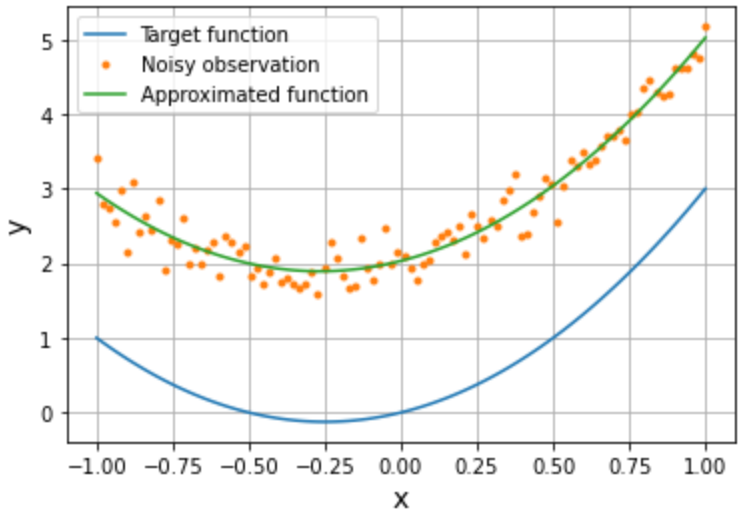
TP555 - AI/ML

Lista de Exercícios #2

Regressão Linear

1. Observe a figura abaixo. Se a função objetivo (i.e., *target function*) é dada por $y = x + 2x^2$, por quê a função aproximadora (i.e., *approximated function*) não se aproxima da função objetivo? **Justifique sua resposta.**



2. Dada a seguinte função hipótese h(x) = a0 + a1*x + a2*x^2 e os valores de x e y:

x = np.array([[-1.], [0.], [1.]])	y = np.array([[-0.1], [1.2], [6.5]])

Responda:

1. Que tipo de sistema de equações é esse? Subdeterminado, determinado ou sobredeterminado? Justifique.
2. Quais sãos os valores de a0, a1 e a2?
3. Qual o EQM entre y e a saída da função hipótese?
4. Baseado no valor do EQM, você acredita que esta seja uma observação ruidosa?

3. Qual técnica de regressão linear você usaria se tivesse um conjunto de treinamento com milhares de atributos e milhões de exemplos? Explique por quais razões você utilizaria esta técnica.
4. Suponha que os atributos do seu conjunto de treinamento tenham escalas muito diferentes. Qual abordagem de regressão linear pode sofrer com isso e como? O que pode ser feito para mitigar este problema?

5. Suponha que você use o ***gradiente descendente em batelada*** e plote o erro de cada época. Se você perceber que o erro aumenta constantemente, o que provavelmente está acontecendo? Como você pode resolver isso?
6. Entre as versões do gradiente descendente (GD) que discutimos (batch, estocástico e mini-batch), qual deles chega mais rapidamente à vizinhança da solução ótima? Qual deles realmente converge? O que você pode fazer para que os outros também convirjam?
7. Em sala de aula, nós discutimos os 3 tipos de algoritmos baseados no gradiente descendente, batch, estocástico e mini-batch, porém, o código do mini-batch foi o único que não foi apresentado. Portanto, neste exercício eu peço que você:
    a. Implemente o algoritmo do mini-batch. Crie uma implementação em forma de função ou classe para que você consiga reutilizar sua implementação mais facilmente.
    b. Teste suas implementações com a seguinte versão ruidosa da função objetivo y = 2*x1 + 2*x2 + w, onde x1, x2 e w são M = 1000 valores retirados de uma distribuição aleatória Gaussiana normal padrão (i.e, com média 0 e variância igual a 1) e utilizando a função hipótese h = a1*x1 + a2*x2. O objetivo aqui é verificar que o algoritmo do mini-batch também se aproxima do valor ótimo (obtido com a equação normal) quando o valor do passo de aprendizagem é ajustado para seu valor ótimo. (**Dica**: plote o gráfico das curvas de contorno com o histórico dos valores dos pesos)
    c. Plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 para cada iteração do mini-batch, e o gráfico de iteração versus erro,
    d. Encontre manualmente (tentativa e erro) o valor ótimo do passo de aprendizagem (**Dica**: utilize os gráficos da superfície de contorno com os parâmetros a1 e a2 para cada iteração do mini-batch e o gráfico de iteração versus erro para saber se aquele passo é o ótimo. Acesse os links abaixo para entender como você pode plotar os gráficos de contorno.),
        i. https://matplotlib.org/3.1.1/gallery/images_contours_and_fields/contour_demo.html#sphx-glr-gallery-images-contours-and-fields-contour-demo-py
        ii. https://www.python-course.eu/matplotlib_contour_plot.php
    e. Compare os resultados do mini-batch com os resultados obtidos com o GD em batelada (batch) e GD estocástico (**Dica**: para a comparação, use os códigos que estão nos slides da aula e plotem os gráficos da superfície de contorno com os parâmetros a1 e a2 para cada iteração, ou seja, o histórico de atualização dos pesos, e o gráfico de iteração versus o erro para GD em batelada e estocástico).
    f. Baseando-se nos gráficos do item anterior, a que conclusões você pode chegar quanto ao treinamento dos 3 tipos de gradiente descendente?
8. Dada a seguinte função hipótese e assumindo o erro quadrático médio como função de erro

$$h(x) = a_0 + a_1 x + a_2 x^2.$$

Encontre as equações de atualização dos pesos/parâmetros para esta função. Em seguida, utilizando os vetores x e y definidos abaixo, encontre os parâmetros a0, a1 e a2 através do método da regressão de forma fechada (equação normal) e com ***gradiente descendente em batelada***.

$$y(x) = 3 + 1.5x + 2.3x^2 + w,$$

onde x é um vetor coluna com M = 1000 valores retirados de uma distribuição aleatória uniformemente distribuída no intervalo de -5 a 5 e w é outro vetor coluna com M valores retirados de uma distribuição aleatória Gaussiana com média 0 e variância igual a 10.

a. Plote o gráfico do número de iterações versus o erro quadrático médio.
b. Baseado no gráfico acima, encontre manualmente o melhor valor para o passo de aprendizagem. (**Dica**: encontre empiricamente, ou seja, através de tentativa e erro, o valor do passo que faça com que o erro decresça rapidamente nas primeiras épocas e que após algum tempo se estabilize, se tornando quase constante.)

9. Neste exercício você vai utilizar o arquivo ***training.csv*** onde a primeira coluna são os valores de x (feature) e a segunda de y (label). Baixe o arquivo do endereço: training.csv. Após, leia o conteúdo do arquivo, ou seja, os vetores x e y, com os seguintes comandos:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('training.csv', header=None)

x = df[0].to_numpy()
y = df[1].to_numpy()

fig = plt.figure(figsize=(10,10))
plt.plot(x, y, 'b.')
plt.show()
```

Em seguida, utilize o algoritmo do ***gradiente descendente em batelada*** para encontrar os parâmetros de cada uma das seguintes funções hipóteses.

a. h = a0 + a1*x (polinômio de ordem 1)
b. h = a0 + a1*x + a2*x^2 (polinômio de ordem 2)
c. h = a0 + a1*x + a2*x^2 + a3*x^3 (polinômio de ordem 3)
d. h = a0 + a1*x + a2*x^2 + a3*x^3 + a4*x^4 (polinômio de ordem 4)

onde x^b = x^b.

Para cada uma das funções hipótese acima faça o seguinte:

a. Encontre os valores ótimos dos parâmetros através do método de forma fechada, i.e., equação normal, ou também conhecida como método dos mínimos quadrados.
b. Encontre as equações de atualização dos parâmetros/pesos assumindo o erro quadrático médio como função de erro.
c. Encontre o valor ótimo do passo de aprendizagem.

d. Plote um gráfico que mostre x vs. y e x vs. h, ou seja, um gráfico comparando os dados originais com a estimativa (i.e., hipótese) da função que gerou y.
e. Plote um gráfico com o número de iterações versus o erro quadrático médio.

Em seguida responda às seguintes perguntas

A. Qual das funções-hipótese acima aproxima melhor a função alvo (target), ou seja, qual produz o menor erro ao final do treinamento?
B. Dado que você encontrou os parâmetros que otimizam cada uma das funções hipótese acima (ou seja, você agora tem um modelo treinado que pode predizer o resultado para novos exemplos (generalizar)), use os dados contidos no arquivo predicting.csv e calcule o erro quadrático médio para cada um dos modelos (i.e., função hipótese). Qual função hipótese resulta no menor erro quadrático médio de validação, ou seja, com os dados do arquivo predicting.csv? Crie um gráfico que mostre os erros de treinamento e validação versus a ordem do polinômio. O que você consegue concluir a respeito dos valores plotados?

10. Neste exercício você irá utilizar a esquema de decaimento programado do passo de aprendizagem conhecido como ***decaimento exponencial*** juntamente com o algoritmo do ***gradiente descendente estocástico*** com ***critério de parada*** definido como sendo quando a ***diferença absoluta*** entre o erro da iteração atual e a anterior caia abaixo de 0.001 ou que o número máximo de iterações tenha sido atingido. Defina o número máximo de iterações como

maxNumIter = n_epochs*M,

onde n_epochs = 1 e M = 1000. Dada a seguinte função observável

y_noisy = 2.5*x1 + 1.3*x2 + w,

onde x1, x2 e w são variáveis aleatórias seguindo a distribuição Gaussiana normal padrão, ou seja, com média zero e variância unitária. Gere M = 1000 exemplos a partir desta função observável. Agora faça o seguinte:

a. Plote a superfície de erro considerando a seguinte ***função hipótese***: h(x1,x2) = a1*x1 + a2*x2.
b. Encontre os valores ótimos dos pesos através da fórmula fechada, ou seja, a equação normal.
c. Treine o modelo utilizando apenas o algoritmo do ***gradiente descendente estocástico***.
d. Plote os seguintes resultados:
    i. A ***superfície de contorno*** mostrando a evolução da atualização dos pesos.
    ii. O gráfico de erro versus número de iterações necessárias para que o critério de parada seja atingido.
    iii. O gráfico da variação da atualização (i.e., alpha*vetor_gradiente) versus o número de iterações necessárias para que o critério de parada seja atingido. Mostre a variação da atualização para cada peso em figuras separadas.
e. Agora, treine o modelo utilizando o algoritmo do ***gradiente descendente estocástico*** com os esquema de decaimento programado do passo de

aprendizagem conhecido como ***decaimento exponencial***. A fórmula matemática deste esquema é dada por

alpha = alpha_initial * exp(-k * t),

Onde k é a taxa de decaimento da exponencial, t é o número da iteração e alpha_initial é o valor inicial do passo de aprendizagem. **OBS**.: Não se esqueça de encontrar os melhores valores (aqueles que façam com que o tempo de convergência e a oscilação do GDE diminuam) de alpha e k.

f. Plote todos os resultados listados no item (d) para esta implementação do gradiente descendente estocástico com decaimento exponencial.
g. Imprima os valores dos pesos encontrado com a equação normal, com o GDE e o GDE utilizando decaimento exponencial.
h. Qual das duas versões precisa de menos iterações para que o critério de parada seja atendido?
i. Qual valor dos pesos obtidos pelo GDE e GDE com decaimento exponencial é próximo ao do obtido com a equação normal?

DICA: Para que você tenha o mesmo comportamento para ambas versões do algoritmo do GDE (sem e com o decaimento exponencial), “reinicie” o gerador de números pseudo-aleatórios antes do laço de repetição que implementa cada uma das versões do GDE, com a função seed no módulo random da biblioteca NumPy: np.random.seed(42).

11. Neste exercício você irá aplicar o ***escalonamento mín-max*** aos atributos de treinamento de um modelo de regressão linear. Dada a seguinte função observável

y_noisy = x1 + x2 + w

onde x1 é um vetor coluna com M amostras retiradas de uma distribuição Uniforme no intervalo [-5.0, 5.0), x2 é um vetor coluna com M amostras retiradas de uma distribuição Uniforme no intervalo [-0.5, 0.5) e w é também um vetor coluna com M amostras retiradas de uma distribuição Gaussiana Normal com média 0 e variância unitária. Gere um conjunto de treinamento com M = 1000 exemplos. Utilize o algoritmo do **gradiente descendente em batelada** com a seguinte função hipótese

h(x1, x2) = a1*x1 + a2*x2,

com a1 e a2 iniciais iguais a -10 e -10, respectivamente.

a. Sem aplicar escalonamento de features aos exemplos de treinamento, plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 encontrados durante as iterações (ou seja, o histórico de valores que o algoritmo encontra durante o treinamento do modelo) para o conjunto de treinamento e o gráfico de erro quadrático médio versus o número de iterações. **OBS.1**: Não se esqueça de encontrar, manualmente, o valor ótimo para o passo de aprendizagem. Este deve ser o maior valor possível que não faça o algoritmo oscilar entre os vales. **OBS.2**: Não se esqueça também, de calcular o valor ótimo dos pesos, através da equação normal, e plotá-los no gráfico de contorno juntamente com o histórico dos pesos.
b. Aplique a ***normalização min-máx*** aos atributos de treinamento, plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 encontrados durante as iterações e o gráfico de erro quadrático médio versus o

número de iterações. **OBS.1**: Não se esqueça de encontrar, novamente, o valor ótimo para o passo de aprendizagem. Este deve ser o maior valor possível que não faça o algoritmo oscilar entre os vales. **OBS.2**: Não se esqueça também, de calcular novamente o valor ótimo dos pesos, através da equação normal, e plotá-los no gráfico de contorno juntamente com o histórico dos pesos.

c. Existe diferença entre os formatos da superfície de erro sem e com escalonamento?
d. Houve diminuição do número de iterações necessárias para a convergência?
e. Se a ***padronização*** fosse aplicada ao invés da ***normalização***, como ficaria o formato da superfície de erro e o número necessário de iterações para a convergência do algoritmo?

12. Neste exercício você irá aplicar ***escalonamento de features*** aos dados de treinamento e teste. Dada a seguinte função objetivo (ou modelo gerador)

$$y = x1 + x2,$$

onde x1 é um vetor coluna com M amostras retiradas de uma distribuição Gaussiana com média 0 e desvio padrão unitário e x2 é um vetor coluna com M amostras retiradas de uma distribuição Gaussiana com média 10 e desvio padrão igual a 10. Gere dois conjuntos de dados (aleatórios), com M = 1000 amostras cada. Um dos conjuntos será utilizado para treinamento e o outro para teste, ou seja, validação do modelo treinado. Ou seja, você deve criar um conjunto y_treinamento = x1_treinamento + x2_treinamento e outro conjunto y_validação = x1_validação + x2_validação. Use sementes diferentes para gerar os conjuntos de treinamento e validação (e.g., np.random.seed(42) e np.random.seed(84)). Utilize o **gradiente descendente em batelada** com a seguinte função hipótese

$$h = a1*x1 + a2*x2,$$

com a1 e a2 iniciais iguais a -20 e -20, respectivamente. Para todos os casos abaixo, treine os modelos com o mesmo número máximo de iterações, por exemplo, 2000 iterações e um critério de parada que faça o algoritmo parar quando o erro de treinamento entre duas épocas consecutivas for menor do que 0.001, ou seja, o algoritmo irá parar se o erro for menor do 0.001 ou se atingir o número máximo de iterações. Pede-se

a. Sem aplicar nenhum escalonamento de features aos exemplos de treinamento, plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 encontrados durante as iterações (ou seja, o histórico de valores que o algoritmo encontra durante o treinamento do modelo) para o conjunto de treinamento e o gráfico de erro quadrático médio versus o número de iterações para os conjuntos de treinamento e teste. **OBS.1**: Não se esqueça de encontrar, manualmente, o valor ótimo para o passo de aprendizagem. **OBS.2**: Não se esqueça de encontrar o valor ótimo dos pesos e plotá-los no gráfico de contorno com o histórico dos pesos.
b. Aplique a normalização min-máx às features de treinamento e teste, plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 encontrados durante as iterações para o conjunto de treinamento e o gráfico de

erro quadrático médio versus o número de iterações para os conjuntos de treinamento e teste. **OBS.1**: Não se esqueça de encontrar o valor ótimo para o passo de aprendizagem. **OBS.2**: Não se esqueça que o conjunto de testes é normalizado com os valores mín-máx encontrados para o conjunto de treinamento. **OBS.3**: Não se esqueça de encontrar o valor ótimo dos pesos/parâmetros e plotá-los no gráfico de contorno com o histórico dos pesos.

c. Aplique a padronização às features de treinamento e teste, plote a superfície de erro, a superfície de contorno com os parâmetros a1 e a2 encontrados durante as iterações para o conjunto de treinamento e o gráfico de erro quadrático médio versus o número de iterações para os conjuntos de treinamento e teste. **OBS.1**: Não se esqueça de encontrar o valor ótimo para o passo de aprendizagem. **OBS.2**: Não se esqueça que o conjunto de testes é padronizado com os valores de padronização encontrados para o conjunto de treinamento. **OBS.3**: Não se esqueça de encontrar o valor ótimo dos pesos/parâmetros e plotá-los no gráfico de contorno com o histórico dos pesos.
d. Repita os itens b e c aplicando desta vez a normalização min-máx e a padronização, respectivamente, também aos targets/rótulos (ou seja, os valores de y).
e. Baseado nos resultados anteriores, o que você pode concluir a respeito do escalonamento de features? (**Dica**: Comente a respeito das formas das superfícies de erro, dos números de iterações necessárias para se alcançar o ponto ótimo, isso se ele é alcançado, da diferença entre o erro quadrático médio obtido para o conjunto de treinamento e o obtido para o conjunto de testes (são similares ou diferentes), da diferença entre os valores do erro quadrático médio para os 3 casos acima, i.e., sem escalonamento e com os 2 tipos de escalonamento com e sem escalonamento dos labels (qual resulta no menor erro? Escalonar os labels traz algum benefício? Como ficam as superfícies de erro quando se escalona os labels?), e o que mais você achar interessante comentar. Quanto mais detalhada sua análise dos resultados, melhor será sua avaliação neste exercício.)

13. Sistemas de comunicação digital usam modulação IQ para transmissão de sinais. Neste exercício, vamos entender o desequilíbrio IQ (do inglês, IQ Imbalance) e seu efeito no sinal demodulado.

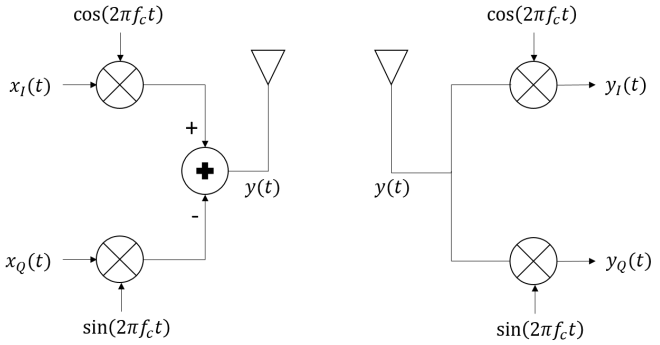
Modulação e demodulação IQ

A modulação e demodulação IQ típica é mostrada na figura abaixo. Considere que a informação a ser transmitida, $x(t)$, é um sinal complexo,

$$x(t) = x_I(t) + jx_Q(t).$$

A saída do modulador IQ é dada por

$$y(t) = \Re\left\{x(t)e^{j2\pi f_c t}\right\} = x_I(t)cos(2\pi f_c t) - x_Q(t)sin(2\pi f_c t)$$



No receptor, multiplicamos o sinal $y(t)$ com $cos(2\pi f_c t)$ e $sin(2\pi f_c t)$ em cada ramo, respectivamente, seguido por filtragem passa-baixa, a qual não consideramos aqui neste exercício, para extrair os sinais $y_I(t)$e $y_Q(t)$. Ignorando um fator de escala de 1/2, podemos recuperar $x_I(t)$ e $x_Q(t)$.

A modulação/demodulação IQ é baseada na ideia de que um sinal real em banda passante é convertido para banda base com duas ondas senoidais de igual amplitude e com diferença de fase de 90 graus em relação uma à outra.

Para sinais IQ balanceados, nós obtemos

$$y_I(t) = y(t)cos(2\pi f_c t),$$

$$y_Q(t) = y(t)sin(2\pi f_c t).$$

Quando os sinais estão equilibrados (ou balanceados), as duas ondas senoidais são ortogonais uma à outra e carregam informações de sinal ortogonais (independentes). Quando estes estão desequilibrados, a ortogonalidade é perdida e os componentes se tornam correlacionados.

Desequilíbrio de fase

Em um demodulador IQ ideal, a diferença de fase entre os sinais usados para demodular o ramo I e o ramo Q é de 90 graus, resultando em $cos(2\pi f_c t)$ e $sin(2\pi f_c t)$ sendo usados para para demodular $y_I(t)$ e $y_Q(t)$, respectivamente. Quando há desequilíbrio de fase, a diferença de fase pode não ser exatamente de 90 graus. Podemos considerar que $cos(2\pi f_c t)$ é utilizado para demodular $y'_I(t)$ e $sin(2\pi f_c t + \beta)$ para demodular $y'_Q(t)$.

Desequilíbrio de amplitude

Quando há desequilíbrio de amplitude, há uma variação na amplitude de um dos ramos de seno ou de cosseno do demodulador. Isso pode ser modelado como usando $cos(2\pi f_c t)$para demodular $y'_I(t)$ e $\alpha sin(2\pi f_c t)$ para demodular $y'_Q(t)$.

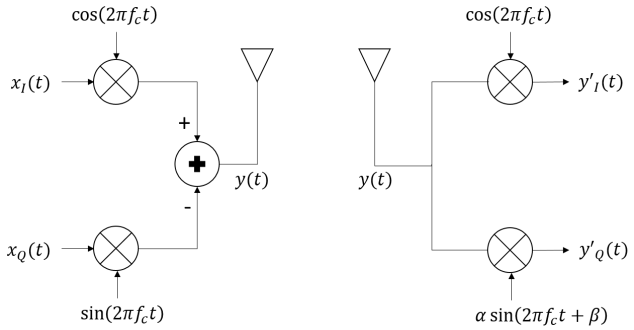
O sinal recebido, incluindo os efeitos do desequilíbrio de fase e amplitude é dado por

$$y'_I(t) = y(t)cos(2\pi f_c t),$$

$$y'_Q(t) = \alpha\, y(t) sin(2\pi f_c t + \beta).$$

Esses desequilíbrios podem levar a artefatos na saída do demodulador, resultando em distorção do sinal recebido. Como vimos acima, um sinal IQ em desequilíbrio foi convertido para banda base com duas ondas senoidais de amplitudes desiguais e/ou diferenças de fase diferentes de 90 graus.

A modulação e demodulação IQ com desequilíbrio de amplitude e fase é mostrada na figura abaixo.



Aplicando algumas identidades trigonométricas a $y'_Q(t)$, temos

$$y'_Q(t) = \alpha\, y(t) sin(2\pi f_c t) cos(\beta) + \alpha\, y(t) cos(2\pi f_c t) sin(\beta).$$

Observe que o componente em quadratura desbalanceado, $y'_Q(t)$, neste caso é igual a uma versão em escala do componente de quadratura equilibrada $y_Q(t)$ mais uma versão em escala do componente em fase $y_I(t)$. Em outras palavras,

$$y'_Q(t) = \alpha cos(\beta)\, y_Q(t) + \alpha sin(\beta) y_I(t)$$

Como exemplo da diferença para o caso ideal (i.e., sem desbalanceamento), considere o que aconteceria se filtrássemos os componentes de alta frequência resultantes em ambos os ramos I e Q do demodulador, obtemos uma diferença de amplitude para o caso desequilibrado. Essa diferença depende da razão de diferença de amplitude (i.e., α) e da diferença de fase em quadratura (i.e.,β) das duas ondas senoidais. Corrigir o desequilíbrio de IQ significa realizar uma operação digital (uma combinação linear dos ramos I e Q) para restaurar a ortogonalidade dos sinais I e Q. Isso geralmente é feito no domínio digital após uma caracterização das imperfeições I e Q ter sido feita.

Observe que se expressarmos as formas balanceadas e desbalanceadas em notação matricial, chegamos à relação

$$\begin{bmatrix} y'_I(t) \\ y'_Q(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \alpha \sin(\beta) & \alpha \cos(\beta) \end{bmatrix} \begin{bmatrix} y_I(t) \\ y_Q(t) \end{bmatrix},$$

o que implica em um método para corrigir o desbalanceamento IQ. Para isso, simplesmente multiplicamos o vetor de sinais não balanceados pelo inverso da matriz (a inversa existe se $\alpha \neq 0$ e β não for um múltiplo ímpar de $\pi/2$ radianos).

Porém, uma desvantagem potencial desse processamento é que o ruído, se presente, pode ser amplificado no ramo Q. Para valores típicos de α próximos da unidade e β próximos de zero, essa amplificação de ruído é desprezível.

OBS.: Neste exercício, nós consideramos que o modulador é ideal (i.e., sem desbalanceamento IQ), mas poderíamos fazer a mesma análise considerando um modulador com desequilíbrio de fase e amplitude e um demodulador ideal.

Exercício: após esse overview sobre desbalanceamento IQ, usando o notebook IQImbalanceExercise.ipynb (faça o download do notebook), faça o seguinte:

1. Execute célula por célula do notebook entendendo o que foi feito para gerar os sinais que são modulados e demodulados. Veja que é feito a correção do desbalanceamento IQ através da inversão da matriz mostrada acima.
2. Agora, você deve realizar a mesma correção, porém encontrando os parâmetros de um **modelo de regressão linear** para correção do desbalanceamento IQ através do **algoritmo do gradiente descendente em batelada**.
   (**Dica**: o modelo de regressão linear é encontrando analisando-se a equação em forma matricial acima. Perceba que apenas um dos ramos é afetado pelo desbalanceamento IQ).
   (**Dica**: use a variável ***y_imb***, i.e., sinal recebido com desbalanceamento para criar a matriz de atributos ***X*** e a variável ***y_per***, i.e., sinal perfeito, sem nenhum desbalanceamento, para criar os rótulos de treinamento, ou seja, a saída esperada.)
   (**Dica**: Não se esqueça de encontrar o valor do passo de aprendizagem ideal para o treinamento. No caso deste exercício, ele pode estar em torno de 1000.)
3. Após o treinamento do **modelo de regressão linear**, plote gráfico de (i) épocas versus erro, (ii) a superfície de erro, e a (iii) superfície de contorno com as variações dos parâmetros a cada época, mostre que os parâmetros caminham em direção ao mínimo global, o qual deve ser encontrado através da **equação normal**.
4. De posse dos parâmetros ótimos, faça a correção do desbalanceamento IQ e plote a constelação do sinal recebido e que teve seu desbalanceamento IQ corrigido e o compare com a constelação do sinal IQ perfeitamente recebido, i.e., o sinal que não sofreu desbalanceamento IQ.
5. Se você fizer a padronização dos atributos, qual será o valor do passo de aprendizagem ideal?
6. De posse dos parâmetros ótimos obtidos após a padronização dos atributos e treinamento do modelo, faça a correção do desbalanceamento IQ e plote a constelação do sinal recebido e que teve seu desbalanceamento IQ corrigido e o compare com a constelação do sinal IQ perfeitamente recebido, i.e., o sinal que não sofreu desbalanceamento IQ.
   (**Dica**: Os pesos obtidos pelo gradiente descendente com os atributos padronizados precisam ser transformados, pois a padronização altera a localização do ponto de mínimo global.)

Referências

[1] https://dsp.stackexchange.com/questions/40734/what-does-correcting-iq-do

[2] http://www.dsplog.com/2009/03/08/iq-imbalance-in-transmitter/

[3] https://www.sciencedirect.com/topics/engineering/amplitude-imbalance

[4] http://www.dsplog.com/2013/01/31/imrr-transmit-iq-gain-phase-imbalance/

14. Usando o código da implementação do gradiente descendente em mini-batches desenvolvida no exercício 5 desta lista, faça o seguinte:
    a. Acrescente ao código do mini-batch o esquema de ajuste do passo de aprendizagem conhecido como decaimento exponencial, o qual possui a seguinte equação de ajuste do passo: alpha_init * exp(-*k*t*), onde alpha_init é o passo de aprendizagem inicial, k é a taxa de variação da exponencial e *t* é o número da iteração. (**OBS**.: Lembre-se que k, assim como alpha, é um hiperparâmetro que precisa ser ajustado).
    b. Gere a seguinte função observável: y_noisy = 2*x1 + 3*x2 + 4*x3 + w, onde x1, x2, x3 e w são vetores coluna com N = 1000 amostras retiradas da distribuição Gaussiana Normal Padrão (i.e., média zero e variância unitária).
    c. Encontre os valores ótimos com a equação normal.
    d. Usando ***Grid Search***, encontre um par de valores alpha_init e k que resultem no menor erro possível com o conjunto de treinamento gerado. (**Dica**: se baseie no exemplo: linear_regression_grid_search.ipynb)
        i. Usem na primeira rodada do Grid Search, os seguintes valores:
            alpha_init = [0.0001, 0.001, 0.01, 0.1]
            k = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]
        ii. Gere uma figura do erro em função de alpha_init, plotando os valores do erro para cada valor de k.
        iii. Na próxima rodada do Grid Search, de posse de um intervalo onde alpha_init e k produzem o menor valor de erro entre os vários valores considerados, refine os valores de alpha_init e k. Repita essas rodadas de refinamento quantas vezes você achar necessário para encontrar bons valores para alpha_init e k.
        iv. Usando os melhores valores encontrados para alpha_init e k, imprima:
            1. O erro obtido com o modelo treinado através do mini-batch e o erro obtido com a equação normal.
            2. Os valores dos pesos obtidos com a equação normal e com o mini-batch.
        v. Plote um gráfico mostrando a evolução do valor do passo de aprendizagem e outro com os elementos do vetor gradiente em função das iterações.
15. Usando o exemplo: "**stocastic_gradient_descent_with_adam.ipynb**" como base para a resolução deste exercício, faça o seguinte:

a. Implemente 2 novos métodos de atualização dos pesos: AdaGrad e RMSProp, respectivamente.
b. Crie vetores para manter o histórico de atualizações da variável ***v*** e do valor de ***alpha*** resultante (i.e., o valor de alpha inicial dividido pela raiz quadrada de ***v*** + ***epsilon***).
c. Usando ***alpha*** igual a 0.02, número de épocas igual a 1 e **GDE puro**, treine o modelo e plote
    i. Superfície de contorno com histórico de atualização dos pesos e ponto de mínimo obtido com a equação normal.
    ii. Curva do erro em função das iterações.
    iii. Gráfico com o histórico das atualizações.
d. Usando ***alpha inicial*** igual a 0.02, número de épocas igual a 1, **epsilon** igual a 1e-8 e GDE com **AdaGrad**, treine o modelo e plote
    i. Superfície de contorno com histórico de atualização dos pesos e ponto de mínimo obtido com a equação normal.
    ii. Curva do erro em função das iterações.
    iii. Gráfico com o histórico de atualizações de **alpha para cada um dos pesos.**
    iv. Gráfico com o histórico de atualizações da variável **v para cada um dos pesos.**
    v. Gráfico com o histórico das atualizações.
e. Usando ***alpha inicial*** igual a 0.02, número de épocas igual a 1,beta igual a 0.99999, **epsilon** igual a 1e-8 e GDE com **RMSProp**, treine o modelo e plote
    i. Superfície de contorno com histórico de atualização dos pesos e ponto de mínimo obtido com a equação normal.
    ii. Curva do erro em função das iterações.
    iii. Gráfico com o histórico de atualizações de **alpha para cada um dos pesos.**
    iv. Gráfico com o histórico de atualizações da variável **v para cada um dos pesos.**
    v. Gráfico com o histórico das atualizações.
f. Compare os resultados obtidos e relate as diferenças que você percebeu com relação
    i. A convergência das versões do gradiente descendente. (i) Todas as versões convergem? (ii) Elas se estabilizam no ponto de mínimo global? (iii) O caminho tomado pelo algoritmo vai diretamente até o ponto de mínimo?.
    ii. Ao tempo de convergência. Quantas iterações são necessárias para que eles se aproximem do ponto de mínimo.
    iii. A variação de ***alpha*** e da variável ***v***, para AdaGrad e RMSProp. (i) Como a variação dessas variáveis afeta o treinamento das duas versões? (ii) Elas se aproximam do ponto de mínimo? Caso não, qual é o motivo disso?

iv. A variação das atualizações para o GDE puro e o GDE com RMSProp. O que você percebe com relação a variação das atualizações para cada um dos pesos? Elas são semelhantes para as 2 versões? Caso não, qual é o motivo dessa diferença nas atualizações?

16. Nesta atividade, vamos abordar uma instância do problema de regressão de grande interesse prático e com uma extensa literatura: a **predição de séries temporais**. A fim de se **prever** o valor futuro de uma série de medidas de uma determinada grandeza, um procedimento típico consiste em construir um modelo matemático de estimação baseado na hipótese de que os valores passados da própria série podem explicar o seu comportamento futuro.

Seja *x(n)* o valor da série temporal no instante (discreto) *n*. Então, o modelo construído deve realizar um mapeamento do vetor de entradas $\boldsymbol{x}(n) \in \Re^{K\times 1}$, o qual é formado por um subconjunto de K amostras passadas, i.e.,

$$\boldsymbol{x}(n) = [x(n\text{-}1), x(n\text{-}2), \ldots, x(n\text{-}K)]^T,$$

para uma saída $\widehat{y}(n)$, que representa uma estimativa do valor futuro da série *x(n)**.

* Esta modelagem está pressupondo o caso em que desejamos prever o valor da série um passo à frente.

Neste exercício, vamos trabalhar com a série de temperatura mínima diária referente à cidade de Melbourne, Austrália, no período de 1981 a 1990. As observações estão em graus Celsius e há 3650 amostras no total. Os dados são creditados ao *Australian Bureau of Meteorology*.

Inicialmente, vamos explorar um modelo linear para a previsão, tal que:

$$\widehat{y}(n) = w^T \boldsymbol{x}(n) + w_0.$$

Para o projeto do preditor linear, separe os dados disponíveis em dois conjuntos - um para treinamento e outro para validação. No caso, reserve as amostras referentes ao ultimo ano (1990) em seu conjunto de validação. Além disso, utilize um esquema de validação cruzada do tipo k-fold para selecionar o melhor valor do hiperparâmetro K.

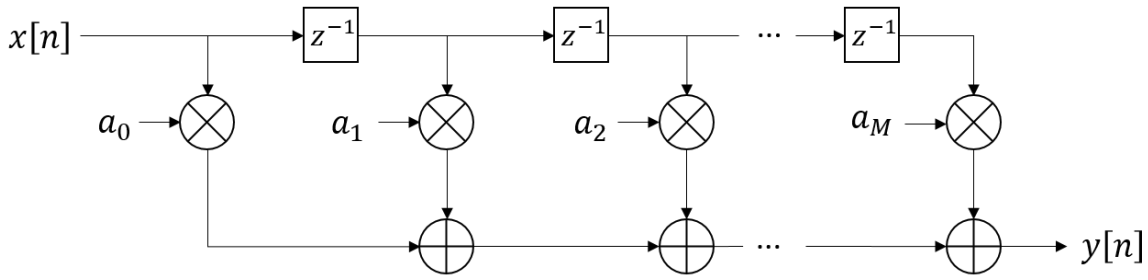
Faça a análise de desempenho do preditor linear ótimo, no sentido de quadrados mínimos irrestrito, considerando:

1. A progressão do valor médio do erro quadrático médio (MSE, do inglês mean squared error), junto aos dados de teste, em função do número de entradas, K, do preditor (desde K=1 a K=30).
2. O gráfico com as amostras de validação da série temporal e com as respectivas estimativas geradas pela melhor versão do preditor (i.e., usando o valor de K que levou ao mínimo erro de validação).
3. Plote as primeiras 100 amostras do conjunto de validação e compare com as primeiras 100 amostras preditas pelo melhor modelo.
4. Analise o gráfico acima e diga o que você consegue perceber através desta comparação. Para te ajudar a entender, plote a correlação cruzada entre o conjunto de validação e os valores preditos. Onde ocorre o pico da correlação? O que isso significa? Leia a referência abaixo.

Observação: Neste exercício, não é necessário utilizar regularização, nem efetuar normalizações nos dados.

Referência:
[1]
https://towardsdatascience.com/how-not-to-use-machine-learning-for-time-series-forecasting-avoiding-the-pitfalls-19f9d7adf424

17. Um filtro FIR (do inglês, Finite Impulse Response) de ordem M, tem a seguinte estrutura



Essa estrutura pode ser representada matematicamente através da seguinte equação

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + \cdots + a_M x[n-M] = \sum_{k=0}^{M} a_k x[n-k],$$

onde M é a ordem do filtro, x é o sinal de entrada do filtro, y é a saída do filtro, ou seja, o sinal filtrado e a_k, $\forall k$ são os coeficientes do filtro.

Perceba que esta é uma equação linear, onde temos uma combinação linear dos atributos, neste caso versões atrasadas do sinal de entrada $x[n]$, em relação aos pesos, que são os coeficientes do filtro.

Se a ordem do filtro, ou seja, o parâmetro M, já estiver definido previamente, só é necessário encontrar seus coeficientes.

Assim, podemos aplicar algoritmos de regressão linear para encontrar esses coeficientes.

Agora, vamos supor que temos um filtro operando em um sistema de telecomunicações e que queremos replicar este filtro para uso em outro sistema. Porém, nós não conhecemos o projeto do filtro. Ou seja, seu tipo (e.g., passa-baixas, passa-altas, etc.), sua ordem, frequência de corte, etc.

Como podemos projetar um filtro que se aproxime das características do filtro desejado?

Uma possível abordagem é injetar ruído branco Gaussiano (i.e., média igual a zero e variância unitária) em sua entrada e usar as entradas e a saídas resultantes para treinar um modelo de regressão linear. Portanto, faça o seguinte:

- Gere 1000 amostras retiradas de uma distribuição Gaussiana normal padrão.

- Passe essas amostras através do filtro FIR dado no notebook de exemplo
    - Estude o código presente no notebook de exemplo para entender como fazer isso.
    - A ordem do filtro FIR do notebook de exemplo é igual a 45.
- Use uma função hipótese com o formato da equação que representa o filtro FIR (equação acima), note que temos um modelo linear.
    - A função ***createDataset*** cria uma matriz de atributos de entrada de acordo com a equação que representa o filtro FIR, ou seja, gera os atributos da função hipótese desejada. Portanto, use esta função para gerar a matriz que será usada durante o treinamento do modelo de regressão linear.
    - Cada linha da matriz de atributos retornada pela função ***createDataset*** corresponde a **um elemento** da array de saída do filtro FIR, ou seja, o valor esperado (rótulo) para a n-ésima linha da matriz de atributos corresponde ao n-ésimo elemento do vetor de saída do filtro.
    - Portanto, temos uma matriz de atributos com dimensões $N \times M+1$ e um vetor de rótulos (i.e., valores esperados) com dimensões $N \times 1$ que vem ser usados para o treinamento do modelo de regressão linear.
- Use um objeto da classe ***SGDRegressor*** para treinar o modelo de regressão linear.
- Após o treinamento do modelo de regressão linear, faça:
    - Calcule e imprima o valor do erro quadrático médio (use todo o conjunto de dados).
    - Imprima e compare os pesos do modelo com os coeficientes do filtro FIR. Eles são próximos?
    - Passe o sinal de teste (sinal com a soma de dois sinais cossenoidais gerados no notebook de exemplo) através do modelo de regressão linear (use a função ***createDataset*** para converter o sinal de teste em uma matriz e o método ***predict*** do modelo de regressão linear para filtrar o sinal de teste).
    - Plote a FFT do sinal de saída do modelo de regressão linear.
    - Compare a FFT do sinal de saída do modelo de regressão linear com a FFT do sinal de saída do filtro FIR. Os resultados são similares?

Referências

[1] 'FIR filter', https://scipy-cookbook.readthedocs.io/items/FIRFilter.html

[2] 'Filters', https://pysdr.org/content/filters.html