**Projeto da Disciplina de Data Mining**

Prof. Manoelaprof.manoela@ica.ele.puc-rio.br

**Componentes do Projeto:**

André Luis Mendes Teixeira

Gabriela de Camargo Santa Rosa

Mariana Fernandes Coy

**Histórico de Versões**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** | **Aprovado por** |
| 30/12/2021 | 1.0 | Documento de  elaboração do projeto de DM sobre modelo de algoritmos de classificação | André Teixeira | Gabriela C. Santa Rosa  Mariana Fernandes Coy |
| 02/01/2022 | 2.0 | Revisão da versão 1.0 com os resultados de pré-processamento e SVM | Gabriela C. Santa Rosa | André Teixeira  Mariana Fernandes Coy |
| 06/01/2022 | 3.0 | Revisão da versão 2.0, após correções de pré-processamentos | Gabriela C. Santa Rosa | André Teixeira Mariana Fernandes Coy |
| 07/01/2022 | 4.0 | Revisão Final | André Teixeira  Mariana Fernandes Coy | Gabriela C. Santa Rosa |

Sumário

[Proposta de trabalho 3](#_Toc92563217)

[Análise Exploratória 3](#_Toc92563218)

[Pré-processamento 4](#_Toc92563219)

[Valores Nulos (Missing Values) 4](#_Toc92563220)

[Transformação dos atributos categóricos 7](#_Toc92563221)

[Normalização 9](#_Toc92563222)

[Balanceamento 10](#_Toc92563223)

[Treinamento do modelo e inferências usando os algoritmos de classificação 13](#_Toc92563224)

[SVM (Support Vector Machine) 13](#_Toc92563225)

[Árvore de Decisão 16](#_Toc92563226)

[Random Forest 19](#_Toc92563227)

[KNN 22](#_Toc92563228)

[Conclusão 24](#_Toc92563229)

[Anexos 26](#_Toc92563230)

# Proposta de trabalho

Este trabalho tem como proposta o desenvolvimento de um modelo de predição, dada uma determinada base, para a aplicação dos conceitos aprendidos na disciplina de Data Mining.

Para isto, foi utilizado a base de dados sugerida em aula para o desenvolvimento deste trabalho, onde foi proposto um problema de classificação de uma base de dados contendo 27 atributos numéricos e categóricos que descrevem o estado de saúde de cavalos, e três classes de saída que indicam o que aconteceu com o animal: morreu, viveu ou em estado de eutanásia. A ideia é prever se um cavalo pode sobreviver ou morrer baseado nas condições médicas passadas.

Para este trabalho temos já previamente separadas as bases de treino e teste, passadas como “horse.csv” (base de treino) e “horseTest.csv” (base de teste). A base de treino contém 299 registros, e a de teste, 89.

Por se tratar de um problema de classificação, foi feito o desenvolvimento dos modelos através dos seguintes algoritmos:

* Support Vector Machine (SVM)
* Árvores de Decisão
* Random Forest
* K nearest neighbors (KNN)

Foi utilizada a linguagem de programação Python usando a estrutura do Google Colaboratory para realização de treinos e inferências dos modelos.

# Análise Exploratória

A análise exploratória dos dados nos ajuda a prever que dados são considerados relevantes ou irrelevantes para o modelo, para que possamos desconsiderá-los com o intuito de melhorar o desempenho de predição do modelo.

Algumas observações foram feitas através do documento de dicionário de dados e ao analisar os dados oriundos da base de treino:

- Removidos os campos hospital\_number (irrelevante para o resultado do estado do animal), respiratory\_rate (indicado no dicionário de dados como de uso duvidoso devido à grandes flutuações) e cp\_data (indicado como não significantes, conforme o dicionário de dados).

- Exclusão das colunas lesion\_2 e lesion\_3 uma vez que a maioria dos valores são "zero" e, de acordo com o gráfico de dispersão em relação à classe, se torna irrelevante para o modelo.

Calendário

Descrição gerada automaticamente

Figura 1 – Gráfico de dispersão dos atributos comparados com atributos lesion\_2 e lesion\_3 (utilizado seaborn.pairplot)

# Pré-processamento

Para a etapa de pré-processamento foram considerados o tratamento de Missing Values, transformação dos atributos categóricos, normalização dos atributos numéricos e balanceamento das classes.

Inicialmente foi feita a separação das bases de treino e teste para entradas e saídas.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 2 – Programação para separação das bases de treino e teste em entradas e saídas

# Valores Nulos (Missing Values)

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figura 3 – Matriz para verificação de valores nulos nos atributos utilizando a biblioteca missingno

Após verificação da matriz de valores nulos, foram feitas as exclusões dos atributos nasogastric\_reflux\_ph, abdomo\_appearance e abdomo\_protein por excesso de nulos.

Assim, os valores nulos se configuraram desta forma:

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figura 4 – Matriz para verificação de valores nulos após a exclusão de atributos

Foi verificado a quantidade de valores nulos por atributo, conforme abaixo:

Uma imagem contendo Texto

Descrição gerada automaticamente

Figura 5 – Verificação da quantidade de valores nulos por atributo

Após esta verificação, foi feito o tratamento para os missing values, considerando para os atributos categóricos a utilização da moda para preenchimento dos valores nulos, e para os atributos numéricos foi atribuído o valor da média dos registros para cada respectivo atributo.

Para fazer o tratamento de missing values, foi considerado a separação das bases em dataframes auxiliares, sendo um considerando os atributos categóricos e outro considerando os atributos numéricos.

O primeiro passo foi verificar quais seriam os atributos numéricos e categóricos da base, utilizando a classe make\_column\_selector e atribuindo às variáveis auxiliares numerical\_columns e categorical\_columns.

Texto

Descrição gerada automaticamente

Figura 6 – Programação para verificar os atributos numéricos e categóricos das bases

Para o tratamento dos missing values foi utilizada a classe SimpleImputer. Para fazer a transformação das bases, foi utilizado a classe ColumnTransformer, uma vez que a base tem atributos mistos (categóricos e numéricos).

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 7 – Programação para o tratamento dos missing values nas bases de treino e teste

Após o tratamento, como resultado as bases foram transformadas em arrays. Uma vez que outros tratamentos são necessários, as bases foram transformadas novamente em dataframes.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Figura 8 – Programação para a transformação das bases de treino e teste novamente em dataframes

Foi feita novamente a verificação na base de treino após o tratamento de missing values.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figura 9 – Matriz para verificação de valores nulos após o tratamento de missing values na base de treino

As mesmas considerações acima foram replicadas na base de teste.

# Transformação dos atributos categóricos

Após, foi feito o tratamento para conversão dos atributos categóricos em numéricos. Foi verificado que, após o tratamento de missing values descrito acima, todos os atributos das bases de treino e teste estavam sendo considerados como categóricos.

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 10 – Tipo dos atributos na base de treino

Portanto foi necessário fazer a conversão dos atributos numéricos das bases para o tipo numérico.

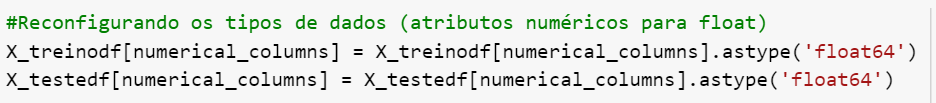


Figura 11 – programação para reconfiguração dos atributos numéricos nas bases

Uma imagem contendo Tabela

Descrição gerada automaticamente Tabela

Descrição gerada automaticamente com confiança média

Figura 12 – Verificação dos tipos de atributos das bases, após reconfiguração

Após esta reconfiguração, foi utilizada a classe OneHotEncoder da biblioteca de pré-processamento do Scikit-learn. Não foi utilizado o LabelEncoder pois neste tipo de transformação cada rótulo do atributo iria ser transformado em um número inteiro e isso poderia gerar problema, pois o modelo poderia entender como rótulos de pesos diferentes. Anteriormente já havia sido verificado quais são os atributos categóricos da base. Será necessário utilizar esta informação novamente para esta transformação.

Texto

Descrição gerada automaticamente

Figura 13 – Importação das classes OneHotEncoder e StandardScaler para a transformação dos atributos categóricos em numéricos e normalização

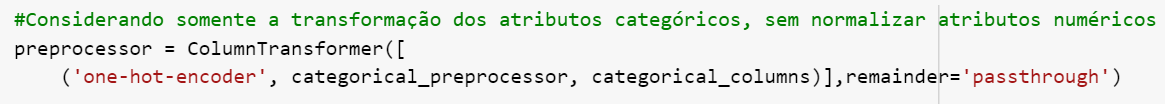


Figura 14 – Utilização da classe ColumnTransformer para a transformação das bases

Para verificar os resultados sem normalizar os atributos numéricos, foi optado por utilizar dataframes auxiliares, para não modificar as bases. Após, foi feito o fit na base de treino e a transformação dos atributos categóricos utilizando a classe OneHotEncoder nas bases auxiliares de treino e teste.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 15 – Transformação dos atributos categóricos nas bases auxiliares de treino e teste

# Normalização

Na primeira inferência do modelo SVM, sem a normalização dos dados, foi verificado um resultado de desempenho ruim do modelo, conforme imagem abaixo:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura 16 – Resultado da Matriz de Confusão do modelo utilizando SVM, sem a normalização dos dados

Para tanto, foi feito a normalização das bases de treino e teste, utilizando a classe StandardScaler do Scikit-learn, onde foi possível verificar a melhora no desempenho do modelo. Como a transformação anterior dos atributos categóricos foram feitas em bases auxiliares apenas como verificação, pois era previsível que fosse necessária a normalização dos dados, a transformação a seguir foi feita considerando as próprias bases de treino e teste, que não haviam sofridas as transformações para os atributos categóricos. Portanto neste momento foram feitas simultaneamente nas bases de treino e teste as transformações para os atributos categóricos (utilizando a classe OneHotEcoder) e a normalização para os atributos numéricos (utilizando a classe StandarScaler).

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 17 – Transformação dos atributos categóricos nas bases de treino e teste

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura 18 – Resultado da Matriz de Confusão do modelo utilizando SVM, com a normalização dos dados

# Balanceamento

Após a normalização dos dados, foi verificado que o modelo previu uma quantidade relativa de animais eutanasiados como vivos ou mortos (conforme última imagem). Verificando o balanceamento da base de treino, de fato há um desbalanceamento entre as classes, conforme imagem abaixo:

Gráfico, Gráfico de barras

Descrição gerada automaticamente

Figura 19 – Gráfico representando o Balanceamento das Classes da base de treino

Para melhorar o balanceamento entre as classes, foi feito um over-sampling na base de treino da classe minoritária (euthanized), duplicando os registros já existentes.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 20 – Programação para Balanceamento da classe minoritária

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 21 – Programação para Balanceamento da classe minoritária e resultado do balanceamento

Após a base de treino balanceada, podemos verificar um desempenho melhor do modelo:

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 22 – Resultado da Matriz de Confusão do modelo utilizando SVM, após o balanceamento da classe minoritária

# Treinamento do modelo e inferências usando os algoritmos de classificação

Após o pré-processamento das bases detalhado anteriormente, foi feita o desenvolvimento dos modelos, utilizando os diversos algoritmos, conforme a seguir.

# SVM (Support Vector Machine)

O SVM foi o primeiro modelo considerado para o desenvolvimento deste trabalho. Foi utilizado a mesma programação do exercício de crédito bancário, dado em aula, para exemplificação deste modelo.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 23 – Programação de treinamento do modelo utilizando SVM

Como a etapa de pré-processamento foi feita utilizando este modelo para verificação dos resultados, foi possível verificar as diferenças de resultados, conforme a aplicação do pré-processamento.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** | | |
| SVM | SEM normalização | 0,62921 | 0,10838 | 0,23077 | 0,00000 | 0,76259 |
| COM normalização | 0,87640 | 0,76261 | 0,85106 | 0,55556 | 0,93805 |
| Balanceamento classe minoritária | 0,93258 | 0,877635 | 0,930233 | 0,888889 | 0,944444 |

Tabela 1 – Evolução do desempenho do modelo SVM

Na tentativa de melhorar ainda mais o desempenho do modelo, foi utilizado o Grid Search.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 24 – Programação do modelo SVM utilizando o Grid Search

A primeira tentativa de utilização dos hiperparâmetros acabou resultando em um desempenho pior, conforme mostrado abaixo:

Uma imagem contendo Gráfico

Descrição gerada automaticamente Uma imagem contendo Tabela

Descrição gerada automaticamente

Figura 25 – Resultado da Matriz de Confusão do modelo utilizando SVM, após a aplicação do Grid Search, e resultado dos melhores parâmetros

Na tentativa de melhorar o desempenho, foi executado uma nova rodada do modelo, considerando o aumento do parâmetro C para 10000, onde obteve um desempenho consideravelmente melhor, conforme abaixo:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente Uma imagem contendo Padrão do plano de fundo

Descrição gerada automaticamente

Figura 26 – Resultado da Matriz de Confusão do modelo utilizando SVM, após a aplicação do Grid Search com C = 10000, e resultado dos melhores parâmetros

Foi feita mais uma rodada de verificação, agora considerando C = 10000 e gamma = 0,01, onde o modelo obteve o desempenho de 100% nas métricas utilizadas.

Interface gráfica do usuário

Descrição gerada automaticamente Uma imagem contendo Padrão do plano de fundo

Descrição gerada automaticamente

Figura 27 – Resultado da Matriz de Confusão do modelo SVM, após a aplicação do Grid Search com C = 10000, e resultado dos melhores parâmetros

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** | | |
| SVM | SEM normalização | 0,62921 | 0,10838 | 0,23077 | 0,00000 | 0,76259 |
| COM normalização | 0,87640 | 0,76261 | 0,85106 | 0,55556 | 0,93805 |
| Balanceamento classe minoritária | 0,93258 | 0,877635 | 0,930233 | 0,888889 | 0,944444 |
| Grid Search - C: 1000, gamma: 0,001 | 0,82022 | 0,666823 | 0,772727 | 0,666667 | 0,872727 |
| Grid Search - C: 10000, gamma: 0,001 | 0,98876 | 0,98002 | 1,00000 | 0,96296 | 0,99048 |
| Grid Search C: 10000, gamma: 0,01 | 1 | 1 | 1 | 1 | 1 |

Tabela 2 – Evolução do desempenho do modelo SVM, após Grid Search

# Árvore de Decisão

Para o treinamento do modelo utilizando o algoritmo de Árvore de Decisão, em vez de utilizar o OneHotEncoder para transformação dos atributos categóricos, foi utilizado o OrdinalEncoder (mesma função do LabelEncoder, para mais de um atributo) para manter o número de colunas para que posteriormente fosse possível verificar nos nós os atributos a que se referem.

Texto

Descrição gerada automaticamente

Figura 28 – Importação da classe OrdinalEncoder para transformação dos atributos categóricos no modelo de Árvore de Decisão

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 29 – Transformação e normalização dos atributos categóricos e numéricos nas bases de treino e teste

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 30 – Programações para o treinamento do modelo de Árvore de Decisão e visualização gráfica do modelo

Diagrama

Descrição gerada automaticamente

Figura 31 - Visualização gráfica do modelo, gerando como nó raiz o atributo packed\_cell\_volume

Os resultados apresentados do modelo com os atributos numéricos sem e com normalização foram os mesmos.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura 32 –Resultado da Matriz de Confusão do modelo utilizando Árvore de Decisão para base com os atributos numéricos sem normalizar e normalizados

Após o balanceamento da classe minoritária, o modelo apresentou um melhor desempenho, considerando o kappa e F1.

Gráfico, Gráfico de mapa de árvore

Descrição gerada automaticamente

Figura 33 – Resultado da Matriz de Confusão do modelo de Árvore de Decisão, após o balanceamento da classe minoritária

Após a utilização do Grid Search, o modelo apresentou melhora no desempenho, considerado satisfatório.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 34 – Programação do modelo de Árvore de Decisão utilizando o Grid Search

Interface gráfica do usuário

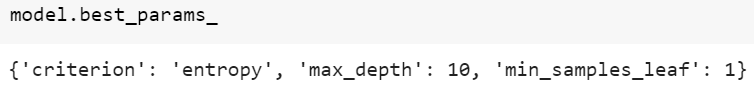
Descrição gerada automaticamente 

Figura 35 – Resultado da Matriz de Confusão do modelo de Árvore de Decisão, após a aplicação do Grid Search, e resultado dos melhores parâmetros

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figura 36 - Visualização gráfica do modelo, gerando como nó raiz o atributo lesion\_1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** | | |  |
| Árvore de Decisão | SEM normalização | 0,820225 | 0,670675 | 0,744186 | 0,692308 | 0,880734 |  |
| COM normalização | 0,820225 | 0,670675 | 0,744186 | 0,692308 | 0,880734 |  |
| Balanceamento classe minoritária | 0,820225 | 0,70358 | 0,823529 | 0,75 | 0,842105 |  |
| Grid Search - criterion: entropy, max\_depth: 10, min\_samples\_leaf: 1 | 0,977528 | 0,959674 | 0,956522 | 1 | 0,981132 |  |

Tabela 3 – Evolução do desempenho do modelo, utilizando Árvore de Decisão

# Random Forest

O próximo modelo treinado foi utilizando o Random Forest. As mesmas premissas de pré-processamento do modelo de SVM foram consideradas para este modelo.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 37 – Programações para o treinamento do modelo Random Forest

Os resultados das métricas com os atributos numéricos da base sem normalizar e normalizando apresentaram o mesmo resultado.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 38 –Resultado da Matriz de Confusão do modelo utilizando Random Forest para base com os atributos numéricos sem normalizar e normalizados

Após o balanceamento da classe minoritária, houve uma evolução nas métricas e o resultado foi conforme abaixo:

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 39 –Resultado da Matriz de Confusão do modelo Random Forest para base com os atributos numéricos normalizados e balanceamento da classe minoritária

Na tentativa de melhorar o desempenho do modelo, foi utilizado o Grid Search.

Texto

Descrição gerada automaticamente

Figura 40 – Programação do modelo Random Forest utilizando o Grid Search

Já na primeira tentativa de utilização dos hiperparâmetros, após 20 minutos de execução, as métricas do modelo apresentaram 100% de desempenho.

Interface gráfica do usuário

Descrição gerada automaticamente Uma imagem contendo Texto

Descrição gerada automaticamente

Figura 41 – Resultado da Matriz de Confusão do modelo Random Forest, após a aplicação do Grid Search, e resultado dos melhores parâmetros

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** | | |
| Random Forest | SEM normalização | 0,831461 | 0,666583 | 0,744186 | 0,555556 | 0,905983 |
| COM normalização | 0,831461 | 0,666583 | 0,744186 | 0,555556 | 0,905983 |
| Balanceamento classe minoritária | 0,88764 | 0,789797 | 0,85 | 0,846154 | 0,910714 |
| Grid Search - max\_features: 0.3, min\_samples\_leaf: 1, n\_estimators: 400 | 1 | 1 | 1 | 1 | 1 |

Tabela 4 – Evolução do desempenho do modelo, utilizando Random Forest

# KNN

O último modelo treinado foi utilizando o KNN.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 42 – Programação de treinamento do modelo utilizando KNN

As mesmas premissas de pré-processamento do modelo de SVM foram consideradas para o modelo utilizando o KNN. As métricas apresentaram 100% de desempenho já na primeira verificação (bases sem normalização e classes desbalanceadas). Embora seja muito importante que os registros estejam normalizados para a utilização deste modelo, para esta base não se mostrou considerável (talvez pelo fato da maioria dos atributos serem categóricos). Após a normalização e balanceamento, o desempenho se manteve em 100%.

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 43 – Resultado da Matriz de Confusão do modelo utilizando KNN, para bases sem normalização, base normalizada e após balanceamento da classe minoritária

Após aplicar o Grid Search neste modelo, considerando a verificação de hiperparâmetros para n\_neighbors, o modelo encontrou como melhor parâmetro n\_neighbors = 2 e as métricas apresentaram uma queda no desempenho, conforme mostrado na figura abaixo:

Texto

Descrição gerada automaticamente com confiança média

Figura 44 – Programação do modelo utilizando o Grid Search no KNN

Uma imagem contendo Aplicativo

Descrição gerada automaticamente Linha do tempo

Descrição gerada automaticamente com confiança média

Figura 45 – Resultado da Matriz de Confusão do modelo utilizando KNN, e resultado do melhor parâmetro

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** |
| KNN | SEM normalização | 1 | 1 | 1 |
| COM normalização | 1 | 1 | 1 |
| Balanceamento classe minoritária | 1 | 1 | 1 |
| Grid Search - n\_neighbors: 2 | 0,910112 | 0,847668 | 0,911135414 |

Tabela 5 – Evolução do desempenho do modelo, utilizando KNN

# Conclusão

Com este trabalho foi possível verificar que utilizando os diversos modelos de predições para as mesmas bases, podemos encontrar resultado diferentes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Modelo** | | **Acurácia** | **Kappa** | **F1** | | |
| SVM | SEM normalização | 0,62921 | 0,10838 | 0,23077 | 0,00000 | 0,76259 |
| COM normalização | 0,87640 | 0,76261 | 0,85106 | 0,55556 | 0,93805 |
| Balanceamento classe minoritária | 0,93258 | 0,877635 | 0,930233 | 0,888889 | 0,944444 |
| Grid Search - C: 1000, gamma: 0,001 | 0,82022 | 0,666823 | 0,772727 | 0,666667 | 0,872727 |
| Grid Search - C: 10000, gamma: 0,001 | 0,98876 | 0,98002 | 1,00000 | 0,96296 | 0,99048 |
| Grid Search C: 10000, gamma: 0,01 | 1 | 1 | 1 | 1 | 1 |
| Árvore de Decisão | SEM normalização | 0,820225 | 0,670675 | 0,744186 | 0,692308 | 0,880734 |
| COM normalização | 0,820225 | 0,670675 | 0,744186 | 0,692308 | 0,880734 |
| Balanceamento classe minoritária | 0,820225 | 0,70358 | 0,823529 | 0,75 | 0,842105 |
| Grid Search - criterion: entropy, max\_depth: 10, min\_samples\_leaf: 1 | 0,977528 | 0,959674 | 0,956522 | 1 | 0,981132 |
| Random Forest | SEM normalização | 0,831461 | 0,666583 | 0,744186 | 0,555556 | 0,905983 |
| COM normalização | 0,831461 | 0,666583 | 0,744186 | 0,555556 | 0,905983 |
| Balanceamento classe minoritária | 0,88764 | 0,789797 | 0,85 | 0,846154 | 0,910714 |
| Grid Search - max\_features: 0.3, min\_samples\_leaf: 1, n\_estimators: 400 | 1 | 1 | 1 | 1 | 1 |
| KNN | SEM normalização | 1 | 1 | 1 | | |
| COM normalização | 1 | 1 | 1 | | |
| Balanceamento classe minoritária | 1 | 1 | 1 | | |
| Grid Search - n\_neighbors: 2 | 0,910112 | 0,847668 | 0,911135414 | | |

Tabela 6 – Evolução do desempenho dos modelos

Para todos os modelos foi possível chegar em resultados satisfatório de desempenho, considerando as métricas de acurácia, kappa e F1, especialmente após a utilização de hiperparâmetros. No entanto para o caso deste trabalho, podemos considerar como melhor resultado encontrado utilizando o modelo KNN, uma vez que, apenas com pré-processamento dos atributos (categóricos e normalização dos numéricos) já foi possível atingir um desempenho de 100%, além de se tratar de um modelo fácil de implementação.

# Anexos

Em anexo a este relatório, segue os arquivos utilizados para a realização deste trabalho:

**Base de treino:** horse.csv

**Base de teste:** horseTest.csv

**Modelo SVM:** ProjetoHorse\_SVM.ipynb

**Modelo Árvore de Decisão:** ProjetoHorse\_AD.ipynb

**Modelo Random Forest:** ProjetoHorse\_RF.ipynb

**Modelo KNN:** ProjetoHorse\_KNN.ipynb

**Visualização gráfica da Árvore de Decisão:** tree\_high\_dpi-300.png

**Visualização gráfica AD – após Grid Search:** tree\_high\_dpi-300-Grid-Search.png