanobis distance discriminant

Ao contrário do discriminante anterior, o *Mahalanobis distance discriminant* leva em consideração as variações nas diferentes direções do espaço. Isto é feito incorporando a matriz de covariação no cálculo da distância, o que permite ajustar a métrica à forma e à orientação da distribuição dos dados.

Esta característica torna o *Mahalanobis distance discriminant* mais flexível e robusto, especialmente quando as classes têm distribuições elípticas ou quando as características estão correlacionadas. No entanto, o cálculo da matriz de covariação pode ser computacionalmente mais exigente.

Tendo em conta esta informação para a sua implementação é necessário:

1. É calculada a média de cada classe da variável *target* com base nos dados
2. É calculada a inversa da matriz de covariação para cada classe
3. É feita a classificação de novos pontos calculando a distância *mahalanobis* entre cada ponto e a média das classes, a distância mahalanobis é dada pela fórmula:
    a. $D_M(p, \mu) = \sqrt{(p - \mu)^T C^{-1} (p - \mu)}$

Onde p é o vetor que representa o ponto no espaço, μ é o vetor que representa a média das classes da variável target, C é a matriz de covariação.

2.6. Fisher LDA

Enquanto técnicas como o *PCA* (*Principal Component Analysis*) se focam em encontrar uma representação dos dados num subespaço linear de dimensão reduzida que captura as maiores variâncias nos dados, o *Fisher's Linear Discriminant Analysis* (*LDA*) tem como objetivo maximizar a separabilidade entre classes.

O *PCA* não tem em conta a informação das classes, ou seja, não leva em consideração as *labels* dos dados ao determinar as direções de máxima variância. Embora isso seja útil para representação dos dados, as direções de maior variância nem sempre são as melhores para discriminar duas classes distintas.

O *LDA*, por outro lado, procura encontrar uma projeção linear dos dados num subespaço que maximize a separação entre as classes. Para isso, utiliza um critério de otimização baseado na razão de *Fisher (Fisher's ratio)*, que mede a relação entre a variância entre classes (*inter-class*) e a variância dentro de cada classe (*intra-class*). Ao maximizar esta razão, o *LDA* garante que as

projeções dos dados no novo subespaço mantenham as classes tão distantes quanto possível, enquanto minimiza a dispersão dentro de cada classe.

Por esta razão, não faz sentido aplicar métodos como a *PCA* antes de utilizar o classificador *Fisher LDA*. Isto porque a *PCA* representa os dados num novo eixo que maximiza a variância, mas ignora a informação das classes. Como resultado, esta transformação pode dificultar a tarefa do *Fisher LDA*, que procura encontrar um eixo que maximize a separação entre as classes. Por outras palavras a projeção inicial feita pelo PCA pode destruir as diferenças que existem entre as classes, reduzindo a eficácia do Fisher LDA na discriminação dos dados.

Para a sua implementação foi utilizado o módulo *LDA* da biblioteca *scipy*, no entanto este é implementado da seguinte forma:

1. Calcular as médias das classes da variável target
2. Calcular a matriz de dispersão dentro das classes, que é dada pela seguinte formula:
    a. $S_w = S_1 + S_2 = \sum_{k=1}^{2}(x - m_k)(x - m_k)^T$
3. Calcular o vetor de projeção w, dado pela formula:
    a. $w = S_w^{-1}(m1 - m2), m1\ e\ m2\ são\ as\ médias\ das\ classes$
4. Projetar os dados no novo espaço através de w
    a. $y = w^T x$
5. Calcular o Hiperplano de decisão no novo espaço
    a. $y = x = \frac{w^T m1 + w^T m2}{2}$
6. Classificar dados segundo este hiperplano, *Figura 2-3*

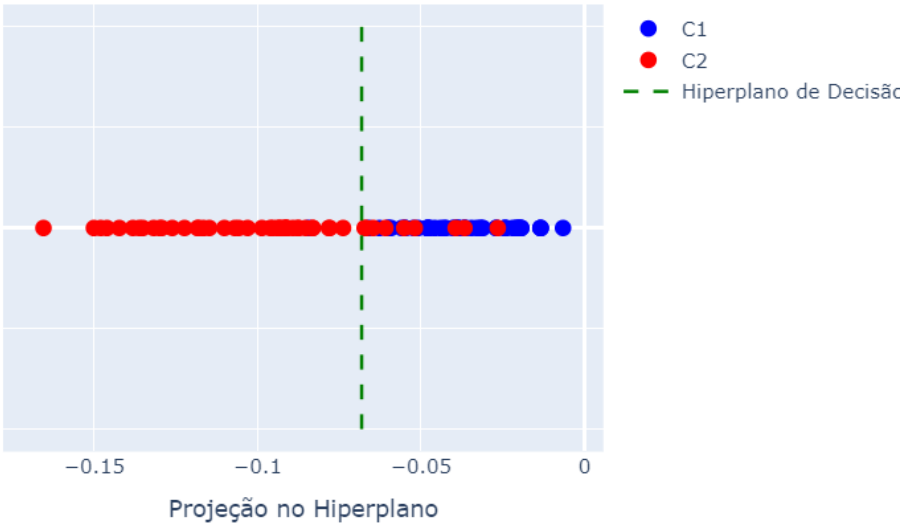


Figura 2-3 - Hiperplano de decisão do Fisher LDA

2.7. Bayes Classifier

Este classificador tem como objetivo recomendar ações que minimizam a soma total de risco esperado, ou seja, ele quantifica os *tradeoffs* entre varias decisões de classificação através de probabilidades e custo associados a ela.

Este classificador, no caso deste projeto, baseia -se em:

- probabilidade à priori, que é probabilidade de uma classe antes de observar sequer os dados
- Verosimilhança Gaussiana, que modela a distribuição dos dados em cada classe, como as nossas features são continuas assume-se uma distribuição normal multivariada:
  - $p(x|\omega_i) = \frac{1}{(2\pi)^{d/2}|C|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)}$
  - Onde:
    - d - distância mahalanobis
    - C - Matriz de covariância
    - μ - vetor médio da classe
- A decisão final é tomada tendo em conta as probabilidades à posteriori ponderadas no passo anterior.

2.8. K-Nearest Neighbors

O k-NN é um método que classifica dados "*unlabelled*" com base na similaridade destes com outros exemplos presentes no *dataset.*

Esta similaridade é a proximidade dos dados, sendo utilizada usualmente para calcular isto a distância euclidiana.

O algoritmo k-NN passa por:

1. Na fase de treino deve ser guardado o vetor de características e as suas labels dos dados de treino
2. Na fase de classificação, a amostra é representada como um vetor no espaço de característica
3. São calculadas as distâncias deste novo vetor a todos aqueles já guardados, e são selecionados os k amostras mais próximas
4. Dentro destes k vizinhos, para classificar o novo vetor, será verificada a classe mais comum e atribuída ao novo vetor.

No caso deste projeto foi utilizado o módulo *KNeighborsClassifier* presente na biblioteca *scipy*.

2.9. SVM

A principal ideia das *Support Vector Machines* é definir um hiperplano ótimo que:

- Minimize o erro de treino
- Maximize a separação entre classes

Para problemas que não são linearmente separáveis, permite alguns erros de classificação

Mapeia implicitamente os dados para um espaço dimensional superior onde se tornam linearmente separáveis, através de funções kernel, como foi usado no projeto, "RBF".

No caso de ser um problema separável linearmente é necessário primeiro resolver o primal problema dado por

$$\begin{cases} minimize\phi(w) = \frac{1}{2}||w||^2 \\ s.t.\ y_i(w^T x_i + b) \geq 1 \forall x_i \end{cases}$$

Em seguida resolver o problema dual de otimização onde queremos maximizar o seu valor para obtermos os multiplicadores de Lagrange ótimos e consequentemente o vetor ótimo de pesos:

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {x_i}^T x_j$$

Uma vez tendo os valores para α, podem ser obtidos os pesos e biases para a função de decisão

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$b = -\frac{1}{2} w^T (x_p + x_m)$$

x_p *e* x_m *são quaisquer support vectors positivos e negativos resptivamente*

No caso de um problema não linear o *primal problema* torna-se:

$$\begin{cases} minimize\phi(w) = \frac{1}{2}||w||^2 + C\sum \xi_i \\ s.t.\ y_i(w^T x_i + b) \geq 1 - \xi_i \forall x_i \ and \ \xi_i \geq 0 \end{cases}$$

C é o parâmetro que influência a margem, quanto maior, maior a penalização por classificações erradas.

ξ introduz *slack variables,* ou seja, permite que alguns padrões caiam dentro da margem ou no lado oposto, mas penaliza-os.

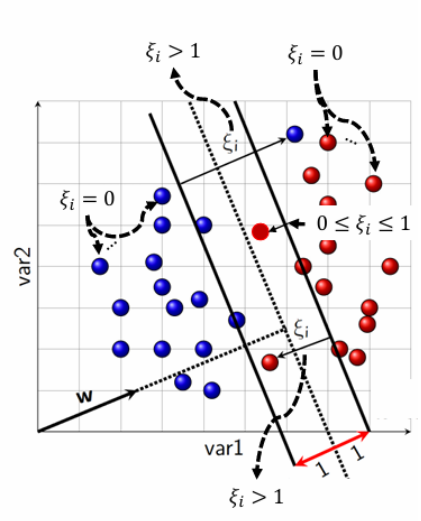


Figura 2-4 - Slack variables

O *dual problema* torna-se:

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (f(x_i)^T f(x_j))$$

Uma vez tendo os multiplicadores de Lagrange podem ser calculados os pesos ótimos e consequentemente os *biases*:

$$w = \sum_{i=1}^{n} \alpha_i y_i f(x_i)$$

$$b = -\frac{1}{2} w^T \left(f(x_p) + f(x_m)\right)$$

A decision fuction torna-se:

$$d = \sum_{SVs} \alpha_i y_i K(x_i, x) + b$$

Onde "K" é o *inner-product kernel*, que pode tomar várias funções no caso deste projeto foi apenas usada a RBF, que foi aquela testada em aula.

2.10. Cross Validation

Para os obter dados de avaliação dos modelos mais coerentes foi utilizado *cross-validation*, que é uma técnica que parte o *dataset* em k partições, onde 1 delas é escolhida para teste e as restantes k-1 são utilizadas para treino, este processo é repetido k vezes. Isto permite obter resultados médios e, portanto, mais confiáveis.

3. Resultados

Tendo todas as implementações feitas basta agora fazer os testes para verificar qual combinação de classificadores se melhor comporta.

Estas têm como objetivo tentar minimizar o erro na classificação de novos dados e os resultados estarão nos subcapítulos a baixo.

Como poderá no teste a seguir a redução de *features* e de dimensionalidade acaba por piorar de forma consistente o desempenho geral dos modelos, e por tanto a partir do capítulo *5* estes teste não serão incluídos pois não acrescentaram novas conclusões.

3.1. Kruskal Wallis

A aplicação do teste de *kruskal wallis*, que nos permitirá verificar quais características melhor discriminam a variável *target*.

E os resultados deste teste podem ser vistos na seguinte *Figura 3-1*.

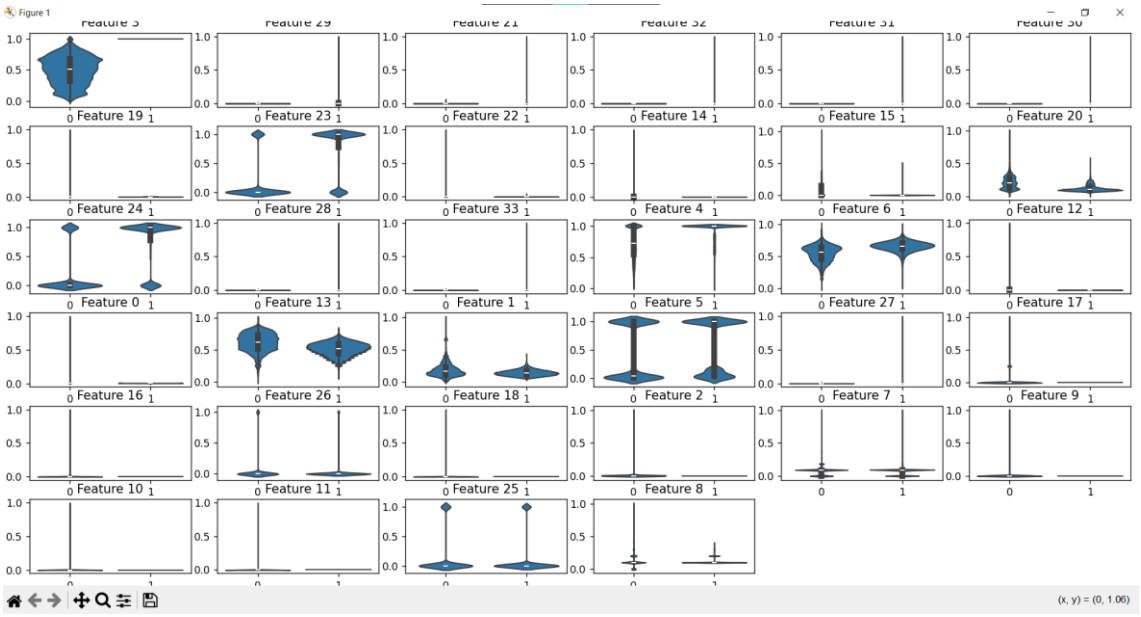


Figura 3-1 - Resultados Kruskal wallis

Como é possível ver na figura acima existem características que discriminam bem as classes do target e outras que simplesmente não fazem qualquer diferença sendo a que melhor discrimina a *feature* 3, então devem ser mantidas todas as *features* antes da *feature* 13, os resultados dos valores h estão na seguinte *Figura 3-2*, ao visualizar estes valores é possível verificar que a escolha de manter apenas as *features* antes da 13 é suportada pois há um grande salto no *H-value* da *feature* 1 para a 13.

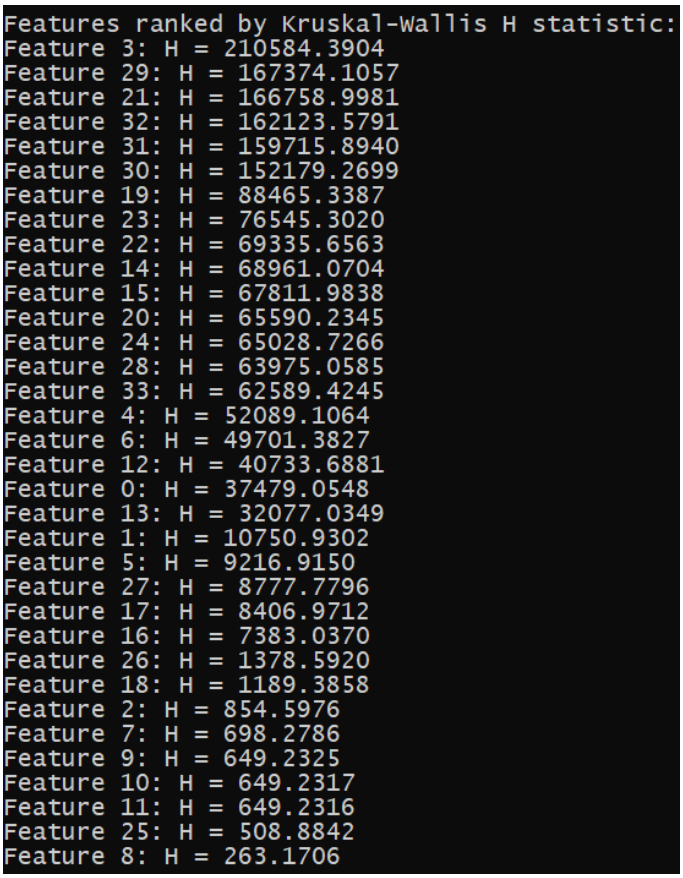


Figura 3-2 - Ranking of H-value

Através da matriz de correlação é possível visualizar *features* que são altamente correlacionadas e portanto redundantes para os classificadores, e então removê-las por não acrescentarem nada à previsão de novos dados. Ao visualizar a *Figura 3-3* é possível verificar que existem sim características bastante correlacionadas e então irão ser removidas aquelas com um nível de correlação superior a 80%, ou seja, as *features* 12, 14,16, 19 e 23

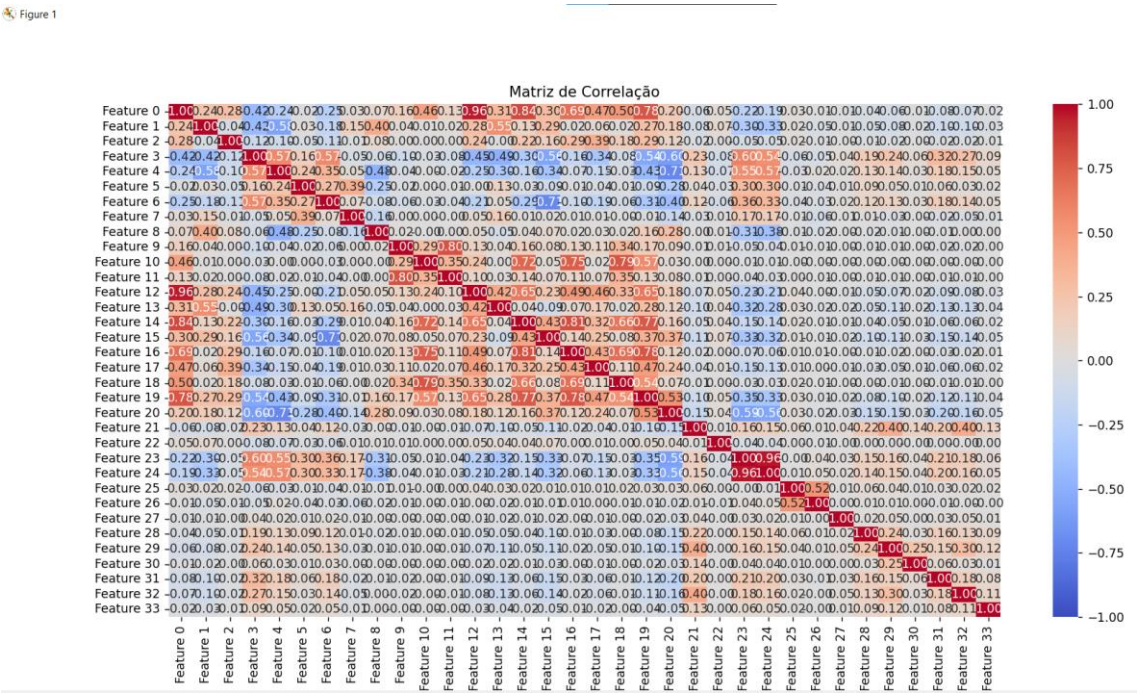


Figura 3-3 - Matriz de correlação

Fazendo agora uma pequena comparação do antes e após a seleção e remoção de features através de estes dois resultados é possível verificar que há uma leve melhoria nos resultados de classificação visível nas *Figura 3-4* e *Figura 3-5*.

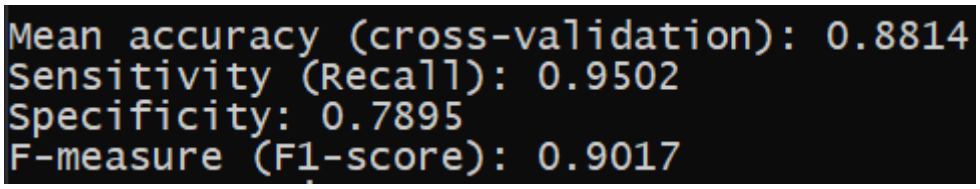


Figura 3-4 - Antes da remoção de variáveis altamente correlacionadas

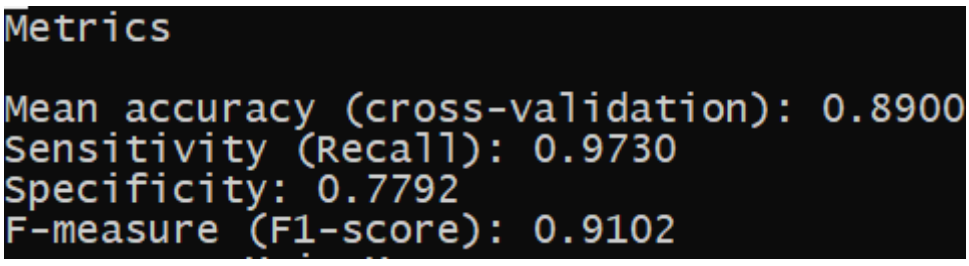


Figura 3-5 - Após a remoção de variáveis altamente correlacionadas

3.2. ROC Curves (feature selection)

Como foi dito em cima o método como as ROC Curves avaliam a discriminação da variável *target* por parte dos classificadores pode também ser aplicada às suas *features* para verificar o mesmo, ou seja, o quão bem elas discriminam a *target*, baseando-se na *AUC*, em baixo encontram-se os resultados obtidos para cada uma das diferentes variáveis, *Figura 3-6* e *Figura 3-7*.

E como é possível verificar os valores não são exatamente iguais aos do *Kruskal Wallis* apesar de serem ainda idênticos, todos os valores perto de 0.5 (Mesmo que um classificador aleatório) ou inferiores podem ser removidos.

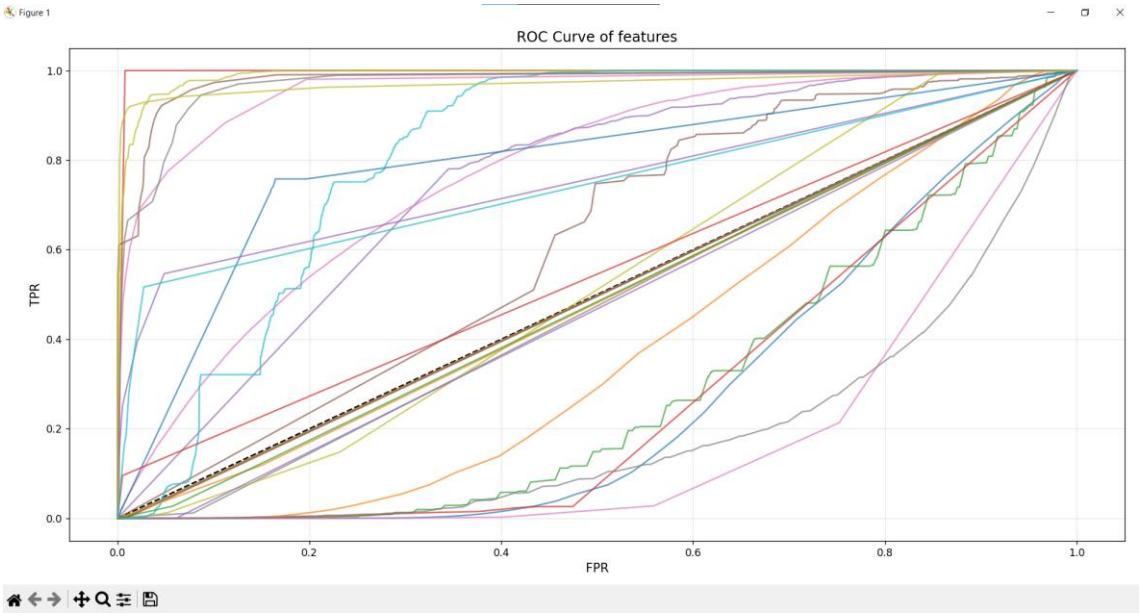


Figura 3-6 - Gráfico com curvas ROC para cada feature

```
Features ranked by ROC AUC score:
1 - Feature 3: AUC = 0.9961
2 - Feature 18: AUC = 0.9903
3 - Feature 25: AUC = 0.9796
4 - Feature 28: AUC = 0.9725
5 - Feature 27: AUC = 0.9708
6 - Feature 26: AUC = 0.9576
7 - Feature 19: AUC = 0.8163
8 - Feature 20: AUC = 0.7923
9 - Feature 6: AUC = 0.7679
10 - Feature 24: AUC = 0.7546
11 - Feature 29: AUC = 0.7454
12 - Feature 4: AUC = 0.7439
13 - Feature 5: AUC = 0.6089
14 - Feature 23: AUC = 0.5452
15 - Feature 8: AUC = 0.5146
16 - Feature 9: AUC = 0.4976
17 - Feature 10: AUC = 0.4976
18 - Feature 11: AUC = 0.4976
19 - Feature 2: AUC = 0.4968
20 - Feature 15: AUC = 0.4956
21 - Feature 22: AUC = 0.4848
22 - Feature 21: AUC = 0.4840
23 - Feature 7: AUC = 0.4738
24 - Feature 14: AUC = 0.4696
25 - Feature 1: AUC = 0.3756
26 - Feature 12: AUC = 0.2849
27 - Feature 13: AUC = 0.2739
28 - Feature 0: AUC = 0.2676
29 - Feature 17: AUC = 0.1924
30 - Feature 16: AUC = 0.1762
```

Figura 3-7 - Valores de AUC de todas as features

3.3. Minimum Distance classifier

3.3.1 Euclidean Distance Discriminant

O primeiro teste será a classificação de dados sem qualquer tipo de *feature selection* ou *reduction*, para verificar como os modelos se comportam sem o devido tratamento de dados prévio.

Novamente é de notar que todas as features categóricas e binárias foram removidas antes de quaisquer testes devido aos modelos aprendidos não considerarem este tipo de dados.

Os resultados podem então ser observados na *Figura 3-8*.

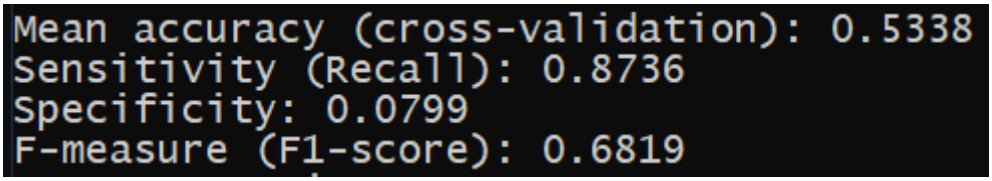


Figura 3-8 - Resultados teste MDC-EDD sem tratamento de dados

Tendo em conta estes resultados é possível concluir o seguinte:

- *Accuracy*: com um resultado de 53.38% isto indica que o modelo acertou cerca de metade das suas previsões. Isto sugere que o modelo não está a ter uma boa performance
- *Sensitivity*: com um resultado de 87.36% isto indica que a maioria dos exemplos de classe positiva o modelo está a acertar
- *Specificity*: com um resultado de 7.99% isto indica que o modelo classifica como positivos a grande maioria de casos negativos
- *F*-score*:* Com um resultado de 68.19% isto indica que o modelo tem um desempenho moderado. Apesar de tudo é um resultado enganoso por causa dos resultados obtidos nos testes *sensitivity* e *specificity*.

O próximo teste será feito com o classificador e com uma normalização através de *z-score*, e os resultados poderão ser visualizados abaixo na *Figura 3-9*.

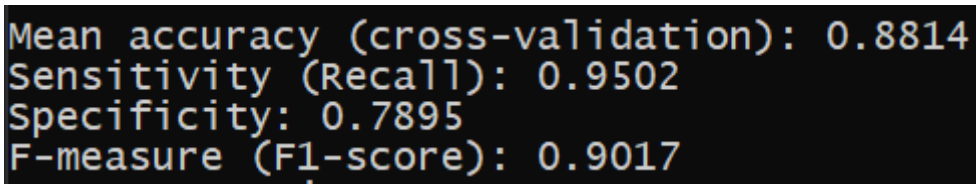


Figura 3-9 - Resultados teste MDC-EDD com normalização z-score

Tendo em conta os resultados é possível concluir o seguinte:

- *Accuracy*: com um resultado de 88.14% isto indica que o modelo acertou grande parte das previsões que fez;
- *Sensitivity*: com um resultado de 95.02% isto indica que praticamente todos os exemplos de classe positiva o modelo está a acertar;
- *Specificity*: com um resultado de 78.95% isto indica que o modelo consegue classificar bem a classe negativa;

- *F*-score*:* Com um resultado de 90.17% isto indica que o modelo tem um desempenho muito bom. Apesar de tudo continua a ser um resultado um pouco *biased* pela *sensitivity*, pois esta é muito alta e a classificação de casos negativos não é tão boa quanto ela

Tendo em conta estes novos resultados a normalização através de z-score teve um impacto muito positivo no classificador, que conseguiu discriminar as classes de uma melhor forma.

Isto acontece, pois, a distância euclidiana é muito sensível à escala dos dados. Então se uma característica tem valores muito maiores que outras elas acabam por contribuir mais para o valor final da distância, fazendo com que a classificação dos dados seja errada, a normalização permitiu corrigir isso.

O próximo teste será com para além da normalização dos dados a aplicação do teste de *kruskal wallis*, que nos permitirá verificar quais características melhor discriminam a variável *target*.

E os resultados deste teste são visíveis na *Figura 3-10*

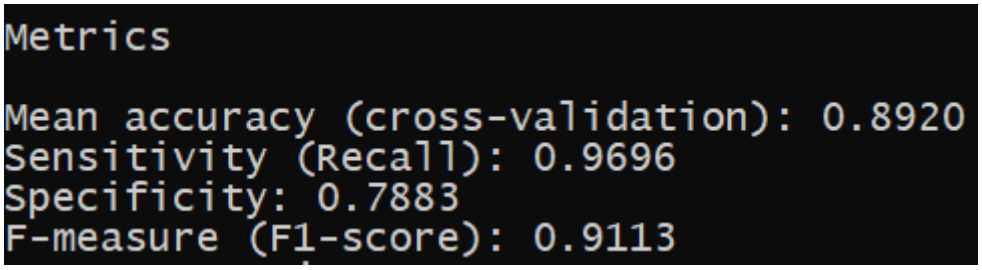


Figura 3-10 - Resultados teste MDC-EDD com normalização e kruskal wallis

Com estes resultados é possível concluir o seguinte:

Os resultados foram bastante idênticos ao anterior sendo apenas ligeiramente melhores, isto significa que ao remover as variáveis altamente correlacionadas não foi perdida qualquer informação, mas também não foi ganha muito mais.

Em seguida será feito um teste com a normalização dos dados e a redução de features através dos AUC scores das mesmas e foram então obtidos os seguintes dados que estão presentes na *Figura 3-11*.

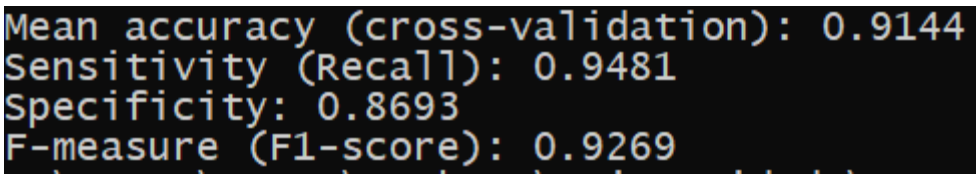


Figura 3-11 - Resultados teste MDC-EDD com normalização e ROC curve (feature reduction)

É possível verificar que apesar de ligeiro o uso de ROC curves para a redução de características tem uma melhor performance que o próprio *kruskal wallis*, alem de ser mais fácil e intuitivo de perceber quais features são ou não relevantes.

Em seguida e o último teste será a combinação de todos os métodos, ou seja, normalização, *kruskal wallis* e *PCA kaiser* e *scree*, e estes resultados estão presentes nas *Figura 3-12* e *Figura 3-14*, o número de componentes principais escolhidos para o *scree test* foi com base no *plot* na *Figura 3-13* e foi escolhido 11.

```
Mean accuracy (cross-validation): 0.8750
Sensitivity (Recall): 0.9499
Specificity: 0.7748
F-measure (F1-score): 0.8969
```

Figura 3-12 - Resultados teste MDC-EDD com normalização, kruskal wallis e PCA (kaiser)

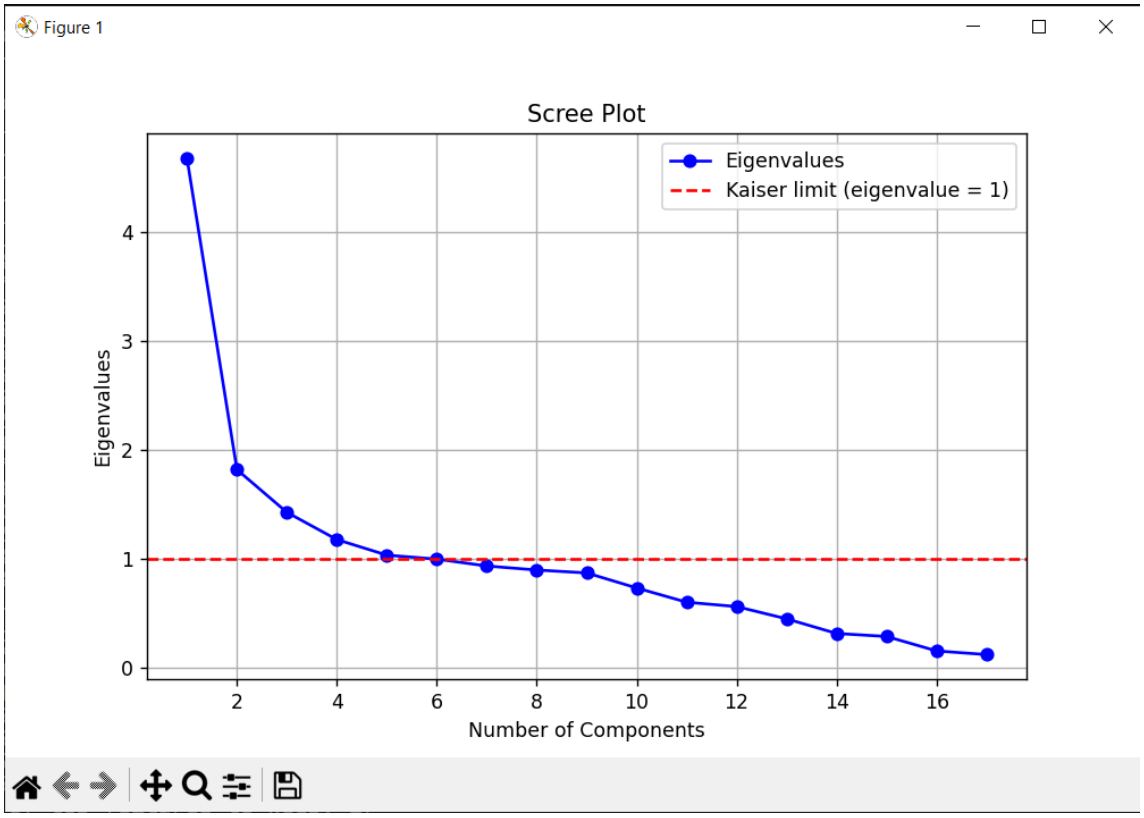


Figura 3-13 - Plot para scree test

```
Mean accuracy (cross-validation): 0.8862
Sensitivity (Recall): 0.9632
Specificity: 0.7833
F-measure (F1-score): 0.9065
```

Figura 3-14 - Resultados teste MDC-EDD com normalização, kruskal wallis e PCA (scree)

Isto pode acontecer devido à redução de dimensionalidade por parte do *PCA* algumas informações discriminativas podem estar a ser perdidas neste processo devido ao seu funcionamento e limitações. Por exemplo se *features* discriminativas não estiverem alinhadas com as direções de maior variância, o *PCA* pode reduzir a capacidade do classificador de distinguir as classes.

3.3.2 Mahalanobis Distance Discriminant

O primeiro teste será a classificação de dados sem qualquer tipo de *feature selection* ou *reduction*, para verificar como os modelos se comportam sem o devido tratamento de dados prévio.

Este teste não foi possível fazer, após uma pesquisa foi possível concluir que quando os dados não são normalizados *features* com escalas muito diferentes podem levar a instabilidade numérica e assim impossibilitando o cálculo da matriz de covariação necessária para a distância mahalanobis.

Tendo em conta o anterior será feito o teste seguinte que é o uso do classificador com os dados normalizados,

Isto é possível verificar na *Figura 3-15*.

```
Mean accuracy (cross-validation): 0.9976
Sensitivity (Recall): 0.9968
Specificity: 0.9988
F-measure (F1-score): 0.9979
```

Figura 3-15 – Resultados teste MDC-MDD com normalização z-score

É possível verificar resultados extremamente altos para todas as métricas oque mostra que o classificador é extremamente discriminante das classes presentes no *dataset*.

Passando então para a aplicação de *kruskal wallis* ao modelo com os dados normalizados, os resultados desta combinação podem ser vistos na *Figura 3-16*.

```
Mean accuracy (cross-validation): 0.9967
Sensitivity (Recall): 0.9976
Specificity: 0.9954
F-measure (F1-score): 0.9971
```

Figura 3-16 - Resultados teste MDC-MDD com normalização z-score e kruskal wallis

Os resultados apresentados por este modelo continuam a ser extremamente bons apesar de ter havido uma ligeira diminuição na performance do mesmo, isto pode significar que apesar de serem removidas *features* que não são tão discriminativas das classes da variável *target*, está se a perder alguma informação e por isso o modelo apresenta uma ligeira maior dificuldade a classificar novos dados.

Em seguida será aplicado um teste ao classificador com normalização e *feature reduction* através da *AUC* das *features*, *Figura 3-17*.

```
Mean accuracy (cross-validation): 0.9972
Sensitivity (Recall): 0.9999
Specificity: 0.9936
F-measure (F1-score): 0.9976
```

Figura 3-17 - Resultados teste MDC-MDD com normalização z-score e ROC curve (feature reduction)

Passando então para a última combinação em que adicionamos ao último modelo também o *PCA* com *kaiser* e *scree test*, e os resultados deste novo teste estão nas Figura 3-18 e Figura 3-20, já o número de componentes principais escolhidos estão na Figura 3-19 foi então escolhido 8.

```
Mean accuracy (cross-validation): 0.9604
Sensitivity (Recall): 0.9965
Specificity: 0.9123
F-measure (F1-score): 0.9665
```

Figura 3-18 - Resultados teste MDC-MDD com normalização z-score, kruskal wallis e PCA (kaiser)

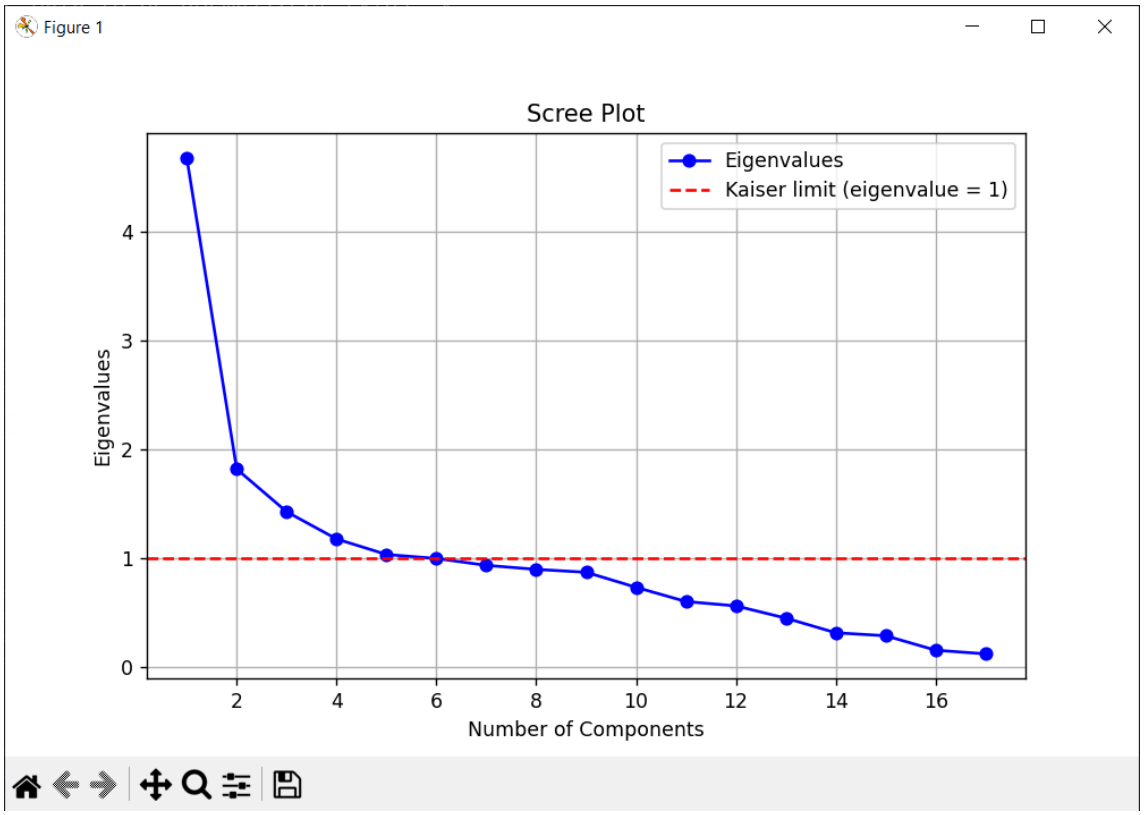


Figura 3-19 - Plot para scree test

```
Mean accuracy (cross-validation): 0.9618
Sensitivity (Recall): 0.9984
Specificity: 0.9131
F-measure (F1-score): 0.9677
```

Figura 3-20 - Resultados teste MDC-MDD com normalização z-score, kruskal wallis e PCA (scree)

Os resultados deste teste foram apesar de bastante satisfatórios piores ainda que o teste com apenas normalização e *kruskal wallis*, isto porque a redução de dimensionalidade por parte do *PCA* pode levar à perda de algumas informações discriminativas. Por exemplo se *features* discriminativas não estiverem alinhadas com as direções de maior variância, o *PCA* pode reduzir a capacidade do classificador de distinguir as classes.

3.4. Fisher LDA

O primeiro teste que seria realizado para este classificador será sem qualquer tipo de tratamento de dados, no entanto na biblioteca que está a ser usada isto já é feito inerentemente para o cálculo das matrizes de dispersão. Os resultados estão na *Figura 3-21* abaixo.

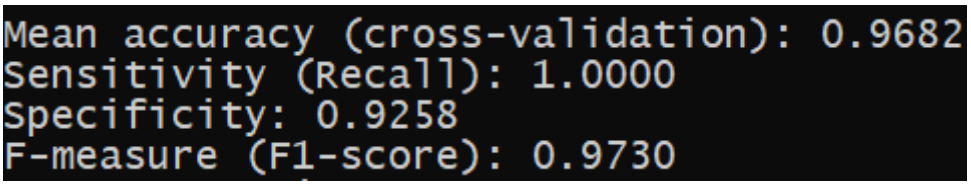


Figura 3-21 - Resultados teste Fisher LDA com normalização

Com a normalização os resultados apresentados pelo Fisher LDA já são excelentes. Assim como os 2 modelos anteriores, oque significa que as *features* escolhidas e a normalização permitem que o modelo discrimine bem os dados.

O próximo teste será a aplicação do teste *kruskal wallis* ao modelo anterior com a normalização dos dados, e os resultados podem ser verificados na *Figura 3-22*.

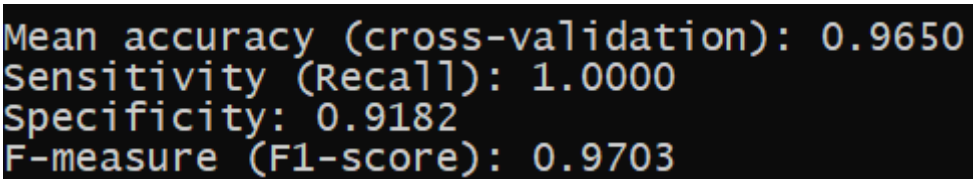


Figura 3-22 - Resultados teste Fisher LDA com normalização e kruskal wallis

Os resultados apresentados por este modelo continuam a ser extremamente bons apesar de ter havido uma ligeira diminuição na performance do mesmo, isto pode significar que apesar de serem removidas *features* que não são tão discriminativas das classes da variável *target*, está se a perder alguma informação e por isso o modelo apresenta uma ligeira maior dificuldade a classificar novos dados.

Em seguida será aplicada a *feature reduction* através da *AUC* das *features*, ao primeiro modelo com a normalização dos dados, e os resultados podem ser visto na Figura 3-23.

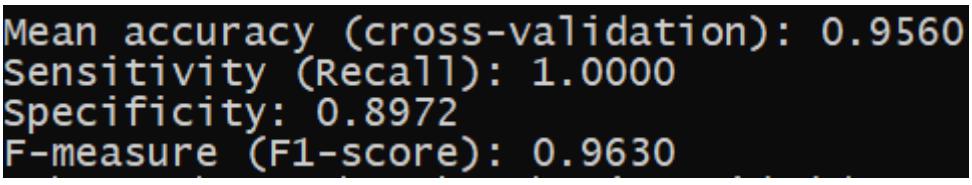


Figura 3-23 - Resultados teste Fisher LDA com normalizaçãoe ROC curves (feature reduction)

Neste caso os resultados obtidos pela redução de *features* através do *AUC* das mesmas acabou por gerar resultados piores que o *kruskal wallis*.

O próximo teste seria a implementação do PCA ao modelo anterior, no entanto como explicado no capítulo *2.6*, a aplicação do PCA seguida de Fisher LDA não funciona, pois, a projeção inicial feita pelo *PCA* pode destruir as diferenças que existem entre as classes, reduzindo a eficácia do *Fisher LDA* na discriminação dos dados. Isto pode mesmo ser visto na *Figura 3-24*, em que o *PCA* foi implementado antes da classificação *fisher*

```
Mean accuracy (cross-validation): 0.8684
Sensitivity (Recall): 0.9399
Specificity: 0.7730
F-measure (F1-score): 0.8911
```

Figura 3-24 - Resultados de Fisher LDA com PCA

3.5. Bayes classifier

Passando então ao próximo modelo este será testado apenas com a normalização dos dados uma vez que a feature reduction e a dimensionality reduction irão apenas piorar os seus resultados.

Então o teste que será feito ao modelo será apenas com a normalização dos dados, e foram então obtidos os seguintes resultados presentes na *Figura 3-25*.

```
Mean accuracy (cross-validation): 0.9983
Sensitivity (Recall): 0.9997
Specificity: 0.9964
F-measure (F1-score): 0.9985
```

Figura 3-25 – Resultados do teste Bayes classifier com normalização z-score

3.6. KNN

Antes de serem iniciados os teste ao classificador k-Nearest Neighbors foi feita uma procura pelo melhor k isto com base no erro, para isto foi definido um range e para cada k foi feita uma *crossvalidation*, para obter a média de erros e assim resultados mais coerentes e foram então obtidos os seguintes resultados, *Figura 3-26*.

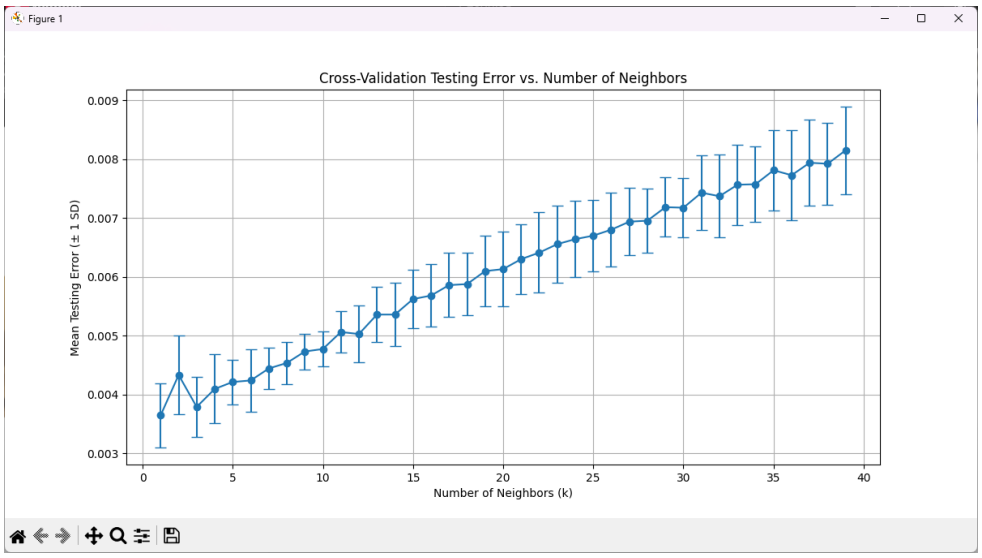


Figura 3-26 - Seleção de melhor K

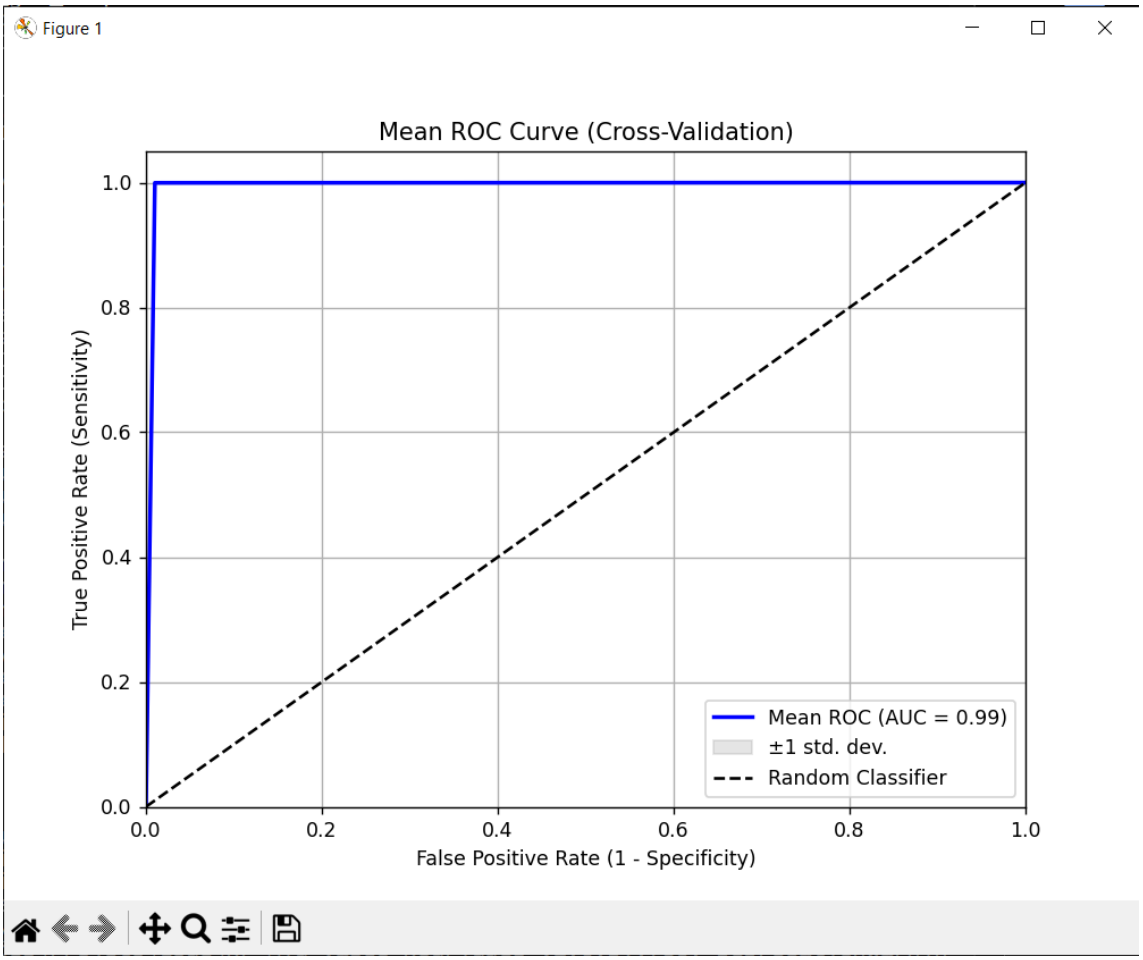
É possível então concluir que o k que oferece a melhor performance é o 3 e será então o utilizado, desconsiderando obviamente o 1 pois levaria a overfitting.

O teste aplicado a este modelo foi apenas com a normalização dos dados, e os resultados estão na *Figura 3-27* em baixo.

```
Mean accuracy (cross-validation): 0.9971
Sensitivity (Recall): 0.9986
Specificity: 0.9951
F-measure (F1-score): 0.9975
```

Figura 3-27 - Resultados teste KNN com normalização z-score

Foram novamente obtidos resultados extremamente bons, mostrando que o k foi realmente bem definido, fazendo com que o modelo se comporte bem em dados nunca antes vistos.



3.7. SVM

Do mesmo modo que foi obtido o melhor k para o classificador anterior, também para a SVM será feita uma *grid search* para os parâmetros C e *gamma* no caso de *kernel* não linear e apenas de apenas C para o caso de *kernel* linear, e então foram obtidos os seguintes resultados, *Figura 3-28*, *Figura 3-29*, *Figura 3-30* e *Figura 3-31*.

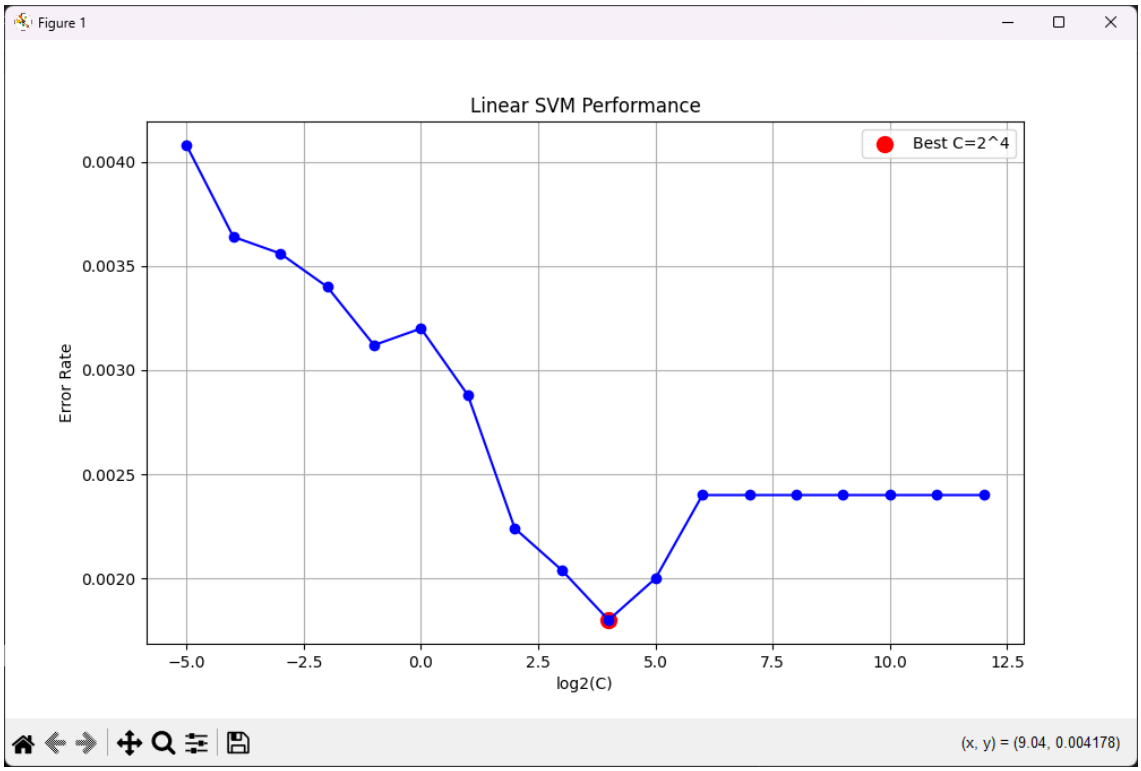


Figura 3-28 - Grid search para melhor C SVM Linear plot

```
Grid Search for linear SVM...
Grid Search Progress: 100%|                                                  | 10/10 [00:02<00:00,  4.55it/s]

--- Best Parameters (Linear SVM) ---
Best C: 16.000000
Best error: 0.001800
```

Figura 3-29 - Grid search para melhor C SVM Linear

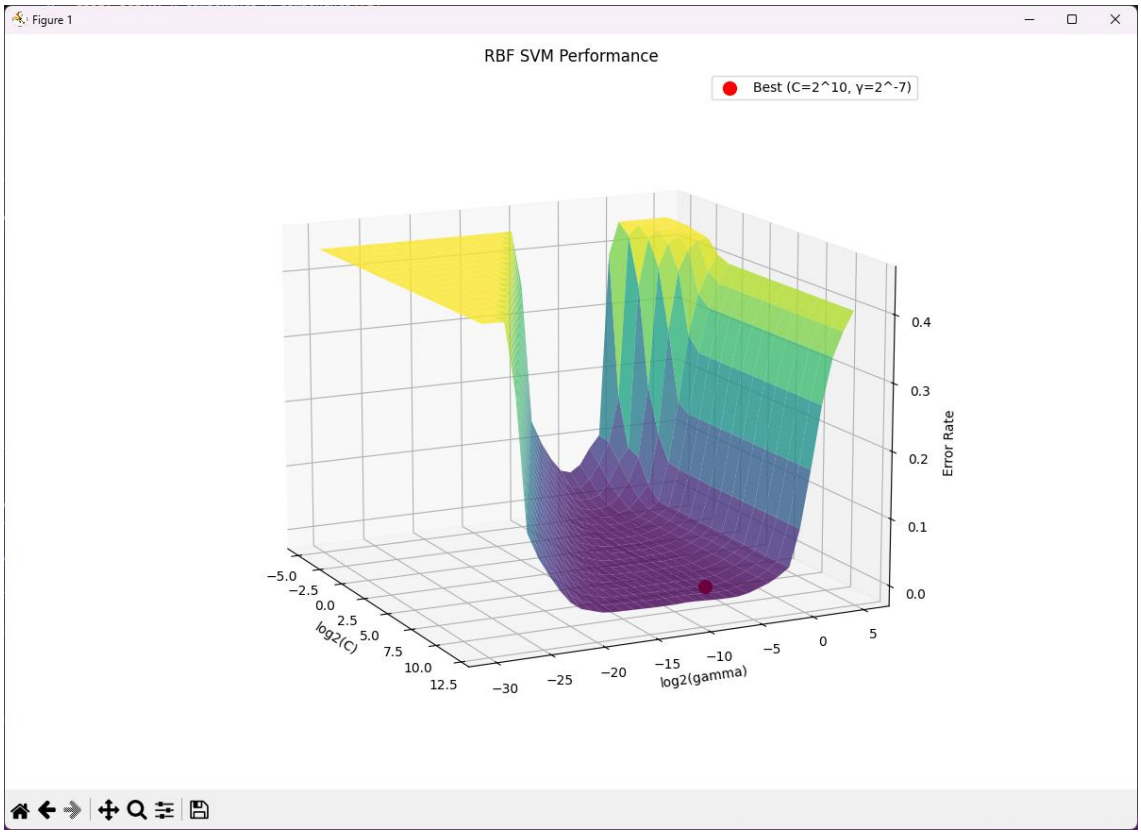


Figura 3-30 - Grid search para melhor C e gama SVM RBF plot

```
Grid Search for RBF SVM...
Grid Search Progress: 100%|                                                  | 10/10 [09:53<00:00, 59.37s/it]

--- Best Parameters (RBF SVM) ---
Best C: 1024.000000
Best gamma: 0.007812
Best error: 0.001600
```

Figura 3-31 - Grid search para melhor C e gamma SVM RBF

Como pode ser visto, para um *kernel* linear foi então obtido o valor para C de 16 e para um não linear foi obtido o valor de C de 1024 e de *gamma* de 0.001600.

Para testar estes modelos foi aplicada apenas aplicada a normalização dos dados, obtendo então os seguintes resultados nas *Figura 3-32* e *Figura 3-33*.

```
Mean accuracy (cross-validation): 0.9999
Sensitivity (Recall): 1.0000
Specificity: 0.9997
F-measure (F1-score): 0.9999
```

Figura 3-32 - Resultados teste SVM - kernel linear com normalização z-score

```
Mean accuracy (cross-validation): 0.9998
Sensitivity (Recall): 0.9998
Specificity: 0.9998
F-measure (F1-score): 0.9998
```

Figura 3-33 - Resultados teste SVM - kernel RBF com normalização z-score

E como pode ser visto foram os melhores resultados obtidos até agora por qualquer modelo, provando assim que a SVM, mais especificamente com kernel linear, foi o classificador que melhor conseguiu discriminar a variável target.

3.8. Resultados Gerais

Como é possível verificar na tabela abaixo o modelo que apresentou melhores resultados foi o SVM com um *kernel* linear, novamente os testes com feature reduction e dimensionality reduction não foram feitos, pois apenas iriam prejudicar a performance dos classificadores como foi possível ver até então

MODELO	Normalização	Normalização/Feature reduction	Normalização/Feature reduction/Dimensionality Reduction
MDC – EDD	0.9017	0.9113	Kaiser-0.8969 Scree-0.9065
MDC – MDD	0.9979	0.9971	Kaiser-0.9665 Scree-0.9677
Fisher LDA	0.9730	0.9703	Kaiser-0.8911 Scree-N/A
Bayes classifier	0.9985	N/A	N/A
k-NN	0.9975	N/A	N/A
SVM Kernel Linear	0.9999	N/A	N/A
SVM Kernel RBF	0.9998	N/A	N/A

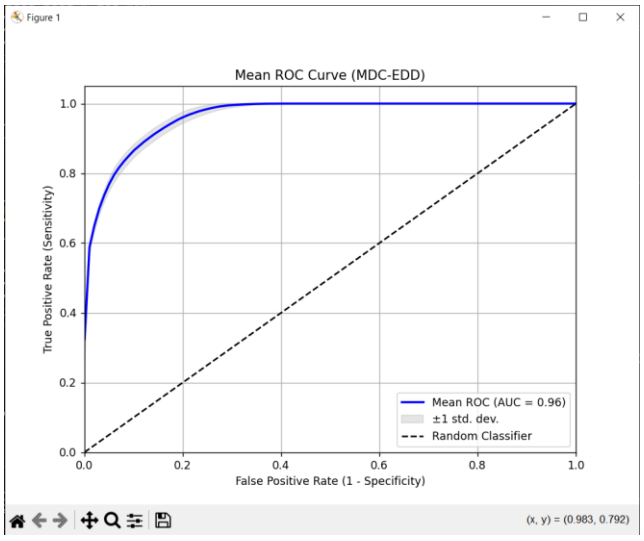


Figura 3-34 - Roc curve MDC-EDD

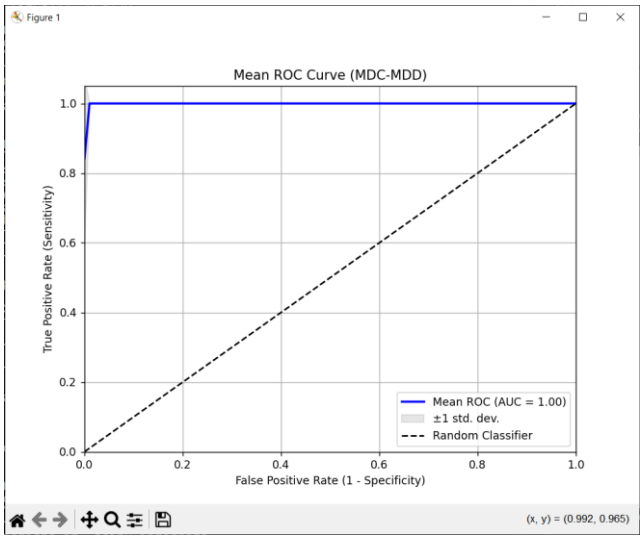


Figura 3-35 - Roc curve MDC-MDD

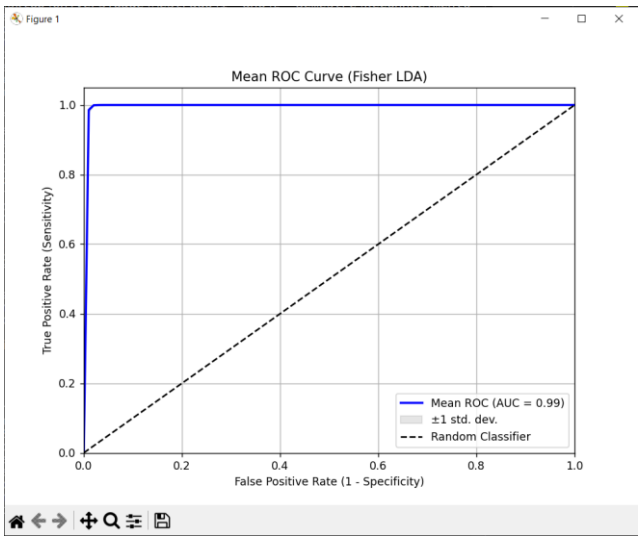


Figura 3-36 - Roc curve Fisher LDA

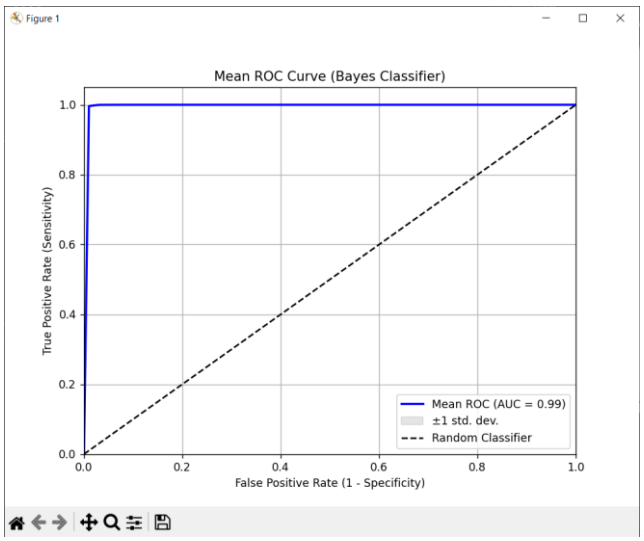


Figura 3-37 - Roc curve Bayes Classifier

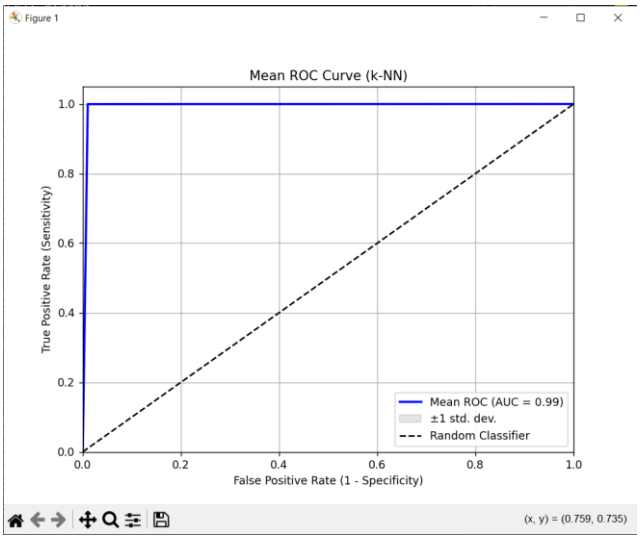


Figura 3-38 - Roc curve k-NN

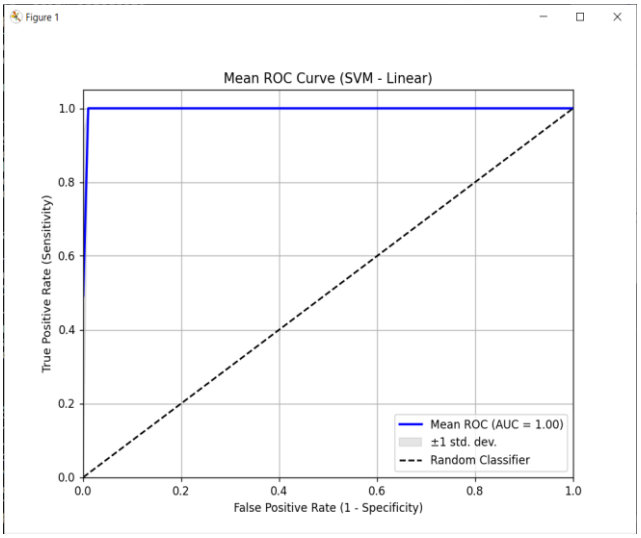


Figura 3-39 - Roc curve SVM-linear

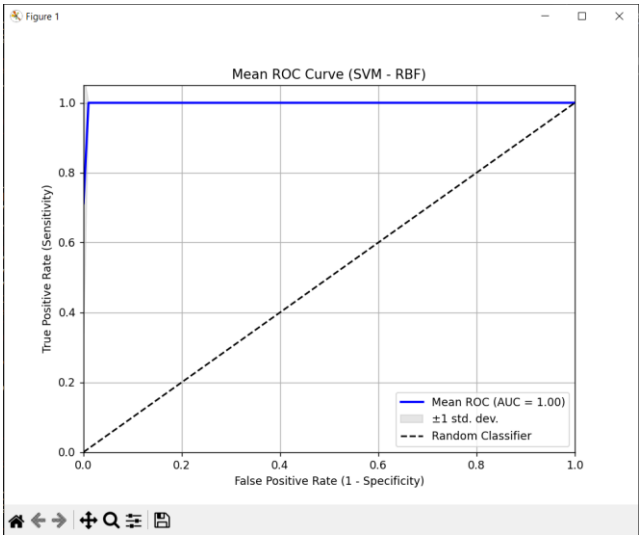


Figura 3-40 - Roc curve SVM-RBF

4. Validação de melhor modelo

Para a validação do melhor modelo foi feito da seguinte forma. Antes de fornecer o dataset ao classificador para este treinar, ele foi dividido em dois um dataset de treino e teste, onde o modelo iria estar a treinar e a ser testado através de cross-validation, e num dataset de validação que após terminar este processo seria utilizado para a validação do modelo em dados novos, simulando assim um cenário real e foram então obtidos os seguintes resultados

```
Calculating predictions...
Using best model - svm with linear kernel...
Cross-validating SVC: 100%|                                                    | 5/5 [00:19<00:00,  3.82s/fold, Acc=1.000, Sens=1.000, Spec=1.000, F1=1.000]

Metrics on trainig data

Mean accuracy (cross-validation): 0.9999
Sensitivity (Recall): 1.0000
Specificity: 0.9997
F-measure (F1-score): 0.9999
Doing validation...
Converting X to float.
Data normalized

Metrics on validating data

Mean accuracy (cross-validation): 0.9765
Sensitivity (Recall): 0.9591
Specificity: 0.9999
F-measure (F1-score): 0.9791
```

Figura 4-1 - Performance of best classifier on validation data

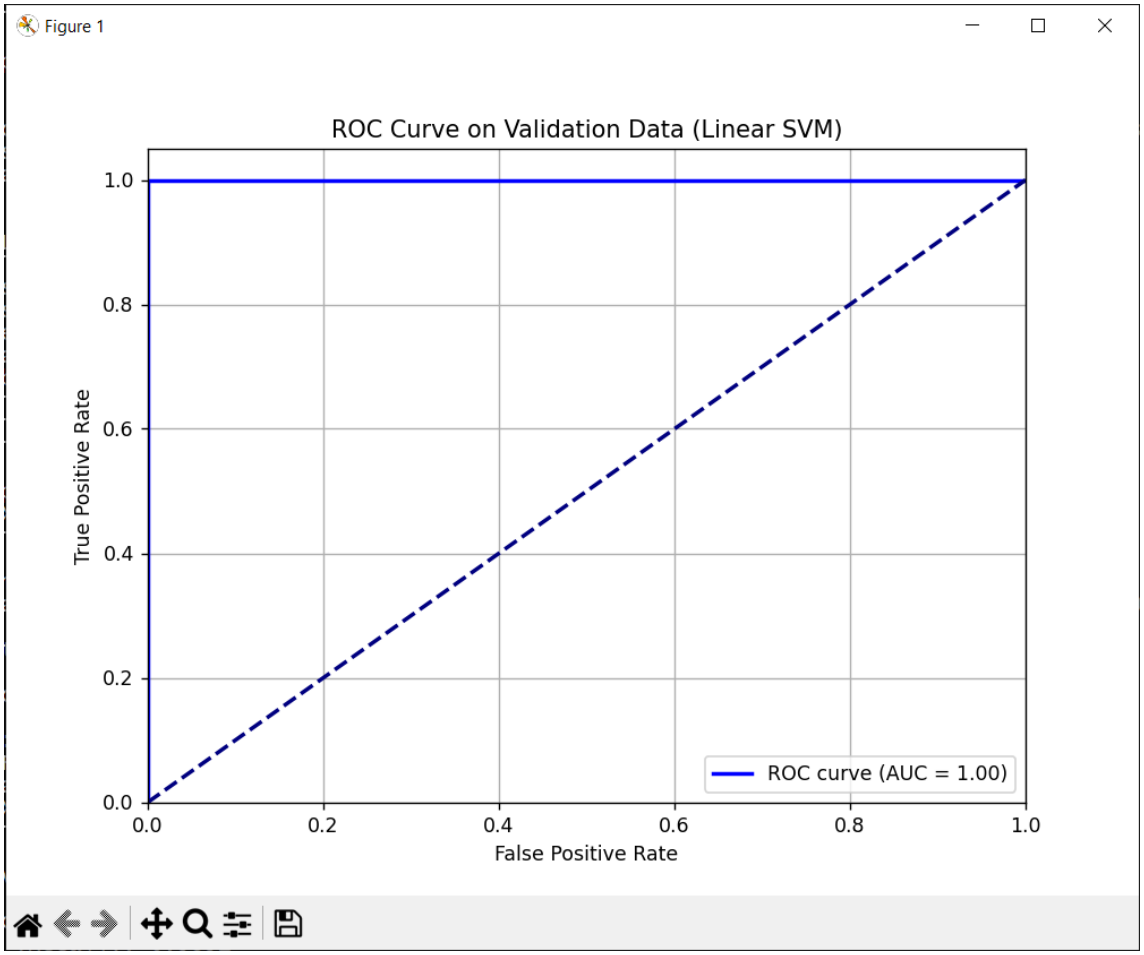


Figura 4-2 - ROC curve of best controller

5. Discussão e Conclusão

Tendo em conta todos os resultados obtidos na secção de resultados é possível concluir que o modelo que apresenta melhores resultados médios em resultados nunca antes visto é a implementação do *Suport Vector Machine* com um *Kernel linear*, obtendo um F1-score médio de 99.99% e em dados de validação obtendo um F1 score de 0.9791 com uma AUC de aproximadamente 1.

Isto revela que o melhor classificador é capaz de ser aplicado em cenários reais e continuar a ter uma performance excelente como mostrou ser capaz nos dados de teste.

É possível ainda concluir que a implementação do teste *Kruskal wallis, ROC curves* e do *PCA* apesar de serem ferramentas poderosas nem sempre são úteis principalmente quando as *features* já tiveram uma seleção minuciosa, como no nosso caso após a remoção das *features* irrelevantes e redundantes, o conjunto de dados já estava bem estruturado e com *features* informativas. Por isso, a aplicação do *Kruskal-Wallis*, ROC curves e do *PCA* resultou alguma perda de informação.

Tendo em conta a informação presente no parágrafo anterior, a realização de testes com esses métodos deixou de ser feita nos consequentes classificadores, *Bayes Classiefier*, *k-NN* e *SVM*

Foi também possível concluir que a implementação do *PCA* antes da aplicação do classificador *Fisher LDA* não faz qualquer sentido pois o *PCA* projeta os dados num novo eixo, focando-se nas direções de maior variância. No entanto, essas direções nem sempre estão alinhadas com as que maximizam a separação entre as classes, que é o objetivo do *Fisher LDA*. Como resultado, o *PCA* pode fazer com que os dados percam informação importante para a discriminação das classes, diminuindo a eficácia do *Fisher LDA* na classificação.

Foi possível também verificar a importância da normalização dos dados e *feature selection* (remoção de *features* categóricas e binárias e *features* redundantes). Pois aplicando estes dois simples métodos foi possível verificar uma melhoria enorme na classificação de novos dados por parte dos classificadores.

6. Referências

[1] Wacker, A G, and D A Landgrebe. 'Minimum Distance Classification in Remote Sensing', n.d.
[2] 'Fisher Linear Discriminant - an Overview | ScienceDirect Topics'. Accessed 21 March 2025. https://www.sciencedirect.com/topics/engineering/fisher-linear-discriminant.
[3] Alexander. 'Mahalanobis Distance: Simple Definition, Examples'. Statistics How To, 8 May 2024. https://www.statisticshowto.com/mahalanobis-distance/.
[4] GeeksforGeeks. 'Euclidean Distance | Formula, Derivation & Solved Examples', 18:50:03+00:00. https://www.geeksforgeeks.org/euclidean-distance/.
[5] Wei, Dagang. 'Demystifying Machine Learning: Normalization'. *Medium* (blog), 12 February 2025. https://medium.com/@weidagang/demystifying-machine-learning-normalization-0cdb8b281234.
[6] 'T-Test, Chi-Square, ANOVA, Regression, Correlation...' Accessed 21 March 2025. https://datatab.net/tutorial/kruskal-wallis-test.