Projeto 1 - MC886

Victor Ferreira Ferrari
RA 187890
vferrari@mpc.com.br

Vinícius Couto Espindola
RA 188115
vinicius.c.e@hotmail.com

I. Introdução

O trabalho tem como objetivo principal explorar modelagem de dados para o uso de regressão linear para fins de análise preditiva. O conjunto de dados a serem explorados consistem do tráfego de uma seção da auto-estrada "Interstate 94", no estado de Minnesota, nos Estados Unidos, associados à uma série de dados meteorológicos junto com informações de datas e feriados. No caso, o tráfego é o valor objetivo que deve ser previsto através de um modelo de regressão linear pelo dados de clima e data. Nota-se que este modelo deve ser generalizável, ou seja, não se restringe apenas ao conjunto de dados fornecidos para treino.

II. Condições e Conjunto de Dados

Na implementação do modelo, foram utilizadas diversas bibliotecas externas da linguagem *Python*, sendo essas `NumPy`, para operações vetorizadas, `Pandas`, para leitura de CSV e manipulação de dados, `MatPlotLib`, para criação de gráficos demonstrativos, `scikit-learn`, para comparação com modelo já implementado de regressão linear e implementação do cálculo de métricas de qualidade, e `holidays`, para informações sobre feriados no estado de Minnesota, nos Estados Unidos. Além disso, bibliotecas padrão da linguagem também foram utilizadas.

O conjunto de dados consiste de 48204 exemplos, cada um com 8 *features* e o valor alvo. Os atributos são:

- Feriado (se o dia é feriado, apenas presente na primeira hora do dia);
- Temperatura (em Kelvin);
- Chuva (por hora);
- Neve (por hora);
- Nuvens (Porcentagem);
- Descrição curta da situação climática;
- Descrição detalhada da situação climática;
- Data/Hora.

III. Implementação

Inicialmente, foram separados os conjuntos de validação e treino a partir do *dataset* original. Para evitar a repetição dessa divisão em toda execução, isso foi feito em um *script* separado, dividindo o arquivo de extensão CSV em 2 (ou 3, opcionalmente possuindo suporte a conjunto de teste). A separação foi feita selecionando aleatoriamente elementos distintos do conjunto original.

Para a *feature scaling* dos dados testamos três métodos: *min-max normalization*, *mean normalization* e *standardization*. Nota-se que a normalização foi feita sobre variáveis específicas que são passadas como argumento da função de normalização evitando variáveis discretas/classificatórias que deveriam ser mantidas como zero ou um.

Para a regressão linear foram implementados três tipos de gradient descent: **Batch (BGD)**, **Mini-Batch (MBGD)** e **Stochastic (SGD)**. O Mini-Batch pode ter seu tamanho definido e, tanto para o SGD quanto para o MBGD, os dados são percorridos de forma aleatória, o que é feito com o auxílio da biblioteca `random` do Python (permuta-se os índices das amostras). Ressalta-se que todos os modelos foram vetorizados ao máximo por questões de otimização. Foram definidos dois critérios de parada: limite de tempo e limite de Epochs. Com a finalidade de análise dos modelos, foram criadas funções que registram o tempo percorrido e os novos gradientes obtidos a cada Epoch.

A equação normal tem uma função dedicada onde, com auxílio do NumPy, é verificado se a multiplicação da transposta da matriz de *features* com a própria é invertível e, caso seja, realiza as contas vetoriais para obtenção dos coeficientes ótimos.

Foi utilizado também, para comparação, um modelo da biblioteca `SKLearn` de regressão linear por SGD. Esse modelo possui funções para prever valores e fornecer o R^2 score.

Em relação à métodos de comparação, a função de custo não aparentou uma boa métrica para averiguar a qualidade da regressão em relação as previsões esperadas, uma vez que o valor que esta retornava não tinha bons referenciais de comparação, a utilizamos apenas para avaliar a convergência dos modelos. Para qualificar a regressão obtida, utilizamos o R^2 score e o *root mean square error* (RMSE), além de verificações extras de *Explained Variance Score* e *Median Absolute Error* pois estes apresentavam dados com referenciais de comparação melhores.

No caso, o R^2 score classifica a regressão entre $[-\infty, 1]$, sendo 1 uma regressão perfeita (como o `SKLearn` implementa essa métrica ela pôde ser usada para comparação), já o RMSE aponta o erro médio absoluto de predições, que se torna útil se soubermos os valores alvos das predições. O *Variance Score* é uma métrica interessante de comparação com o R^2, pois é uma conta similar, mas retirando o erro médio da conta. Valores similares de R^2 e *Variance Score* significa erro médio baixo. O erro mediano é interessante pois é robusto contra *outliers*.

A fim de possibilitar uma análise mais intuitiva dos dados, modelos e outras implementações, visualizamos dados através

de gráficos. Foram implementadas diversas funções que chamam os métodos implementados e coletam dados para realizar a visualização deles. Utilizamos tais funções para tarefas como análise da *learning rate* por modelo, comparar os tipos de GD, análise de convergência e identificação de anomalias.

IV. Soluções Propostas

A priori, realizamos o tratamento dos dados: removemos valores absurdos como temperaturas em zero absoluto e precipitações surreais.

A abordagem inicial para modelar a regressão consistiu de "expandir"as 8 *features* fornecidas pelo conjunto de dados de forma que fosse possível extrair toda informação do conjunto de dados. Os valores de data e hora, por exemplo, foram transformadas em variáveis discretas que representam os dias da semana e as horas do dia, visando identificar dias (fins de semana por exemplo) e horas de pico. Para tal, utilizamos a função weekday() da biblioteca `datetime`. A *feature* que indica feriados foi transformada em uma variável discreta, e, para facilitar a implementação em conjuntos embaralhados, utilizamos a biblioteca `holidays` para marcar as datas que se tratavam de feriados. Das descrições de clima, utilizamos alguns valores mais específicos e outros mais gerais ainda tendo em mente a "expansão dos dados". Variáveis classificatórias (weather_description) foram trasnformadas em valores discretos. Alguns *timestamps* estavam duplicados pois podiam assumir mais de uma *weather_description* ao mesmo tempo, resultando em duplicatas de horas cuja única mudança era a condição climática. Foi considerada apenas a primeira ocorrência. A trasformação dessas variáveis classíficatórias em discretas possibilitou que estes detalhes fossem representados. Ao final da modelagem, o resultado foi setenta e cinco *features* (incluindo o bias).

Com os dados "expandidos"ao máximo, teríamos a maior parte das informações do modelo à nossa disposição. Espera-se que tenhamos um modelo bom, por este poder representar a maior parte das informações que o conjunto fornece, mas também há a possibilidade de *overfitting*, uma vez que a complexidade excessiva gerada pelo número de *features* possa fazer com que a regressão fique muito restrita ao conjunto de treino. Todavia, no caso de um modelo com *overfitting*, podemos isolar as *features* de forma que mantenhamos apenas aquelas que realmente generalizem para os outros conjuntos. Do modelo complexo gerado poderíamos extrair um modelo mais simples que fosse otimizado mais rápido.

Então, foi criado um segundo modelo para simplificar, criar *features* baseadas na visualização dos dados e utilizar na implementação os comandos vetorizados da biblioteca `Pandas`.

È perceptível a diferença entre o tráfego em dias úteis (maior) e fins de semana (menor), então foi criada uma variável classificatória que identifica dias úteis, para substituir as variáveis de dias da semana. Foram criadas variáveis classificatórias para descrever horário de pico (15h-17h), horário comercial (6h-18h) e noturno (0h-5h) para substituir as variáveis para cada hora do dia, e foram utilizadas apenas as descrições climáticas curtas. Por fim, a variável que descrevia data e hora foi dividida em hora, dia, mês e ano, variáveis numéricas, permitindo remover a variável que classificava feriados.

Posteriormente, após análise de resultados e testes, as variáveis classificatórias de horário de pico, comercial e noturno foram trocadas pelas mesmas variáveis do primeiro conjunto de *features*, para cada hora do dia, totalizando 45 *features* (incluindo o bias).

V. Testes e Seleção de Parâmetros

Com três opções de GD à disposição, buscamos escolher o que melhor se adaptasse ao nosso modelo. Para tal, precisaríamos regular os hiperparâmetros: limite de epochs, limite de tempo, *learning rate* e batch size no caso do MBGD. Para analisá-los, utilizamos duas listas em que foram armzenados os coeficientes calculados ao final de cada de cada epoch e o tempo no término do epoch a partir do início da execução do GD.

Para a análise do *learning rate* foram feitos três gráficos (um para cada tipo de GD durante trinta segundos de execução) com quatro curvas de otimização para diferentes valores de *learning rate*. Os resultados podem ser visto na figura 1.

Com os valores de *learning rate* para cada um dos tipos de GD, podemos comparar os três em sua melhor convergência. Para comparar exatamente o funcionamento dos três, podemos convergência com mesmo*learning rate*. Os resultados estão na figura 2

Em relação ao BGD, observa-se que este é o mais rápido em relação à execução de operações finalizando muito mais epochs em menos tempo quando comparado aos concorrentes. Outra vantagem do BGD é sua descida estável que permite o uso de um *learning rate* alto que converge rapidamente. O SGD, mesmo apresentando uma descida instável para *learning rate* maior ou igual à um décimo, converge bem rápido mesmo em LR baixo. Outro detalhe do SGD é que este é o menos otimizado para execuções de instruções, demorando muito mais tempo para completar o mesmo número de epochs que o BGD. Ainda assim, com menos de 5% do número de epochs que o BGD, o SGD consegue minimizar tão bem quanto ou melhor que o BGD. O MBGD é o gradiente que aproveita o melhor de ambos outros. Regulando manualmente o tamanho do batch chegamos ao tamanho de 5% do número de amostras. Nestas proporções, o MBGD consegue desfrutar de descidas estáveis com LR alto assim como o BGD, e também desfruta de descidas extremamente rápidas como o SGD. Com o MBGD podemos combinar LR alto com altas taxas de atualização dos coeficientes, além de ser apenas cerca de 50% mais lento que o BGD.

Para a escolha do método de normalização, buscamos aquele que melhor aproximou as escalas das *features* e que deixou menos valores próximos à zero (o que poderia causar instabilidade numérica gerando coeficientes muito grandes). A *standardization* não aproximou as escalas das suficientemente: enquanto algumas *features* estavam com valores entre [-1,1] outras ficaram no intervalo [0,82]. A *mean normalization* deixou muitos valores próximos a zero e não aproximou

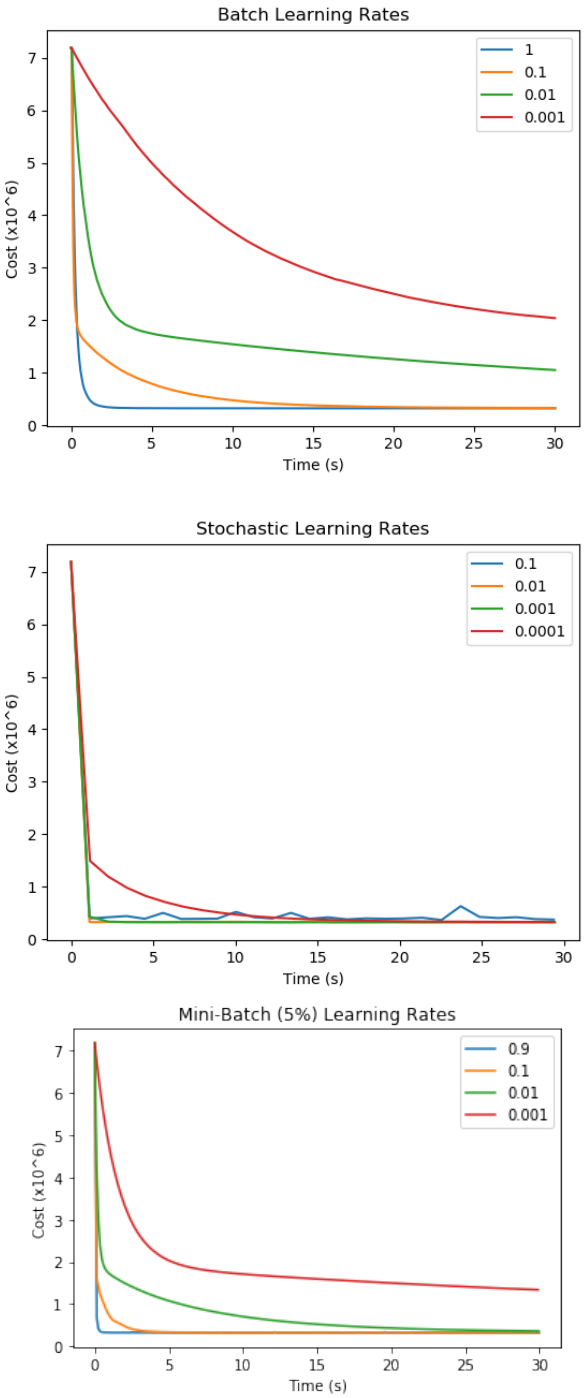

Figura 1. Função de custo em função do tempo para diferentes valores de *learning rate*

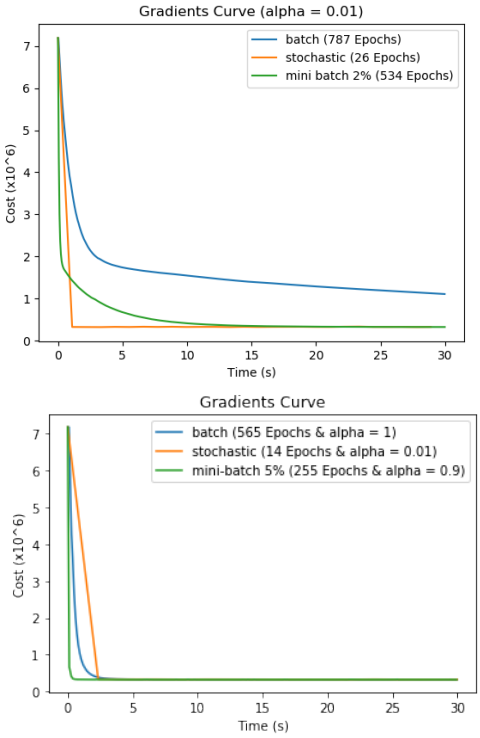

Figura 2. Comparação entre BGD, SGD e MBGD

bem as escalas entre os valores. já a *min-max normalization* garantiu uma boa proximidade de escalas e proporções, sendo também a que retornou os melhores resultados.

Seguindo o material sugerido na página do conjunto de dados, nao obtivemos nenhum valor anômalo: no conjunto de tráfego, apenas dois valores excediam a margem de dois desvios padrões e nenhum excedeu três desvios. O único aspecto incomum foi de que cerca de 50% dos valores se encontravam entre um e dois desvios. No geral, o conjunto de dados aparentou bem distribuido.

VI. Experimentos e Discussões

Inicialmente, foram comparados os dois conjuntos de *features* com os mesmos hiperparâmetros: LR igual a 0,001, limite de epochs igual a 10000 e limite de tempo igual a 30s, utilizando SGD. A comparação foi feita utilizando R^2 score no conjunto de validação, e o primeiro conjunto de *features* se mostrou melhor, alcançando *score* de 0,829 enquanto o segundo alcançou 0,774.

Posteriormente, foram encontrados os melhores valores de *learning rate* para cada tipo de GD, e com isso foi possível comparar os diferentes tipos de GD, como discutido na seção V.

Foi então comprovado que o MBGD é uma boa escolha neste caso, convergindo mais rápido que os outros com um R^2 score similar.

Com o modelo definido, foram feitos os testes finais, com LR igual a 0,1 (pois em 0,9 ainda há bastante instabilidade),

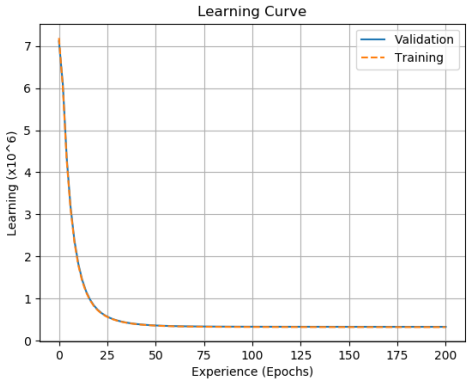

Figura 3. Curva de aprendizagem do modelo implementado.

limite de epochs igual a 10000 e limite de tempo igual a 30s, utilizando MBGD com batch size igual a 5% do conjunto total. A curva de aprendizado está na figura 3. Os resultados foram:

- R^2 Score: 0,829
- Variance Score: 0,829
- RMSE: 815,233
- Erro Mediano: 441,009

Em relação à *overfitting*, o modelo inicial apresentou resultados melhores que o esperado. Observando a curva de aprendizado do modelo para os conjuntos de treinamento e validação, percebemos que o modelo generalizou bem para o problema em questão. Além da curva de aprendizado, comparamos o R^2 score para uma comparação mais precisa, e a perda para o conjunto de validação foi menor do que 10^{-3}. Apesar do grande número de *features* do modelo, este conseguiu representar bem o conjunto de dados sem perder a generalidade. Nota-se que não foi utilizado nenhum polinômio de grau maior que 1 durante os testes, uma vez que houve *underfitting* e que encontrar um polinomio de maneira eficiente em uma quantidade tão grande de *features* seria trabalhoso.

Observamos que as predições que estão fora de uma margem de 25% do valor $real$ ($y < 0.75 \cdot real$ ou $y > 1.25 \cdot real$), compõem cerca de 30% dos casos de treino. Desses 30%, 80% são predições que estão acima de $1.25 \cdot real$. Visualizando alguma predições, percebemos que os casos em que mais ocorrem erros são os que apresentam tráfego muito abaixo da média e as predições são relativamente altas. Dos 80%, o tráfego real estava abaixo de 2000 em quase todos. O mesmo padrão se repete para o conjunto de validação. Infelizmente, não tivemos tempo para explorar estas amostras e entender o porquê do modelo não representá-los adequadamente.

Por último, esse modelo foi comparado ao SGDRegressor da biblioteca `SKLearn` pelas mesmas métricas e com os mesmos hiperparâmetros, exceto LR (LR=0,01). Os resultados deste modelo foram:

- R^2 Score: 0,828
- Variance Score: 0,829
- RMSE: 816,471
- Erro Mediano: 459,209

Percebe-se que os resultados são bem similares em todas as métricas de qualidade. A principal diferença é o tempo gasto em cada um: o modelo implementado gastou os 30s de tempo limite, e o SGDRegressor parou muito mais rápido, possivelmente por utilizar incremento mínimo como critério de parada. O LR foi modificado pois, por ser SGD e não MBGD, requer *learning rate* menor para atingir o melhor resultado.

Não foi possível aplicar a equação normal no modelo inicial, uma vez que este não apresentou uma matriz `X.T.dot(X)` invertível. No segundo modelo, mesmo havendo inversa, a equação normal não retornava solução boa, com score de -800000. Foi então feito um terceiro modelo, sem variáveis climáticas, feriados ou representando cada hora, mas com os horários de pico, comercial e noturno citados na seção IV. As *features* representando hora, dia, mês e ano continuam, assim como a que indica dia útil.

No terceiro modelo, a equação normal forneceu resultado. Comparando os resultados dos 3 métodos testados, temos que são muito similares, dados por:

- R^2 Score: 0,744 (normal) ou 0,743 (outros)
- Variance Score: 0,744 (normal) ou 0,743 (outros)
- RMSE: 999,1 (normal) ou 1000,4/1000,6 (outros)
- Erro Mediano: 503,5 (normal) ou 505,7/502,9 (outros)

Vê-se então que a equação normal é um bom método de regressão, seu principal problema sendo a dificuldade de encontrar um conjunto de *features* que seja ao mesmo tempo adequado para o método e represente bem o que se está modelando.

VII. Conclusões

Apesar do modelo apresentar bons resultados nos conjuntos de treinamento e validação, o conjunto assumiu um número muito alto de *features*, que são possívelmente desnecessárias, fazendo com que a matriz resultante seja singular. Os casos para os quais as predições não foram adequadas apresentavam um padrão, o que pode implicar que o modelo representa bem apenas um subconjunto do problema. Também é possível que este modelo ou o próprio dataset apresentem um erro ou complexidade inerente para que seja difícil de ser modelado. Para explorar estes aspectos do problema, o caminho ideal seria testar mais conjuntos de *features* simples e complexos, e aumentar a complexidade do modelo por regressão polinomial de grau maior que 1.

Em relação aos modelos implementados, observamos que há um tradeoff entre a velocidade em que os coeficientes são obtidos e a qualidade deles. Todavia, com *learning rate* adequado, vê-se que a qualidade do modelo é bem similar entre os três tipos de GD.

Enfim, o melhor tipo de GD depende da aplicação e do tempo disponível, porém neste caso percebe-se que o MBGD é um bom meio-termo entre a rapidez do SGD e a convergência do BGD.