Inteligência Artificial

Trabalho 2

Relatório Técnico

FCUP 2020/2021

Filipe Colla David - 201810097

Márcia Carneiro – 201202560

Resumo:

Machine Learning é um método de análise de dados que automatiza o processo de construção de modelos analíticos.
Este relatório tem como objetivo explorar diferentes abordagens para criação e aperfeiçoamento de modelos de classificação usando um conjunto de dados recolhido pela Columbia University durante estudo sobre *Speed Dating.* Para a criação destes modelos foram usados dois algoritmos, *Machine Learning: Árvores de Decisão (*CART - Classification and Regression Trees) e Naive Bayes. Com base nas experiências realizadas ao longo deste trabalho, esperamos oferecer ao leitor algumas conclusões sobre diferentes abordagens na criação e aperfeiçoamento de modelos para *Machine Learning* e as vantagens e desvantagens das mesmas.

Palavras-Chave: Machine Learning, *Árvores de Decisão,* Naive Bayes, Modelos Analíticos.

Introdução:

O presente relatório do trabalho prático é elaborado no âmbito da unidade curricular de Inteligência Artificial.

Este trabalho tem como objetivo criar e avaliar vários modelos de classificação para um conjunto de dados recolhidos entre 2002 e 2004, pela Columbia University sobre participantes em eventos experimentais de *speed dating.*

A realização deste trabalho foi dividida em cinco partes:

- Breve analise superficial dos dados
- Uso da linguagem *Python* juntamente com as bibliotecas pandas, matplotlib para fazer uma análise mais profunda (pré-processamento).
- Teste de diferentes configurações para a criação dos modelos, usando os algoritmos CART (*Classification And Regression Trees*) e NaiveBayes (*Gaussian*) com o auxílio das bibliotecas sklearn.
- Comparação dos resultados obtidos pelos diferentes algoritmos e respectivas configurações.
- Conclusões.

Algoritmos:

Árvores de decisão

Os algoritmos de *machine learning* baseados em árvores de decisão são considerados um dos melhores e mais utilizados métodos de aprendizagem supervisionada. Os métodos baseados em árvores dão-nos muitas vezes modelos preditivos de alta precisão, estabilidade e de fácil interpretação. Ao contrário dos modelos lineares, mapeiam muito bem relações não lineares podendo ser adaptados para resolver vários tipos de problema (classificação ou regressão). Para este trabalho, a variável objetivo sendo categórica (match) falamos em árvores de classificação.

As árvores de decisão dividem recursivamente os dados em relação à pureza de sua variável de objetivo. O algoritmo é projetado para encontrar a forma mais preditiva, a fim de dividir um conjunto de dados em subconjuntos. Esses novos conjuntos de dados serão mais puros do que o conjunto de dados original. Num sentido geral, a "pureza" pode ser considerada como o quão homogéneo um grupo é. As duas medidas de impureza mais utilizadas para as decisões da árvore de decisão são o índice de Gini e a entropia. Dependendo da impureza utilizada, os resultados da classificação da árvore podem variar. Isso pode causar um impacto pequeno ou às vezes grande no modelo.

O algoritmo de árvore de classificação e regressão (CART) utiliza o índice de Gini para originar divisões binárias. O Índice de Gini ou impureza de Gini, calcula a probabilidade de uma variável ser classificada incorretamente quando selecionada aleatoriamente. O índice de Gini varia entre 0 e 1, onde 0 expressa a pureza da classificação, ou seja, todos os elementos pertencem a uma mesma classe ou apenas a uma classe. O valor 1 indica a distribuição aleatória de elementos em várias classes e um valor de 0,5 do Índice de Gini mostra uma distribuição igual dos elementos em algumas classes.

Para a construção de uma árvore de decisão neste trabalho foi utilizado o algoritmo CART.

Algoritmo CART

Árvore de classificação e regressão (CART) é um método de classificação que usa dados históricos para construir a árvore de decisão, permitindo classificar novos possíveis dados. Para usar o CART, é necessário saber o número de classes a priori. Para a realização deste trabalho, o conjunto de dados fornecidos continha 10 atributos: 'age', 'age_o','goal','date','go_out','int_corr','length','met', 'like' e 'prob'. Sendo o objetivo dividir a amostra de aprendizagem em partes cada vez menores, o CART pesquisa todas as variáveis possíveis e todos os valores possíveis, a fim de encontrar a melhor forma de dividir os dados em duas partes com homogeneidade máxima. Para essa divisão CART utiliza então o índice de Gini em que no caso de um atributo de valor discreto como por exemplo 'like' ou 'prob', o subconjunto que fornece o índice Gini mínimo é o selecionado como um atributo de divisão. Semelhante acontece no caso de atributos de valor contínuo como 'age', a estratégia é subdividir em intervalos contínuos e selecionar o conjunto de valores adjacentes como um possível ponto de divisão e ponto com menor índice de Gini.

O processo é então repetido para cada uma das frações de dados resultantes. Dito de uma forma resumida, a metodologia do CART consiste em 3 partes: Construir a árvore máxima, escolha do tamanho mais adequado para a árvore bem como outros parâmetros e, classificar novos dados utilizando a árvore construída.

Naive Bayes

No campo da estatística, *Naive Bayes classifiers*, são uma família de classificadores probabilísticos baseados na aplicação do teorema de *Bayes*. O teorema de *Bayes* descreve a probabilidade de um acontecimento baseando-se no conhecimento prévio de condições que possam afetar a probabilidade desse acontecimento.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

Figura 1 - Teorema de Bayes

A aplicação do teorema de *Bayes* no algortimo *Naive Bayes classifiers* baseia-se no uso de inferências fortes *(naive*) entre as diferentes *features*. Existem vários tipos de classificadores *Naive Bayes*, mas neste trabalho foi usado o Gaussian naive Bayes.

O *Gaussian Naive Bayes* é usado para trabalhar com dados que sejam contínuos, pois assume-se que cada classe está distribuída de acordo com a curva de Gauss. A probabilidade das *features* é obtida pela fórmula:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figura 2 - Função utilizada para calcular a probabilidade de uma determinada feature

Podemos ainda assumir diferentes valores para σ (Desvio padrão):

- É independente de Y (i.e, σi),
- Ou independente de Xi (i.e, σk)
- Ou ambos (i.e, σ)

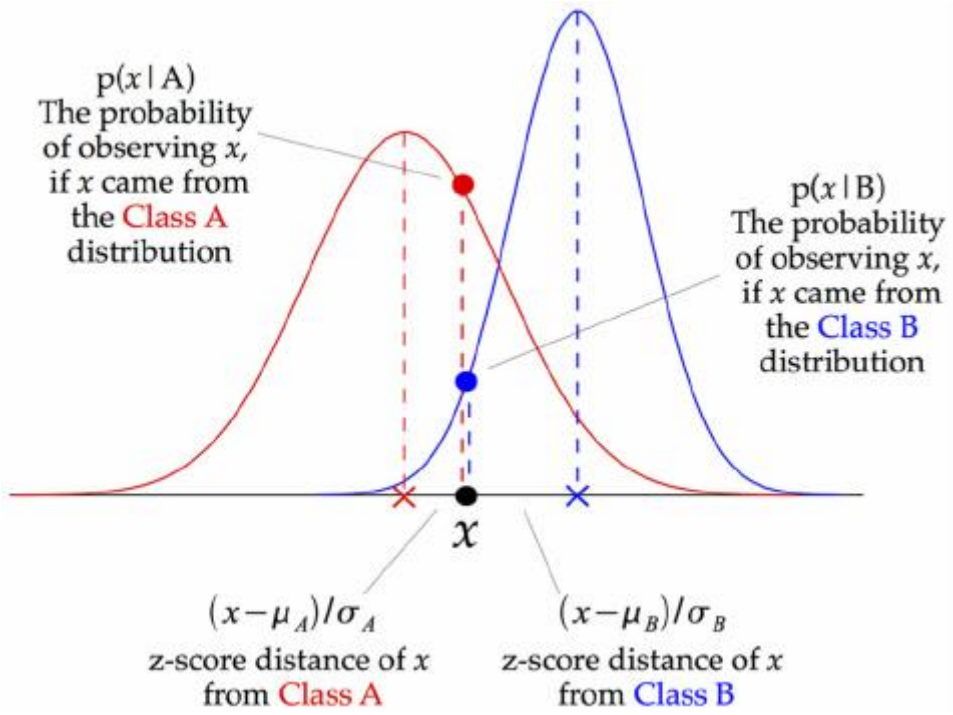

Figura 3 - Funcionamento de um modelo Gaussian Naive Bayes

Para cada classe temos uma distribuição gaussiana e para cada x temos dois pontos diferentes consoante a classe. Para calcular a pontuação de cada ponto x, calculamos a distância entre o ponto x da classe A e da classe B e a média da classe, que depois é dividido pelo desvio padrão da classe.

Análise exploratória dos dados

Descrição do conjunto de dados:

O conjunto de dados é composto por 14 atributos dos quais 13 são *features*, e 1 é *label* (match)

Tabela 1 - Descrição do conjunto de dados

Atributo	Descrição
id	número de identificação do participante
partner	número de identificação do par
age	idade do participante
age_o	idade do par
goal	Qual é o seu objetivo principal ao participar neste evento? Passar uma noite divertida = 1 Conhecer novas pessoas = 2 Conseguir um encontro = 3 Procurar um relacionamento sério = 4 Dizer que consegui = 5 Outro = 6
date	Em geral, quão frequentemente sai para encontros? Várias vezes por semana = 1 Duas vezes por semana =2 Uma vez por semana = 3 Duas vezes por mês = 4 Uma vez por mês = 5 Várias vezes por ano = 6 Quase nunca = 7
go_out	Com que frequência sai (não necessariamente para encontros)? Várias vezes por semana = 1 Duas vezes por semana =2 Uma vez por semana = 3 Duas vezes por mês = 4 Uma vez por mês = 5 Várias vezes por ano = 6 Quase nunca = 7
int_corr	Correlação entre os *ratings* de interesses (desporto, museus, caminhadas, música, filmes, livros, etc.) do participante e do seu par ([-1,1]).
length	A duração de 4 minutos para o encontro é: Demasiado curta = 1 Demasiado longa = 2 Adequada = 3
met	Já conhecia o seu par anteriormente? (0/1)
like	Quão gostou do seu par? (escala 1-10; nada = 1; muito =10)
prob	Qual é a probabilidade do seu par ter gostado de si? (escala 1-10; pouco provável = 1; muito provável =10)
match	Há *match*? (0/1) (variável objetivo)

Análise das variáveis:

A análise das varáveis foi feita usando a biblioteca pandas, do qual extraímos o número de elementos NaN de cada variável, a média, a mediana e o desvio padrão.

NaN representam os dados não numéricos, que não podem ser lidos pelos algoritmos e requerem algum tratamento.

Destes dados analisamos a percentagem de elementos NaN, e obtivemos o seguinte quadro:

	Qtd elem NaN	% de elem NaN
id	1	0.011936
partner	0	0.000000
age	95	1.133922
age_o	104	1.241346
goal	79	0.942946
date	97	1.157794
go_out	79	0.942946
int_corr	158	1.885892
length	915	10.921461
met	375	4.476009
like	240	2.864646
prob	309	3.688231
match	0	0.000000

Figura 4 - Quantidade e respectiva percentagem de elementos NaN

Com base nos resultados obtidos por esta análise, apenas length tem um número significativo de elementos NaN, sendo que as restantes variáveis têm cada uma menos de 5% de elementos NaN.

Média, mediana e desvio padrão:

	Média	Mediana	Desvio padrão
id	8.960248	8.00	5.491329
partner	8.963595	8.00	5.491068
age	26.358928	26.00	3.566763
age_o	26.364999	26.00	3.563648
goal	2.122063	2.00	1.407181
date	5.006762	5.00	1.444531
go_out	2.158091	2.00	1.105246
int_corr	0.196010	0.21	0.303539
length	1.843495	1.00	0.975662
met	0.494315	0.00	0.499999
like	6.134087	6.00	1.841285
prob	5.207523	5.00	2.129565
match	0.164717	0.00	0.370947

Figura 5 - Média, Mediana e Desvio Padrão dos Dados

Descrição dos passos de pré processamento:

O primeiro passo de pré processamento, foi retirar primeira coluna de dados que servia de índice.

Para o identificador id, que representa o número de identificação de cada candidato, qualquer elemento NaN teria que ser excluído do conjunto de dados não podendo ser-lhe imputado qualquer valor, fosse ele arbitrário, ou derivado das métricas, como a média (por exemplo), pois estaríamos a atribuir a um identificador, um conjunto de valores que podiam alterar a fiabilidade do modelo.

Para as restantes variáveis, à exceção da variável *length*, os elementos NaN são negligenciáveis, pelo que optamos por excluir os elementos NaN, pois não teriam um impacto significativo no modelo.

Para a variável *length* foram realizados testes com os modelos para ver como a imputação de diferentes métricas poderia afetar a precisão dos modelos.

Tabela 2 - Comparação da atribuição de diferentes métricas aos valores em falta de length

```
GAUSSIAN-NB
Sem NaN -> Number of mislabeled points out of a total 688 points : 116 ~ 16%
Média -> Number of mislabeled points out of a total 838 points : 149 ~ 17%
Mediana -> Number of mislabeled points out of a total 838 points : 153 ~ 18%

CART
Sem NaN -> Number of mislabeled points out of a total 688 points : 149 ~ 21%
Média -> Number of mislabeled points out of a total 838 points : 200 ~ 23%
Mediana -> Number of mislabeled points out of a total 838 points : 198 ~ 23%
```

Com base nestes resultados, conclui-se que o modelo base de ambos os algoritmos é mais preciso quando os valores NaN de *length*, são excluídos.

Experiências e resultados

CART

Implementação

Para a implementação do algoritmo foi utilizada a linguagem python com recurso ao package Python Scikit-learn. DecisionTreeClassifier é uma classe capaz de realizar classificação de vários atributos de um conjunto de dados, requerendo como paramentos de entrada uma matriz X que consiste nas amostras para os 10 atributos fornecidos do speed dating e, uma matriz Y de valores contendo os rótulos de classe para os exemplos de treino, match. Grande parte destes dados continham valores not a number, sendo estes valores considerados como em falta é necessário proceder ao seu tratamento. Foi utilizado o método dropna do pandas, que elimina as linhas de dados onde constam valores em falta. Foi também testada uma outra forma de ultrapassar esta falta de dados, utilizando o método fillna(data.mean()) do pandas, que neste caso utiliza a média de valores da respetiva coluna para preencher os valores em falta. No entanto, em termos de performance os resultados não foram superiores aos obtidos com a eliminação das respetivas linhas e como tal foi utilizado o método dropna. O relatório de classificação utilizando o método fillna será apresentado também aquando da apresentação dos resultados, figura 6.

Em algoritmos de machine learning supervisionado é comum separar o conjunto de dados em dados de treino e teste. Os dados na base de treino, são usados como o próprio nome indica, para treinar o modelo e construir a árvore, enquanto os dados na base de teste, testam a performance do modelo fora da amostra (com dados novos). A figura seguinte representa esquematicamente a divisão dos dados.

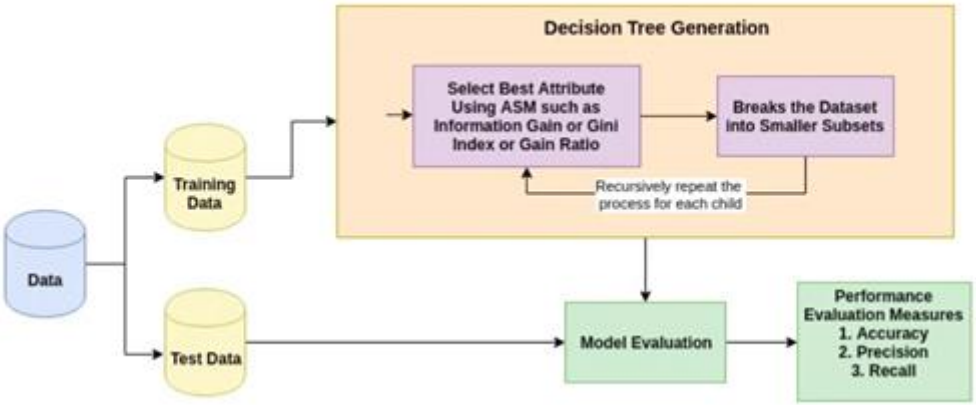

Figura 6 - Esquema de divisão de dados para construção de árvores de decisão

Fonte: www.datacamp.com.

Para esta divisão dos dados foi utilizada a técnica de test_train_split que permite então estimar o desempenho do algoritmo fazendo previsões sobre dados não usados para treinar o modelo.

Configurar train test split:

A biblioteca do scikit-learn fornece a implementação do train-test-split através da função: train_test_split(). Tem um parâmetro de configuração principal que é o tamanho do conjunto de teste. Habitualmente expresso como uma percentagem entre 0 e 1 para conjuntos de dados de teste e a restante para o conjunto dos dados de treino. Segundo alguma bibliografia não existe divisão ideal e, para este trabalho foi utilizado um conjunto de teste de 20% de forma a permitir um conjunto de treino relativamente grande.

Outra consideração importante será o número de linhas atribuídas aos conjuntos de treino e teste aleatoriamente. Isto pode ser feito selecionando a variável random_state com um valor inteiro. Para a construção da árvore não foi utilizado nenhum valor 42, para que sempre que seja executado o código seja sempre determinístico e assim será mais fácil demonstrar e avaliar os resultados.

Exatidão e precisão dos dados obtidos e otimização de performance

A performance pode ser calculada comparando os valores do conjunto de teste e os valores do modelo criado. O scikit-learn fornece métodos para o cálculo sendo que, a exatidão e precisão podem ser ajustadas variando os parâmetros no algoritmo da árvore de decisão, como por exemplo a profundidade da árvore, o número mínimo de amostras folhas, o critério para divisão dos dados que neste caso será Gini ou por exemplo, a estratégia utilizada para dividir.

A profundidade máxima da árvore pode ser usada como uma variável de controlo de pré-pruning. É importante notar que existe uma diferença de performance do modelo nas diferentes bases (treino e teste). Essa diferença que é habitualmente a favor da base de treino, uma vez que o modelo já conhece aqueles dados. Um modelo estará em underfitting quando a performance dele é má tanto na base treino, quanto na base teste. Quando aumentamos a complexidade do modelo, a performance dele melhora em ambas as bases. Contudo, um modelo muito complexo pode ficar muito ajustado para a base de treino, ou seja, ele acerta muito na base de treino mas tem pouco poder de generalização. A isto é chamado de overfitting. Na perspetiva de quem trabalha os dados, o desafio consistirá então em maximizar a precisão sem perder a capacidade de generalização. No sentido de tentar perceber a variação da exatidão com o aumento da profundidade da árvore foi construído, com os dados do trabalho, o gráfico seguinte:

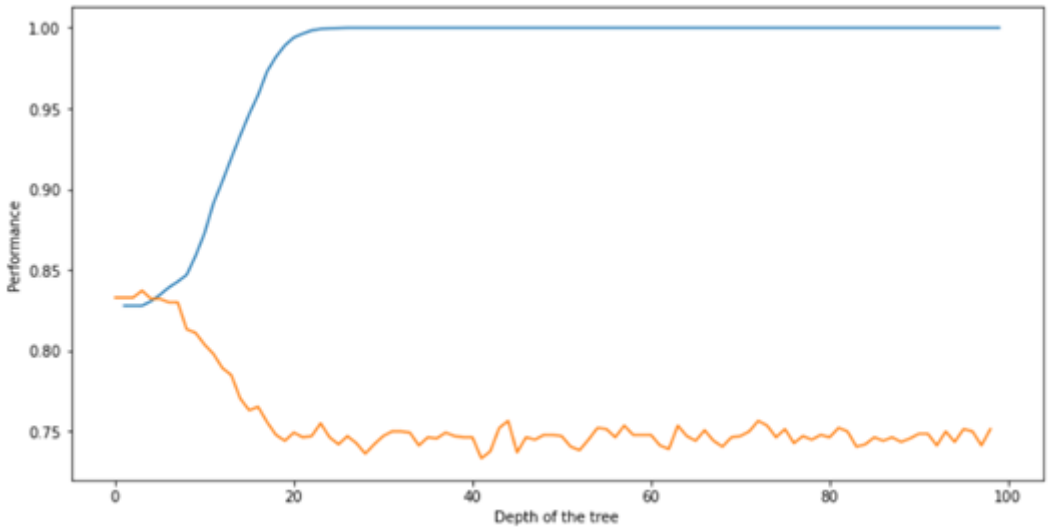

Figura 7 - Variação da performance em função da profundid*ade máxima da árvore. Amarelo dados teste, azul dados treino.*

Com linha amarela está representada a variação da exatidão com os dados de teste e, com linha cor azul a variação da exatidão com os dados de treino. Verifica-se que com o aumento da complexidade da árvore há um aumento da exatidão para os dados de treino e que, este valor estabiliza para profundidades máximas cerca de 20. Por outro lado, para os dados treino vai diminuindo. Nesse sentido optou-se por construir a árvore fixando a profundidade máxima a 20 de forma a maximizar a exatidão.

O mesmo método foi utilizado para avaliar o impacto na performance com a variação com o número mínimo de amostras folhas que basicamente especifica o número mínimo de amostras necessárias para estar num nó folha e, está então apresentado no gráfico seguinte:

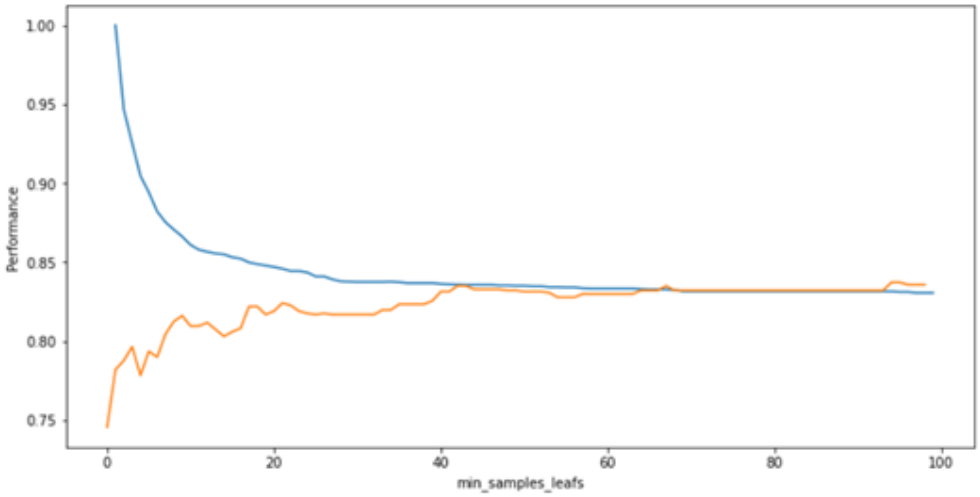

Figura *8* - Variação da performance em função do número mínimo de amostras num nó folha. Amarelo dados teste, azul dados treino.

Avaliando o gráfico acima verifica-se uma boa relação entre exatidão dos dados treino e teste com amostras mínimas nos nós folha de 20, sendo então esse o valor utilizado para a construção da árvore.

Na figura abaixo apresenta-se um enxerto da árvore de decisão criada, onde o nó raiz corresponde ao atributo like menor que 6,25 com uma impureza de Gini de 0,285. A árvore total obtida tem uma profundidade de 10 e 157 nós folha.

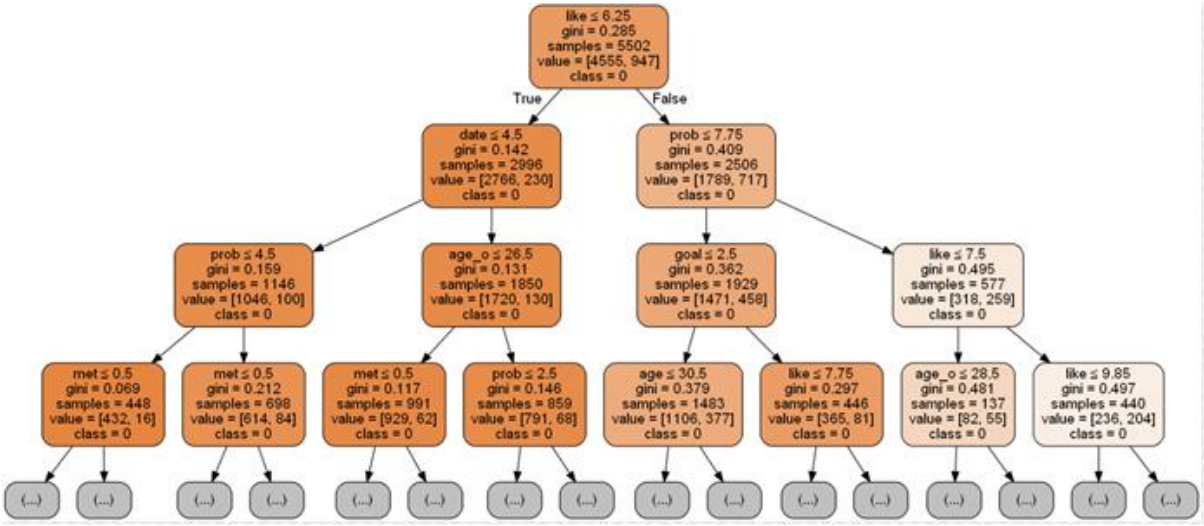

Figura 9 - Excerto da árvore de decisão obtida

Resultados obtidos:

A exatidão, que consiste então na fração de previsões corretas obtidas para o modelo criado, foi de 83,87% para dados de teste e 83,42% para dados treino.

A biblioteca do scikit-learn para machine learning permite ainda obter o relatório de classificação para a árvore construída, sendo este apresentado abaixo:

report:	precision	recall	f1-score	support
0	0.85	0.98	0.91	1146
1	0.59	0.12	0.20	230
accuracy			0.84	1376
macro avg	0.72	0.55	0.55	1376
weighted avg	0.80	0.84	0.79	1376

Figura 10 - Relatório de classificação para a árvore de decisão obtida.

report:	precision	recall	f1-score	support
0	0.84	0.98	0.91	1389
1	0.51	0.09	0.15	287
accuracy			0.83	1676
macro avg	0.67	0.54	0.53	1676
weighted avg	0.78	0.83	0.78	1676

Figura 11 - Relatório de classificação para árvore de decisão preenchendo valores NaN com média.

Precisão corresponde à capacidade do modelo rotular uma instância corretamente como match. *Recall* também referido como sensibilidade, significa basicamente a razão entre os casos positivos corretamente identificados e todos os casos positivos reais. Por outro lado, F1-score permite saber

que percentagem de previsões positivas da classe que foram corretas. Pode assim dizer-se que F1 é uma média harmónica entre precisão e *recall*:

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Precisão= True Positives / True Positives +False Positives

Recall= True Positives / True Positives + False Negatives

Exatidão= True Positives+ True Negatives/ True Positives+ True Negatives+ False Positives + False Negatives

Suporte corresponde ao número de ocorrências reais da classe no conjunto de dados teste.

A *confusion matrix* obtida encontra-se representada na figura seguinte, onde se verifica 1127 verdadeiros negativos, 203 falsos negativos, 19 falsos positivos e 27 verdadeiros positivos.

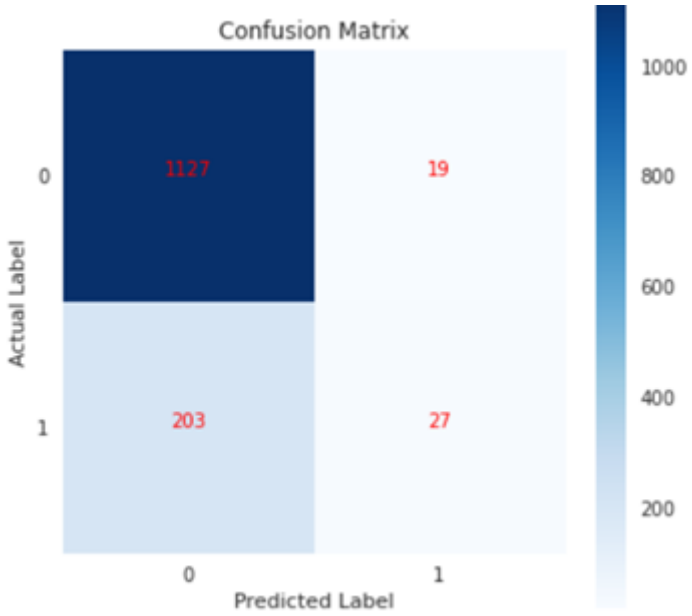

Figura 12 - Confusion matrix obtida.

Verifica-se que o modelo é bom a classificar que não há *match* mas não para classificar *match* uma vez que o número de falsos negativos é elevado tornando o *recall* muito baixo para match=1. Isto pode ser devido ao facto de o número de instâncias não match ser muito superior ao número de instâncias em que há match, permitindo ao modelo uma melhor classificação. O facto de terem sido eliminadas as linhas de dados onde contavam valores em falta também reduziu significativamente a amostra o que, pode também ter prejudicado o modelo.

Naive Bayes

Da mesma forma que no CART, os dados foram separados usando o *train_test_split*, com *test_size = 0.2* e *random_state* = 0. Obtivemos os seguintes resultados:

	precision	recall	f1-score	support
0	0.85	0.94	0.89	1130
1	0.48	0.26	0.33	246
accuracy			0.82	1376
macro avg	0.67	0.60	0.61	1376
weighted avg	0.79	0.82	0.79	1376

Accuracy: 81%

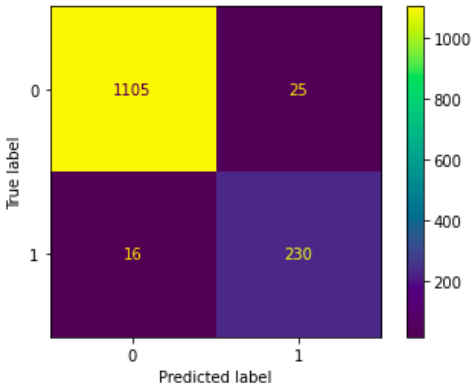

Figura 13 - Resultados obtidos pelo teste 1 NB

Nesta primeira abordagem, podemos observar que a *precision*, o *recall*, o *f1-score* e o suporte são mais altos na previsão match = 0, o que resulta num número elevado de verdadeiros negativos.

Podendo o modelo estar a ser influenciado pela ordem dos dados, foi feito outro teste onde os dados foram baralhados. Foram obtidos os seguintes resultados:

	precision	recall	f1-score	support
0	0.87	0.93	0.90	1158
1	0.41	0.25	0.31	218
accuracy			0.82	1376
macro avg	0.64	0.59	0.61	1376
weighted avg	0.80	0.82	0.81	1376

Accuracy: 82%

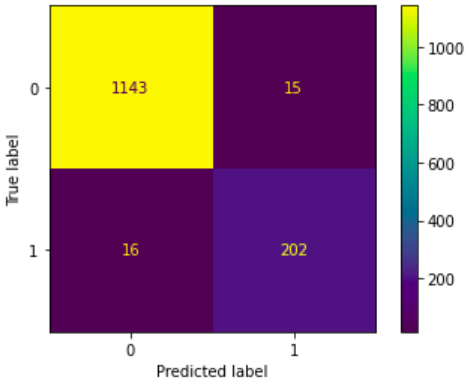

Figura 14 - Resultados obtidos pelo teste 2 NB

Após alguns testes, (correr várias vezes a mesma célula), verificamos um pequeno aumento da *accuracy*, no entanto, podemos concluir que alterar a ordem dos dados, neste caso, não afeta tanto como esperado a eficácia do modelo, no entanto é uma boa prática baralhar os resultados para qualquer modelo.

Numa terceira abordagem, alteramos alguns parâmetros da própria função *gaussianNB()*. Alterando o parâmetro var_smoothing (default var_smoothing = 1e-9), neste caso aumentando para 0.1, obtivemos:

	precision	recall	f1-score	support
0	0.85	0.99	0.91	1156
1	0.67	0.05	0.10	220
accuracy			0.84	1376
macro avg	0.76	0.52	0.51	1376
weighted avg	0.82	0.84	0.78	1376

Accuracy: 84%

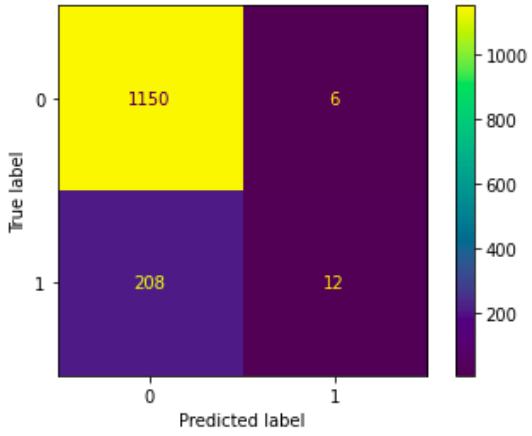

Figura 15 - Resultados do teste 3 NB

O que resulta num melhor balanceamento da precision, no entanto o recall, e o f1 score ficaram com uma diferença entre match e não match muito superior à default.

Alterando o valor para um valor inferior (var_smoothing = 1e-60), obtivemos:

	precision	recall	f1-score	support
0	0.85	0.94	0.89	1126
1	0.49	0.26	0.34	250
accuracy			0.82	1376
macro avg	0.67	0.60	0.62	1376
weighted avg	0.79	0.82	0.79	1376

Accuracy: 81%

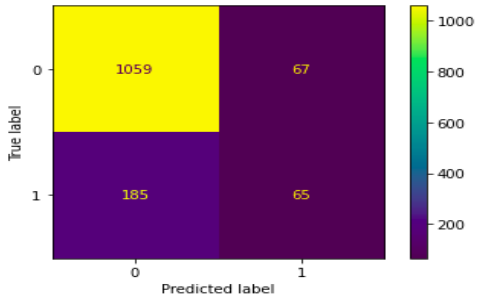

Figura 16 - Resultados do teste 4 NB

Estes resultados são espectáveis sendo que o var_smoothing aumenta ou diminui a distribuição da curva, ao longo do eixo horizontal, consoante o valor que lhe é passado. No caso de ser superior ao default, o algoritmo vai aceitar previsões que estão mais longe da curva, o que pode resultar em *underfitting*, no caso contrário, vai aceitar menos previsões do que estejam mais longe da curva, o que pode resultar em *overfitting*. Neste caso o aumento do valor *var_smoothing*, aumentou a precisão do algoritmo na previsão de match = 1. E a diminuição não resulta numa mudança muito acentuada dos valores obtidos com o valor default.

Para este modelo, chegamos à conclusão que a melhor configuração seria a *default* por ter uma melhor distribuição das métricas, e sendo a métrica *f1-score* uma média harmónica entre o precision e o recall, é um bom indicador para um bom modelo, e neste caso a *weighted avg* da *f1-score* é superior com o *var_smoothing* em *default*, e os dados *shuffled*.

Conclusões

Com este trabalho tivemos a nossa primeira interação com a linguagem python e consideramos que foi bastante útil para a realização deste tipo de trabalhos. O uso de *notebooks* para a estruturação de código é muito eficaz e fácil e permite uma análise rápida dos dados e dos resultados. As bibliotecas de python também foram uma mais-valia para este trabalho por permitirem utilizar ferramentas para estruturar e analisar os dados (pandas) e, no caso da biblioteca para *machine learning* (scikit-learn) esta também tem muitas ferramentas que permitem uma melhor análise dos modelos tal como o seu aperfeiçoamento.

Com estes modelos concluímos que ambos os algoritmos são bastante bons a prever este conjunto de dados. O *GaussianNB()* é o que requer menos alterações de parâmetros sendo que, o parâmetro existe (*var_smoothing*) tem uma alteração muito simples e muito e eficaz podendo assim resolver o overfitting ou *underfitting*. Em relação ao CART, é um algoritmo que permite maior número de alterações nos parâmetros para garantir que um modelo que não seja *overfitting* ou *underfitting*. No entanto, para este conjunto de dados ambos os algoritmos conseguem criar modelos bons e que requerem pouco "*tunning*" para obter um modelo melhor.

Em suma, concluímos que o *GaussianNB* para este conjunto de dados obteve um modelo ligeiramente melhor, por ter um f1-score melhor.

Bibliografia

Documentação sktlearn: <https://scikit-learn.org/stable/>

Documentação pandas: https://pandas.pydata.org/>

Documentação matplotlib <https://matplotlib.org/>