Relatório do 1º Projeto

Teoria da Informação

Miguel Pinto, Sofia Leitão, Tiago Silva
2022216094, 2022214134, 2022216215

Departamento de Engenharia Informática, FCTUC, 04/11/2023

Índice

Acesso e construção dos dados

Para carregar o conjunto de dados contidos no ficheiro *CarDataset.xlxs* importamos o módulo *pandas*. Através da função *read_excel(),* é guardada na variável *data* toda a informação do ficheiro, sendo esta posteriormente convertida para uma matriz (variável *matrix*). De seguida, os nomes das variáveis são armazenados numa lista (*var_names*).

Representação gráfica de variáveis em função de *MPG*

Neste exercício, são pedidos vários gráficos, todos estes relacionando *MPG* com as restantes variáveis (*Acceleration*, *Cylinders*, *Displacement*, *Horsepower*, *Model Year* e *Weight)*.
Primeiramente, a relação que *MPG* tem com a aceleração é, de um modo geral, refletida da seguinte forma: modelos com aceleração máxima entre 0 e 10 apresentam consumos de combustível superiores aos modelos cuja aceleração está compreendida entre 11 a 23. Assim, chegamos à conclusão que um carro com maior aceleração (superior a 10) é, no geral, mais eficiente quanto a *MPG*. No entanto, existem modelos com a variável entre 0 e 10 e valores de *MPG* iguais ou superiores aos compreendidos entre 11 e 23.
De seguida, quanto à relação entre *MPG* e o número de cilindros, existe um pico na quantidade de carros que possuem 4 cilindros, sendo esse o valor mais elevado na escala de eficiência de distância por combustível (*MPG).* Os modelos de 2 e 8 cilindros são os que revelam ter menor eficiência.
Quanto à relação de *MPG* com a cilindrada, o gráfico é maioritariamente decrescente, revelando que, quanto maior foi a cilindrada menor é o valor de *MPG,* tendo sempre em conta que existem modelos que não seguem a regra.
Sobre a relação de *MPG* com os cavalos de cada modelo, deparamo-nos com um gráfico semelhante aos anteriores, sendo ele maioritariamente decrescente (com determinadas exceções). Ou seja, quanto maior for a potência, menor será a eficiência combustível-distância (*MPG*).
Refletindo sobre a relação ao ano de fabrico, o gráfico obtido indica-nos que existe um aumento gradual, quase insignificante, de ano para ano, de eficiência *MPG*, pelo que um modelo produzido mais recentemente tem uma maior probabilidade de ter um maior valor de *MPG*.
Por fim, o gráfico relativo ao peso é inversamente proporcional, ou seja, quanto menor for o peso do carro, maior será o valor de *MPG*.
Para obter cada gráfico (Fig. 1) foi criada a função *data_viz,* sendo que esta recebe uma lista dos nomes das variáveis e uma matriz de dados. De

seguida, remove a variável *MPG* e gera uma matriz 3 x 2 para permitir a visualização dos gráficos de dispersão. Para organizar os dados de maneira a criar o gráfico, as variáveis são percorridas e a cada é atribuído um índice de linha e coluna correspondente à posição. Através da função *scatter(),* o gráfico de dispersão de cada iteração é colocado na sua posição. Para cada gráfico, o tamanho do ponto é 5 e tem cor roxa. O eixo x tem o título da variável que está a ser trabalhada e o eixo y tem sempre o título *MPG.*

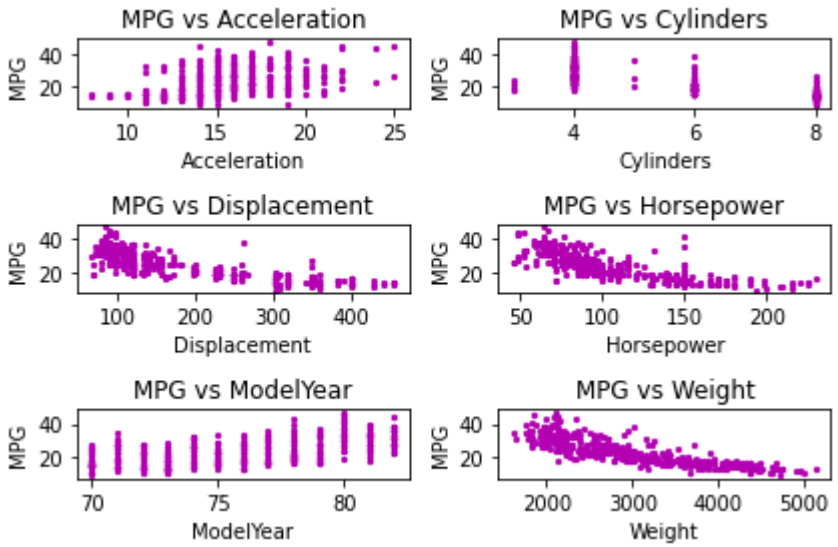

Fig. 1. Visualização dos valores de cada variável em função de MPG

Definição do alfabeto

Para definir um alfabeto apropriado para o conjunto de dados precisamos de converter o tipo de dados da variável *matrix.* Deste modo, através da função *astype(),* o tipo de dados é mudado para uint16. De seguida, o alfabeto de cada variável é definido na lista *alphabet*, que contém tuplos com valores entre 0 e o valor máximo da variável correspondente.

Cálculo de ocorrências de símbolos

Com o objetivo de calcular o número de ocorrências dos símbolos do alfabeto de cada variável, foi desenvolvida a função *count_occurences.* Esta cria um dicionário onde vamos adicionando cada variável como chave, sendo o seu valor o número de ocorrências. De seguida, percorre a lista de variáveis e cria uma chave no dicionário para cada uma delas. As colunas da matriz são percorridas e se o valor de cada iteração já tiver valor na sua chave, incrementa-a, se não, como *default* o seu valor é inicializado a 1.

Representação gráfica de ocorrências

Para representar os resultados do exercício anterior em gráficos de barras, em figuras individuais, foi criada a função *occurence_viz*. Esta percorre a lista das variáveis (*var_names*), extrai as chaves do dicionário (*occurences*), que corresponderão aos títulos no eixo x, e guarda os valores numa lista (*keys*). De seguida, repete o processo para o eixo y, guardando os valores na lista *values*.
Para formatar cada gráfico, os eixos x e y foram definidos com os valores retirados acima e as cores das barras a vermelho. Para terminar, enquanto que o título do eixo x corresponde à variável da lista *var_names* na iteração atual, o título do eixo y é sempre 'Count'.

Agrupamento de símbolos (*binning*)

É pretendido um agrupamento de símbolos, nomeadamente um *binning*, para as variáveis *Weight*, *Distance e Horsepower*. O *binning* pode ser descrito como uma técnica que envolve agrupar valores de dados em determinados intervalos chamados *bins*. O objetivo deste procedimento é simplificar a análise de dados, especialmente quando lidamos com um grande conjunto de dados contínuos.
Este método foi feito através da função *binning*, que recebe uma matriz (*matrix*) com todos os dados do documento, uma variável (*variable*), que representa a coluna que estamos a fazer o *binning,* o intervalo (*interval*), ou seja, o intervalo considerado para realizar o método e o valor máximo do dicionário da variável a ser representada (*max_val*), isto é, o *range* em que a função será aplicada.
Para que o agrupamento seja aplicado apenas nas variáveis solicitadas, só será realizado para os índices 2, 3 e 5 (valores presentes na variável *vars*). De seguida, a variável *vars* é iterada, cada coluna é transformada num array unidimensional e o valor máximo de cada uma é calculados. Caso a coluna iterada seja a da variável *Weight,* o intervalo é definido para 40, ficando definido a 5 para as outras variáveis. A Fig. 2 e a Fig. 3 demonstram, para a variável Horsepower, a diferença entre os gráficos antes e após a aplicação do processo de binning.

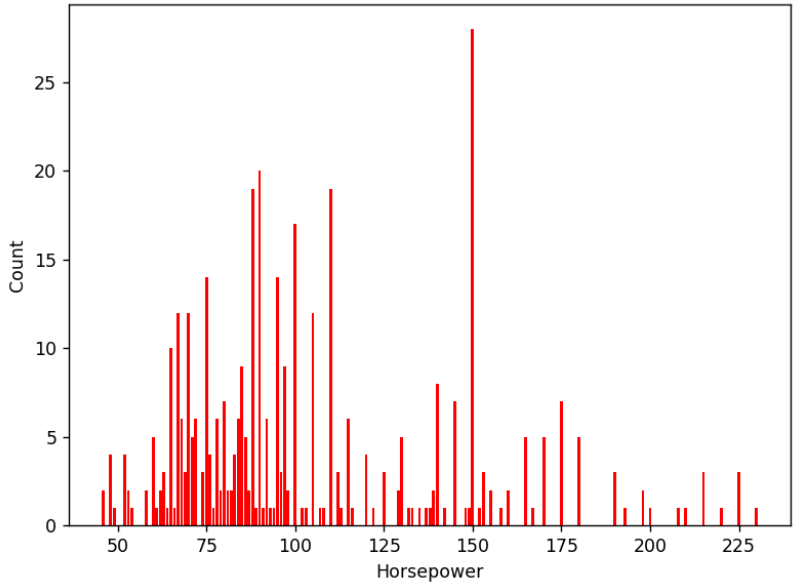

Fig. 2. Visualização de ocorrências de valores (Horsepower)

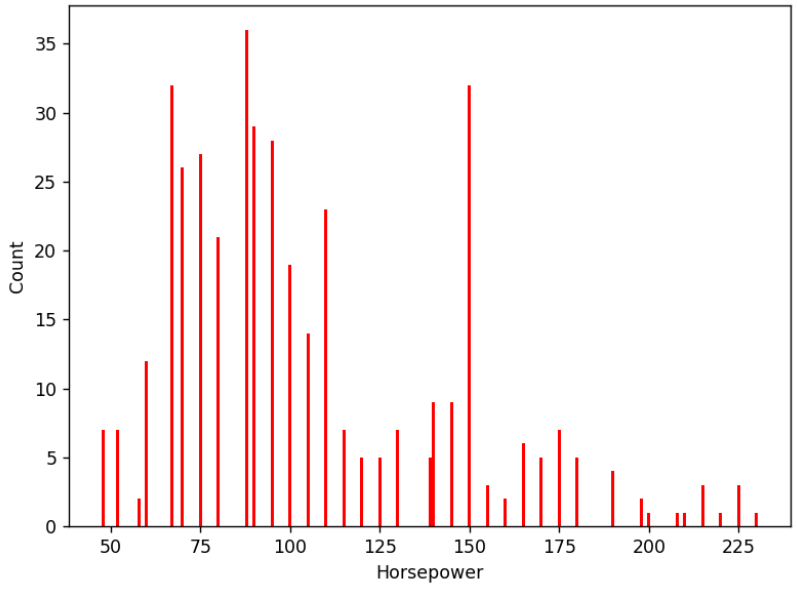

Fig. 3. Visualização de ocorrências de valores com binning (Horsepower)

Cálculo da entropia teórica por variável

A fim de calcular o valor médio de bits por símbolo, foi desenvolvida uma nova função (*entropy*), que calcula e devolve a entropia (*entropy*) de uma matriz (*matrix*) que recebe como parâmetro.

A função começa por encontrar os valores únicos (*unique_values*) e devolve o número de ocorrências de cada valor (*unique_counts*), sendo isto feito através da função *numpy.unique()*. De seguida, o número de variáveis encontradas é iterado e é calculada a probabilidade de encontrar o valor

único. Posteriormente, a entropia de cada valor único é calculada utilizando a fórmula de Shannon. Para finalizar e calcular a entropia geral, a matriz é transformada num array unidimensional (através da função *flatten()*) e a nova variável (*overall_entropy*) é passada pela função *entropy*.
Os valores de entropia de cada um dos símbolos variou bastante. Os mais surpreendentes foram Weight e Cylinders. Por um lado a variável Weight apresentou um valor algo elevado indicando uma grande variedade de valores e, como tal, uma grande incerteza. Por outro lado a variável Cylinders obteve um valor bastante baixo indicando menos variedade e mais certeza nos valores da mesma.
Quanto ao valor geral de entropia obtido, este foi, como previsto, ainda mais alto do que o da variável Weight, ou seja, significa uma grande variedade de valores, tornando o documento mais diversificado, ou seja, menos previsível.

Cálculo da média de bits / símbolo (código de Huffman)

Neste exercício foi solicitado o cálculo do número médio de bits por símbolo utilizando a codificação de Huffman. Para realizar os cálculos foram criadas duas funções: *bits_per_symbol* (calcula o número de bits por símbolo e recebe a uma matriz com todos os dados) e *variance* (calcula a variância de cada símbolo e recebe uma matriz com todos os dados e uma variável numérica).
Primeiramente, a função *bits_per_symbol* começa por guardar cada valor único e a sua ocorrência (variáveis *unique_values* e *unique_counts*, respetivamente). De seguida, são calculados o número de bits por símbolo consoante o número de *unique_values* que encontramos, este é calculado através da multiplicação do comprimento de cada símbolo, determinado pela soma de todos os valores únicos, pela probabilidade. Por fim, acrescenta à lista de bits por símbolo (*bits_per_symbol*) o valor calculado.
Em segundo lugar, a função *variance* começa por guardar cada valor único e a sua ocorrência (variáveis *unique_values* e *unique_counts*, respetivamente). De seguida, é criado um *loop* consoante o número de valores únicos encontrados e calcula a variância de cada símbolo, incrementando a variável *variance*. Calcula para cada iteração a diferença quadrada entre os bits por símbolo e pelo valor único da iteração, posteriormente, calcula a diferença quadrada através da probabilidade de encontrar o valor único. Por último, as diferenças quadradas são somadas.
Por fim, para chamar as funções de maneira a obter o resultado pretendido, as variáveis de *var_names* são iteradas e é criado o código de Huffman para

cada coluna da matriz que contém todos os dados, isto é possível através da função *HuffmanCodec(),* presente na biblioteca *huffman codec. De seguida,* os símbolos e o comprimentos dos mesmos são retirados através da função *get_code_len()*, também presente na biblioteca *huffman codec.* Depois é criado um dicionário que guarda os símbolos como chaves e o seu tamanho como a chave. Para finalizar, são calculados os bits por símbolo de cada coluna da matriz e a variância, também de cada coluna da matriz, imprimindo assim os valores para cada variável.
Como esperado e estudado nas aulas teóricas, podemos observar que os valores obtidos para cada variável são, neste ponto, limitados inferiormente pela entropia. De forma a reduzir o valor da variância podemos pensar em estratégias como agrupamento de símbolos (que pode não ser ideal por levar a problemas de memória), ou colocar os símbolos combinados na lista usando a ordem mais elevada possível de modo a obter variância mínima. A importância de minimizar a variância prende-se com o facto de ser importante otimizar a eficiência de codificação uma vez que resulta numa distribuição mais uniforme dos comprimentos de cada símbolo, minimizar o tamanho de armazenamento uma vez que a mensagem codificada é menor e manter o desempenho de transmissão consistente uma vez que uma variância alta pode levar a flutuações no tempo de transmissão de símbolos.

Cálculo de coeficientes de correlação de Pearson

É pretendido o cálculo dos coeficientes da correlação de Pearson entre a variável *MPG* e as restantes variáveis. Esta foi calculada através de um loop no qual é recorrida a função *corrcoef()* da biblioteca *huffmancodec,* que calcula a correlação entre dois conjuntos de dados, sendo eles a variável *MPG* (*matrix[:, 6]*) e a variável atual (*matrix[:, i]*). A variável *rowvar* quando é *true*, especifica que cada linha no array de entrada representa a variável, e que a coluna representa a observação. A matriz [0,1] é usada para extrair o coeficiente da correlação da matriz, entre a linha 0 e a coluna 1.

Cálculo da informação mútua com MPG

Para calcular a informação mútua entre a variável *MPG* e as restantes variáveis, a função *mutual_information()* é criada, que recebe como parâmetros uma matriz (*matrix*) e as variáveis a testar (*var_1* e *var_2*).
Inicialmente, é criada uma matriz (*prob_matrix*) com as dimensões do valor máximo de *var_1* pelo valor máximo de *var_2*, composta por apenas 0s. Para contar as ocorrências das diferentes combinações de *var_1* e *var_2*, é criado um *loop* que itera as colunas de *bin_matrix* correspondentes a cada variável, incrementando *prob_matrix*[i][j] quando *var_2* = i e *var_1* = j.
Por fim, com o intuito de calcular a informação mútua, são criados dois *loops*, que iteram sobre os valores de 0 até *mpg_max* (inclusive) e sobre os valores de 0 a *var_max* (inclusive), correspondendo estas variáveis ao valor máximo do alfabeto da sua variável, nos quais é calculada a probabilidade de cada combinação criada pelos iteradores dos *loops*, sendo esta a contagem de ocorrências da combinação em *prob_matrix* e a divisão deste resultado pelo número de linhas da matriz (*size*), correspondente ao número total de modelos no *dataset*. Finalmente, para calcular a informação mútua, a cada iteração é incrementada a variável *mi* com a fórmula da probabilidade:

$\sum_{x \in X} \sum_{y \in Y}$ p(x, y).log [p(x, y) / p(x).p(y)]. Os resultados revelam que embora os coeficientes de correlação de Pearson e a informação mútua estejam alinhados em termos de direção das relações, a informação mútua fornece uma perspetiva mais ampla e sensível à dependência geral entre as variáveis. Podemos concluir que os resultados obtidos neste ponto são, de forma geral, coerentes com os coeficientes de correlação de Pearson.

Cálculo de *MPG* previsto e comparação de resultados

Os valores estimados de *MPG* em função das restantes variáveis pode ser obtido utilizando a relação que nos é dada:

$$\begin{aligned} MPGpred = {} & -5.5241 - 0.146 * Acceleration - 0.4909 * Cylinders \\ & + 0.0026 * Distance - 0.0045 * Horsepower + \\ & 0.6725 * Model - 0.0059 * Weight \end{aligned}$$

Após calculá-lo para cada modelo no *dataset*, são apresentados 3 resultados diferentes: o cálculo original, a remoção da fórmula da variável com menos informação mútua e a remoção da variável com maior informação mútua,

sendo que em cada caso é também apresentada a diferença correspondente com o valor real. A função *mi_info* permite a aplicação das variações da relação original, bem como a impressão dos dados obtidos na consola, incluindo o *MPG* real médio, o *MPG* previsto médio, a diferença média entre os dois e a sua variação absoluta, para além de uma *preview* de cada matriz. Os *MPG*s previstos e a sua diferença relativamente a aquele que se verifica são guardados em dois arrays (*pred_mat* e *diff_mat*), ambos inicializados com 0s e com as mesmas características que a coluna 6 da matriz de dados, onde vão ser guardadas as previsões do *MPG* e a diferença do valor real e do previsto, respetivamente.

Em primeiro lugar, para obter os valores das previsões, a fórmula dada é utilizada sobre cada iteração da matriz *pred_mat*, recorrendo aos valores na linha correspondente da matriz *real_mat*. De modo a chegar aos valores da comparação são subtraídos os valores de *pred_mat* aos valores reais de *MPG*, que se encontram na 6ª coluna de *real_mat*.

Em segundo lugar, a variável com a informação mútua menor *(Acceleration*) é removida. Assim, o valor de cada previsão de *MPG* é ajustado devido à omissão da variável no seu cálculo, e a diferença entre o valor real e o previsto é recalculada.

Por último, é utilizado um processo semelhante ao passo anterior para remover a variável com a maior informação mútua (*Weight*). A diferença está no ajuste feito, onde é readicionado o produto do valor na coluna de índice 0 pelo seu multiplicador e retirado o produto que afeta a 5ª coluna, correspondente à variável.

Refletindo sobre os resultados alcançados, destaca-se na aplicação integral da fórmula do *MPG* previsto a semelhança entre ambos os valores médios, sendo que a média da diferença absoluta é significativamente superior à sua análoga, o que evidencia a flutuação entre valores de diferença negativos e positivos. Por outro lado, contrapondo ambos os atos de remover as variáveis portadoras da menor e maior quantidade de informação mútua com o valor real de *MPG*, é evidente o quanto o último afeta o resultado previsto quando comparado com o primeiro, evidenciando a informação mútua como um indicador do impacto que uma variável tem no valor de outra.