MC886 - Relatório do Assignment 01
Regressão Linear

Pedro Stramantinoli P. Cagliume Gomes
175955
p175955@dac.unicamp.br

Ruy Castilho Barrichelo
177012
r177012@dac.unicamp.br

I. Introdução

Apesar de ser um algoritmo relativamente simples no campo do Aprendizado de Máquina, a Regressão Linear ainda é constantemente estudada e utilizada. Como descrito em seu próprio nome, é aplicável em problemas de regressão, nos quais se busca uma previsão de um valor de saída real, ao contrário de um problema de classificação, e, neste caso, consiste em um aprendizado supervisionado - em que são disponibilizadas, além das entradas, as respostas correspondentes, - e se baseia na construção de uma hipótese a partir da realização de um treinamento com um conjunto de dados fornecidos, a qual é então utilizada para estimar valores mediante novos valores até então desconhecidos.

Esse processo visa a minimização da função de Custo, que indica o erro quadrático médio entre as previsões obtidas pelo modelo obtido e as respostas reais, e pode ser aperfeiçoada pelo cálculo dos parâmetros associados às *features* do problema tratado. Esse cálculo é realizado, no escopo deste trabalho, de modo iterativo pelo método de Gradient Descent e analiticamente, pela Equação Normal.

Mesmo com sua simplicidade, este algoritmo possibilita a análise de diversas técnicas e detalhes do estudo prático de aprendizado de máquina, como maior entendimento sobre o impacto de números de iterações, de *learning rates*, de *feature scaling* e normalização, *Cross Validation*, entre outras, como visto a seguir, em um problema de predição de preços de diamantes, mediante algumas de suas características fornecidas de acordo com o *dataset* [2] fornecido.

II. Tratamento dos Dados

Em relação ao tratamento dos dados, dividiu-se o processo em algumas etapas. Após o carregamento dos arquivos originais, realizou-se o processo de *Dummy Coding*, nos quais as informações sobre cor, claridade e corte (*color, clarity, cut*) foram transformadas em valores inteiros a partir das strings inicialmente disponíveis para cada *feature*, de acordo com a ordenação disponibilizada no site do *dataset*, na seção *"Columns"* [2], isto é, estipulando maiores valores inteiros para informações que indicasse maior qualidade. Foi devido à existência dessa ordenação que optou-se pelo uso de *Dummy coding*, em oposição a *One-hot encoding*, pois desejou-se enfatizar o maior peso para dados indicativos de boa qualidade.

Em seguida, ambos dados de treino e teste foram separados em matrizes de *Features* e de *Targets*, sendo que aos primeiros foram adicionadas novas colunas, correspondentes a novas *features* (nesse caso, *features* em graus superlineares).

Então, os dados foram normalizados, por meio da subtração de cada valor pela média correspondente ao atributo em questão (*Mean Normalization*), o que garante médias aproximadamente nulas para os valores das *features*. Com isso, os dados foram submetidos a *Feature Scaling*, para que todos estivessem em escalas similares, especificamente entre -1 e 1, já que foi utilizada a técnica de dividir os valores pelo máximo valor do atributo a que pertencem. Por fim, foram adicionadas colunas de 1, referentes ao x_0, como último passo, para que não houvesse perda de valores em função dos processos anteriores.

No que tange à divisão do conjunto de dados de treino, foram realizadas partições, mais especificamente 5 delas, a partir da técnica de KFold *(K = 5)*, isto é, dividem-se os dados, após serem embaralhados, em *K* partes, das quais *K-1* são destinadas para testes e 1 para validação (como visto na figura abaixo), para que se pudesse realizar *Cross Validation* durante o treinamento dos modelos, processo que permite a validação e, consequentemente, possíveis melhorias nos resultados, sem uso do conjunto de testes. Ainda, essa técnica pode levar à obtenção de conjuntos mais representativos para treinos e validação, amenizando a chance de *overfitting*.

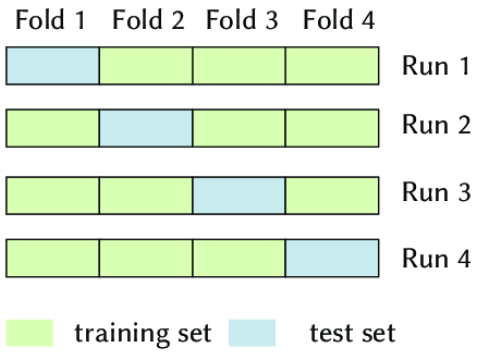


Figura 1. Ilustração do método de KFold, para K = 4 [5]

III. Experimentação e Resultados

No total foram utilizados cinco modelos. A tabela 1 mostra a média dos erros encontrados pela equação Batch Gradient

Descent (GDB) de própria implementação e a média dos erros encontrados pela Equação Normal para cada modelo.

Tabela I
ERROS DOS MODELOS PELOS MÉTODOS DE GRADIENT DESCENT E DE EQUAÇÃO NORMAL

Erros dos Modelos Estudados		
Modelo	Gradient Descent	Eq. Normal
1	8712513	769293
2	3193199	1296307
3	8861793	72142575
4	2529907	69899193
5	2278643	27983412

O Modelo 1 foi calculado a partir do uso de todas as *features* disponíveis do banco de dados, tratadas de forma linear, isto é, o expoente de cada *feature* é igual a um. Optou-se por se iniciar com um modelo simples, que não só confirmasse o funcionamento das funções implementadas, mas também servisse como base para os modelos seguintes, permitindo comparações conforme se aumentasse a complexidade do modelo estudado.

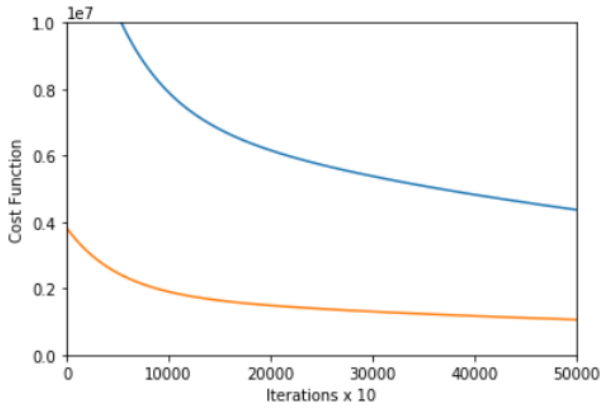
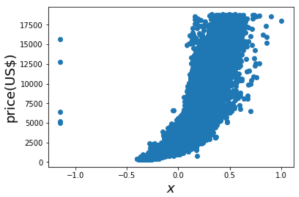
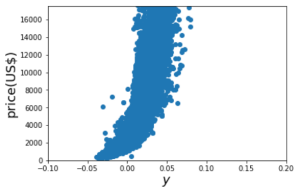


Figura 2. Custo por iteração do Modelo 1. Conjunto de testes em azul, conjunto de validação, em laranja.

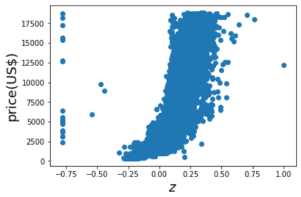
Devido à necessidade de melhor entender a relação entre as *features* e o preço dos diamantes, isto é, o *target*, foram construídos gráficos do tipo: *Target* em função de *Feature*, para cada uma delas, como visto a seguir:



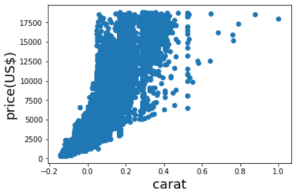
(a) Preço em função da Feature X



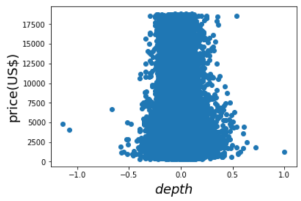
(b) Preço em função da Feature Y



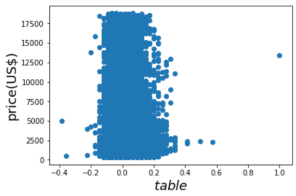
(c) Preço em função da Feature Z



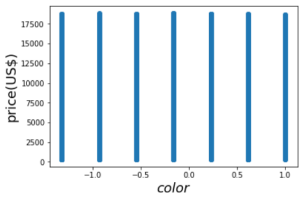
(d) Preço em função da Feature Carat



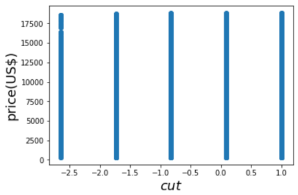
(e) Preço em função da Feature Depth



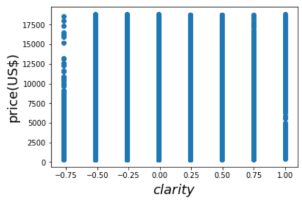
(f) Preço em função da Feature Table



(g) Preço em função da Feature Color



(h) Preço em função da Feature Cut



(i) Preço em função da Feature Clarity

Com isso, pode-se perceber que os gráficos obtidos para *table, depth, clarity, color, cut* não aparentam, a priori, ter relação direta com o *target*, já que ou não seguem a forma de uma função, ou exibem pontos em diversos preços para a mesmo valor da *feature*, logo, experimentou-se retirá-las do modelo para estudar o impacto de sua presença.

Também, ao se analisar os gráficos relativos às *features x, y, z, carat*, percebe-se que não exibem relações lineares, mas sim superlineares, logo, manter os expoentes com valor 1 não estaria de acordo com o que foi observado. Dessa forma, foi gerado o modelo 3, que se utiliza das mesmas *features* do modelo anterior, porém com expoentes quadráticos, para analisar a hipótese do comportamento superlinear retirada dos gráficos.

No entanto, para o próximo modelo, após nova reflexão, concluiu-se que a desconsideração de *features table, depth,*

clarity, color, cut nos modelos foi impremeditada, uma vez que não pode-se garantir, apenas a partir dos gráficos - que são recortes simplificados - que elas não têm influência alguma, já que há a possibilidade de que, ao serem consideradas em conjunto às outras, demonstrarem relações com o valor do *target*. Porém, isso seria de difícil visualização gráfica, ao se tratar de últiplas dimensões acima da terceira, por haver pelo menos 9 *features*.

Com essas considerações, foi gerado o modelo 4, no qual essas *features* então desconsideradas retornaram, de forma linear, enquanto foram mantidas as *features* anteriores com expoentes quadráticos.

Por fim, no modelo 5, tentou-se explorar graus maiores, devido a uma maior análise dos gráficos das *features x, y, z, carat*, nos quais os valores exibem um crescimento muito elevado, obtendo-se um modelo bem mais complexo que os anteriores. Pode-se ver o gráfico de Custo em função do nũmero de iterações tanto em relação ao conjunto de treino quanto ao de validação, a seguir:

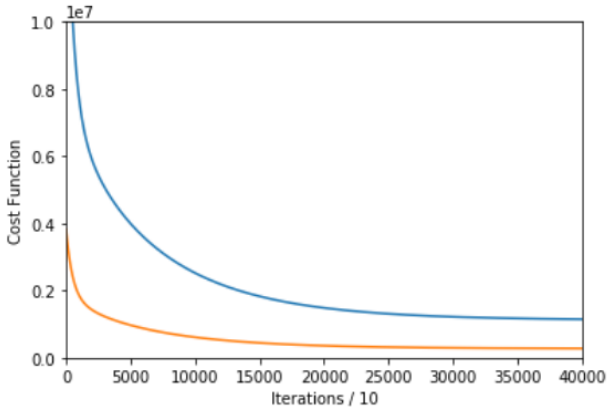


Figura 3. Custo por iteração do Modelo 5. Conjunto de testes em azul, conjunto de validação, em laranja.

Como último passo, escolheu-se um modelo, que foi utilizado para ser executado com o conjunto de testes, ainda intocado. A seguir, pode-se observar o resultado dos erros no conjunto de treino, de validação e de teste.

Erros finais para os diferentes conjuntos de dados		
Conjunto de dados	GD	Normal
Validação	2278643	27983412
Teste	3380401	7939853
Scikit	2004407	—

IV. Análise e Comparações

Dados os resultados apresentados anteriormente, podem-se apresentar comparações não só entre modelos, como também entre soluções por GD e por Equação Normal, além dos erros para o modelo final nos diferentes conjuntos de dados (validação e teste), e a comparação com a implementação disponibilizada pelo Scikit-Learn [4].

Inicialmente, ao se analisar a Tabela 1 em relação aos diferentes métodos usados, percebe-se que para os modelos lineares, a Equação Normal se mostrou mais precisa, além de ser mais rápida em execução. Porém, conforme a complexidade do modelo cresceu, nota-se que sua performance cai drasticamente e sua precisão torna-se bem inferior ao GD. Possíveis causas são aumento das dimensões da matriz devido à introdução de mais *features*, introdução de não-linearidade nos dados de entrada ou ainda, como observado durante a execução nos *KFolds*, usualmente uma das iterações gerava um erro significativamente mais alto que as outras, elevando a média do conjunto, possivelmente devido a um arranjo desfavorável da matriz para inversões ou para a resolução da equação em geral. Ainda, a resolução por meio da Equação Normal pode ter perdido precisão na validação caso tenha encontrado um mínimo global de modo que tenha gerado um *overfitting* no modelo, isto é, tenha obtido parâmetros com vieses muito altos para o conjunto de treino, de modo que, quando testado com novos dados, apresentou erros significativos.

Já sobre a evolução entre os modelos, nota-se que a retirada das *features* que não aparentavam influência sobre o *target* causou uma redução no custo, do modelo 1 para o 2. No entanto, o modelo 3, ainda com tais *features* removidas, apresentou maior erro, o que fortalece a hipótese de que há uma influência, sim, desses atributos no problema, mas, como também foi alterada a complexidade do modelo ao introduzir não-linearidade, não se pode afirmar com certeza que apenas esta alteração é responsável pelo impacto observado. Outro ponto que vai ao encontro dessa suposição é que a reintrodução dessas *features* nos modelos seguintes, embora de modo linear, coincidiu com a redução de erros. Por fim, pode-se observar que o aumento de complexidade está associado à diminuição do erro entre os modelos 4 e 5, uma vez que a utilização de monômios de maior grau resultou em uma nova diminuição de custos, como esperado pela observação dos gráficos de *Target* em função de *Features*.

Em relação ao modelo final, pode-se perceber que, de fato, houve aumento do erro entre o conjunto de validação e o de testes, como esperado, já que dificilmente o modelo não estará com algum viés para o conjunto a partir do qual foi treinado e validado, isto é, as alterações de modelos e aperfeiçoamentos são realizadas sobre estes conjuntos, os quais são constantemente observados durante a atividade, dessa forma, quando apresentado a dados novos e previamente desconhecidos, obtêm-se erros maiores.

Como última análise, quando realizada a comparação com a implementação do Scikit-Learn, observa-se que ambos valores encontram-se na mesma ordem de grandeza, embora a implementação do módulo tenha um resultado um pouco maior do que a metade do obtido pela implementação própria. Assim, pode-se considerar que são resultados adequados, ao se considerar que a comparação foi realizada com um módulo bem consolidado, eficiente e muito utilizado.

Por fim, um fato que chama a atenção é o de que, inicialmente, foi estipulado que um modelo seria executado por muito tempo, visando atingir um *overfit*, para que fosse conhecido um limite aproximado do número de iterações

máximo a ser usado, no entanto, apesar de diversas execuções com número de iterações e tempos de execução altos, esse ponto nunca foi atingido. Isto é, buscava-se, no gráfico de Custo por número de iterações, um comportamento no qual o custo no conjunto de validação aumentasse a partir de certo ponto, enquanto o de treino permanecesse baixo, o que é um indicativo de *overfitting*, uma vez que mostra um alto viés do modelo com os dados de treino, como visto na figura a seguir.

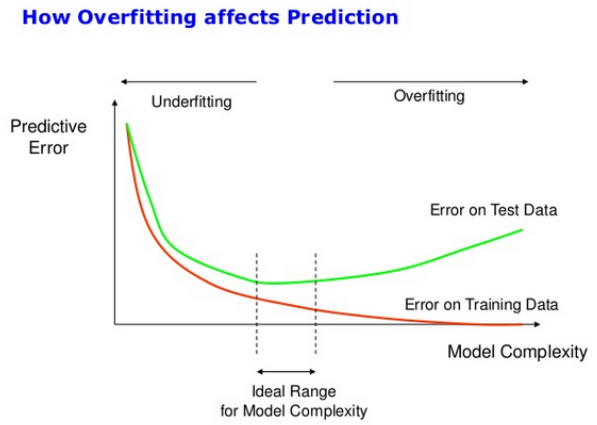


Figura 4. Como o *overfitting* afeta a predição. [6]

V. Conclusão

A partir do trabalho realizado e dos resultados obtidos, foi possível não só observar como ocorre o processo de aprendizado de máquina na prática, mas também a consolidação do entendimento sobre as aplicações vistas em aula. Isto é, pode-se notar, claramente, a influência da complexidade do modelo nos resultados, o impacto da alteração do *learning rate* e do número de iterações nos custos e nas velocidades tanto de convergência, quanto de execução dos algoritmos, além das noções sobre *underfitting* e *overfitting*, detalhes, estes, dificilmente perceptíveis em um contexto apenas teórico.

Ao passo que as atividades eram realizadas, observou-se, também, o nível de dificuldade para criar e melhorar modelos, principalmente devido ao longo tempo de execução dos algoritmos -para os maiores números de iterações- e à falta de conhecimento sobre o campo de estudo do problema. Ou seja, foi, de certo modo, contra as expectativas iniciais de complexidade, diante de um problema que era considerado, então, relativamente simples, mas em que não foi possível obter resultados bons (baixos custos).

Ainda, no que tange à consolidação dos conhecimentos vistos em aula, notou-se a importância e relevância dos dados para a resolução do problema, pela maior facilidade e praticidade do trabalho quando os dados estão organizados e não exigem muito tratamento e pela influência que a escolha de *features* causa em cada modelo.

Por fim, o resultado final - os erros no conjunto de teste e sua comparação com a implementação advinda do Scikit-Learn - foi satisfatório, na medida em que se encontram na mesma ordem de grandeza e relativamente próximos.

Referências

[1] Aurélien Géron. ”Hands-On Machine Learning with Scikit-Learn and TensorFlow”. O’Reilly Media, Inc. March 2017

[2] shivamagrawal. ”Diamonds. Analyze diamonds by their cut, color, clarity, price, and other attributes.”https://www.kaggle.com/shivam2503/diamonds

[3] NumPy v1.15 Manual. https://docs.scipy.org/doc/numpy/

[4] Documentation of scikit-learn 0.19.2. http://scikit-learn.org/stable/documentation.html

[5] The technique of KFold cross-validation, illustrated here for the case K = 4. https://www.researchgate.net/figure/The-technique-of-KFold-cross-validation-illustrated-here-for-the-case-K-4-involves_fig10_278826818

[6] How overfitting affects prediction.https://stats.stackexchange.com/questions/292283/general-question-regarding-over-fitting-vs-complexity-of-models