Quasi - Newton

Pedro Loes e Felipe Sadock

7/28/2021

O método IWLS ou Mínimos Quadrados Ponderados Interativos consiste em uma técnica de otimização baseada na função Escore de Fisher para estimar os coeficientes de distribuições da família exponencial utilizando as técnicas de optimização da família de algoritmos Quasi-Newton. A derivada da função logaritmo da verossimilhança é utilizada para iterativamente estimar os betas que ajustam a curva de dados provenientes desta família de distribuições. Este projeto consistiu na apresentação do funcionamento do método seguido pela simulação do seu comportamento e performance em diferentes cenários, para verificar o custo computacional representado pelo número de iterações e precisão representada pela Deviance dos ajustes. O exemplo das distribuição Poisson foi utilizado para investigar o método com as simulações. Um aplicativo foi desenvolvido para ilustrar o funcionamento do algoritmo com gráficos animados para acompanhamento dos passos do algoritmo em direção a convergência dos betas reais.

1. Introdução

- Este projeto abrangeu uma explicação sobre o funcionamento geral do método IWLS por meio de uma visão geral, mecânica matemática , passo a passo do algoritmo e exemplo da distribuição Poisson.
- Simulações foram desenhadas para explorar os parâmetros do algoritmo no que diz respeito aos betas iniciais, tamanho amostral, tolerância e deviance com intuito de verificar a performance do algoritmo em diferentes cenários e facilitar a tunagem dos parâmetros.
- Finalmente um aplicativo foi construído com diversos botão para tunagem iterativa da optimização apresentando resultados por meio de gráficos animados no formato de gifs, bem como estimativas pontuais de ajuste e adequação dos modelos.
- Etapas do projeto:

    1. Introdução.
    2. Método IWLS.
    3. Simulações.
    4. Aplicativo.

2. Método IWLS

- Os modelo Lineares Generalizados da família exponencial não possuem forma analítica fechada para o Estimador de Máxima Verossimilhança dos Betas como os modelos onde a suposição de normalidade da variável resposta é satisfeita.
- A Estimativa dos coeficientes que capturam a contribuição de covariáveis para a respostas deve ser feito por meio de métodos de optimização. O método IWLS baseado na função Escore de Fisher utiliza a mesma mecânica de derivadas da família Quasi Newton de algoritmos para optimização.

Algoritmo

0. Define um r = 0 para contagem das iterações.
1. Determinar chute inicial para os betas.
2. Calcular o preditor linear.
3. Calcular a média.
4. Calcular a variância.
5. Calcular a matriz de pesos.
6. Calcular o vetor de derivadas.
7. Calcular os novos betas.
8. Verificar se a tolerância mínima de erro para os betas é satisfeita.
9. Caso afirmativo o algoritmo para, caso contrário o algoritmo retorna para etapa 2.

Exemplo

- A distribuição Poisson foi utilizada para ilustrar o funcionamento do algoritmo considerando apenas uma covariável e portanto dois betas estimados.
- Característica da distribuição:
  - Preditor linear $\eta_i = \beta_0 + \beta_1 * X_{1i}$.
  - $Y_i \sim Pois(\theta_i)$ com $\mu_i = \theta_i > 0$
  - $E[Y_i] = Var[Y_i] = \mu_i = \theta_i$.
  - Parâmetro Canônico: $b(\theta_i) = ln(\theta_i)$
  - Função de Ligação: $\mu_i = \theta_i = g(\eta_i) = e^{\eta_i}$.
  - $\frac{d\mu_i}{d\eta_i} = e^{\eta_i}$.
  - $\frac{d\eta_i}{d\mu_i} = \frac{1}{\mu_i}$
- Passo a passo:
  0. $r = 0$.
  1. Qualquer valor arbitrário pode ser atribuído para os betas iniciais.
  2. $\eta_i = \beta_0 + \beta_1 * X_{1i}$.
  3. $E[Y_i] = \theta_i$.
  4. $Var[Y_i] = \theta_i$.
  5. $W = \frac{1}{Var[Y_i]}(\frac{d\mu_i}{d\eta_i})^2$.
  6. $z_i = \eta_i + (y_i - \mu_i)\frac{d\eta_i}{d\mu_i}$.
  7. $\beta^{(r+1)} = (X^T\ W^{(r)}\ X)^{-1}\ X^T\ W(r)\ z^{(r)}$.
  8. $|\beta^{(r+1)} - \beta^{(r)}| \geq$ valor de tolerância arbitrário.

Implementação IWLS Python

- A implementação da função IWLS que otimiza os parâmetros betas possui 6 parâmetros de entrada e retorna um data frame dos betas estimados:
  1. **Y** = Vetor valores observados para serem ajustados pelo modelo GLM.
  2. **Xs** = Matriz de Covariáveis que explicam a resposta X incluindo o intercepto.
  3. **beta0** = Vetor de betas iniciais.
  4. **tol** = Tolerância para parada do algoritmo.
  5. **norma** = Valor arbitrário inicial para norma.
  6. **dist** = Distribuição para a qual o algoritmo realiza a otimização.
- Após a declaração das variáveis betas, norma e r um laço do tipo while é utilizado para realizar as iterações do algoritmo IWLS. O critério de condição norma $<$ tol é utilizado para parada do algoritmo quando a distância euclidiana entre os vetores de betas reais e betas estimados é satisfeita.

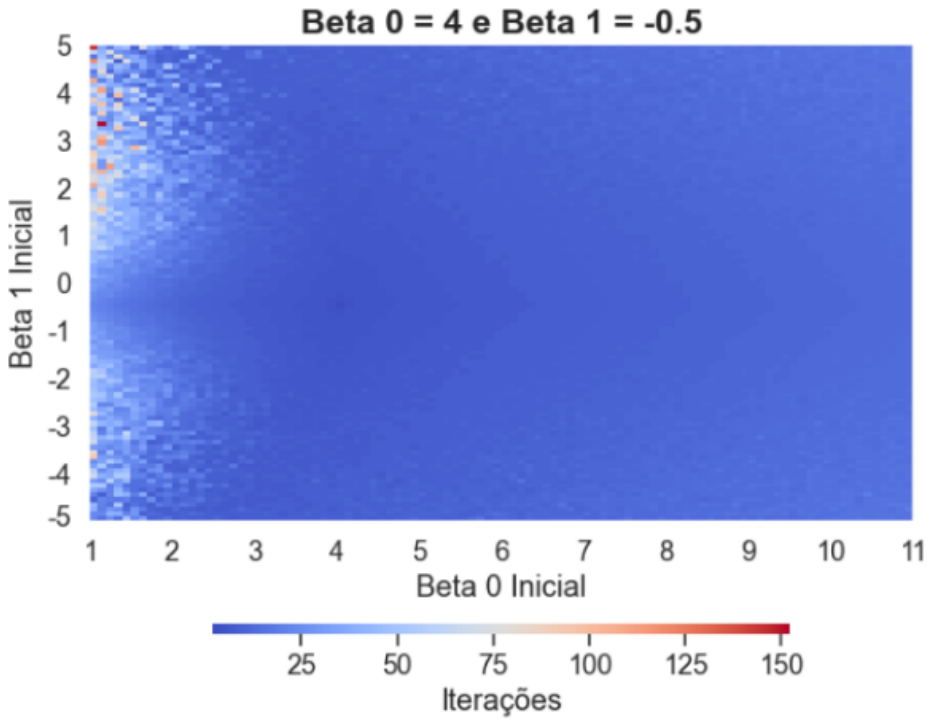
```
# Declara função algoritimo IWLS
def IWLS(Y, Xs, beta0, tol, norma, dist):
  # Declara betas
  betas = pd.DataFrame({"it" : [1],"beta0" : beta0[0],"beta1" : beta0[1]})
  # Declara norma
  norma = norma
  # Declara iteracao
  r = 0
  # Laço de iterações
  while norma > tol:
    # Declara beta da vez
    beta = np.array(betas.iloc[r,[1,2]])
    # Calcula eta0
    eta = mat(Xs, beta)
    # Calcula media inicial
    theta = calcula_theta(eta, dist)
    # Calcula diagonal da matriz Wi
    W_i = calcula_W(eta, theta)
    # Declara matriz W
    W = np.eye(Y.shape[0])
    np.fill_diagonal(W, W_i)
    # Calcula z
    z_i = calcula_Z(eta, Y)
    # Calcula beta_0
    b0 = mat( mat( mat( inv( mat( mat( t(Xs), W) , Xs) ), t(Xs) ), W), z_i)[0]
    # Calcula beta_1
    b1 = mat( mat( mat( inv( mat( mat( t(Xs), W) , Xs) ), t(Xs) ), W), z_i)[1]
    # Gera data frame de novos betas
    df_temp = pd.DataFrame({"it" : [r+1],"beta0" : [b0],"beta1" : [b1]})
    # Incrementa data frame
    betas = betas.append(df_temp, ignore_index=True)
    # Calcula norma euclideana
    norma = np.sqrt(sum((abs(betas.iloc[r+1, [1,2]] - betas.iloc[r, [1,2]]))**2))
    # Incrementa r
    r = r + 1
  # Retorno da função
  return betas
```

3. Simulações

- As simulações foram desenhadas para 4 parâmetros do algoritmo. Cada simulação considerou o espaço de busca de 100 valores para cada parâmetro e verificou os resultados obtidos na terceira dimensão (cor na escala de gradiente de vermelho a azul) em gráficos do tipo mapa de calor.
- Parâmetros reais foram arbitrariamente definidos para construção da variável resposta Y com base em uma variável $X \sim Normal(0, 0.5)$ também arbitrariamente definida.
- As simulações foram executadas utilizando 2 laços do tipo **for** do pacote base da linguagem Python. Os gráficos foram construídos utilizando os pacotes *matplotlib* e *seaborn* da linguagem Python.

Iterações Betas Iniciais Poisson

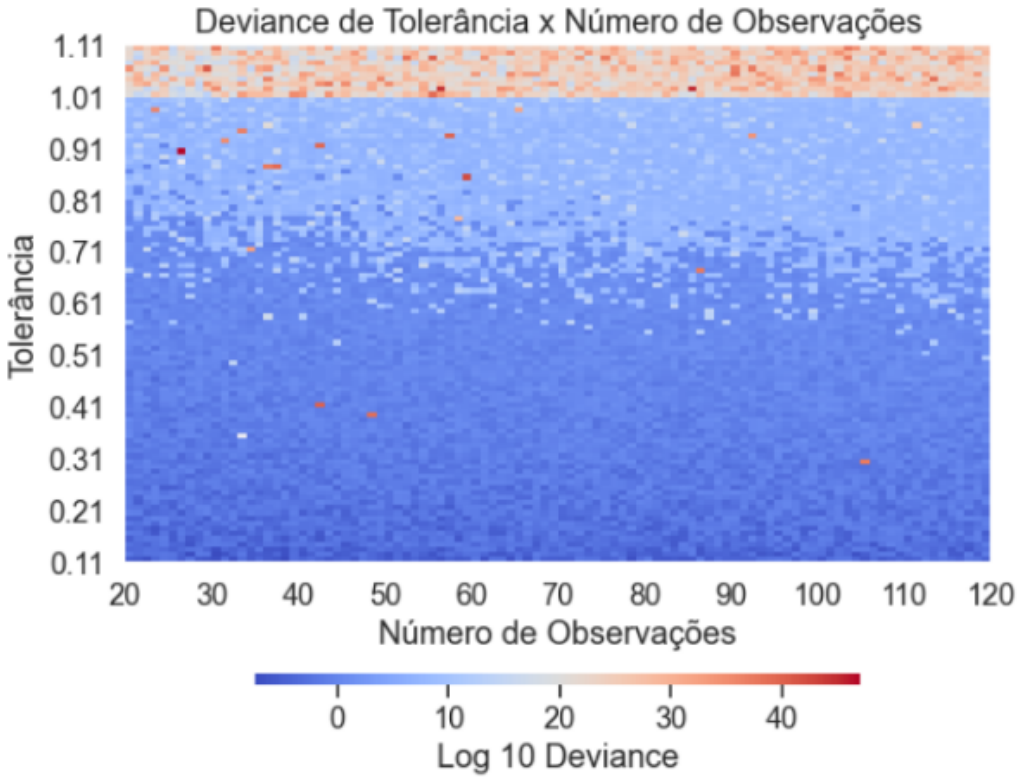
- A primeira simulação foi desenhada para investigar o número de iterações gastas pelo algoritmo IWLS para optimizar os parâmetros reais $\beta_0 = 4$ e $\beta_1 = -0.5$ da distribuição $Poisson(\theta_i)$. Nos eixos β_0 Inicial e β_1 Inicial do gráfico foram considerados 100 β_0 iniciais x 100 β_1 iniciais.



- A simulação evidenciou que a escolha inicial do parâmetro β_1 inicial próximo do β_1 real apresenta custo computacional entre 0 e 10 iterações e entre 100 e 150 iterações para β_1 inicial distante do valor real considerando β_0 com valores entre 1 e 3.
- Para valores de $beta_0$ inicial com valores superiores a 3 a simulação apresentou custo computacional inferior a 25 iterações para qualquer combinação de parâmetros.
- Pode-se observar que as simulações de parâmetros β_0 inicial e β_1 inicial próximos dos valores β_0 e β_1 real apresentaram os melhores resultados com destaque para o número de iterações 5 quando os parâmetros iniciais eram exatamente equivalentes aos parâmetros reais.

Iterações Tamanho de Amostras e Tolerância Poisson

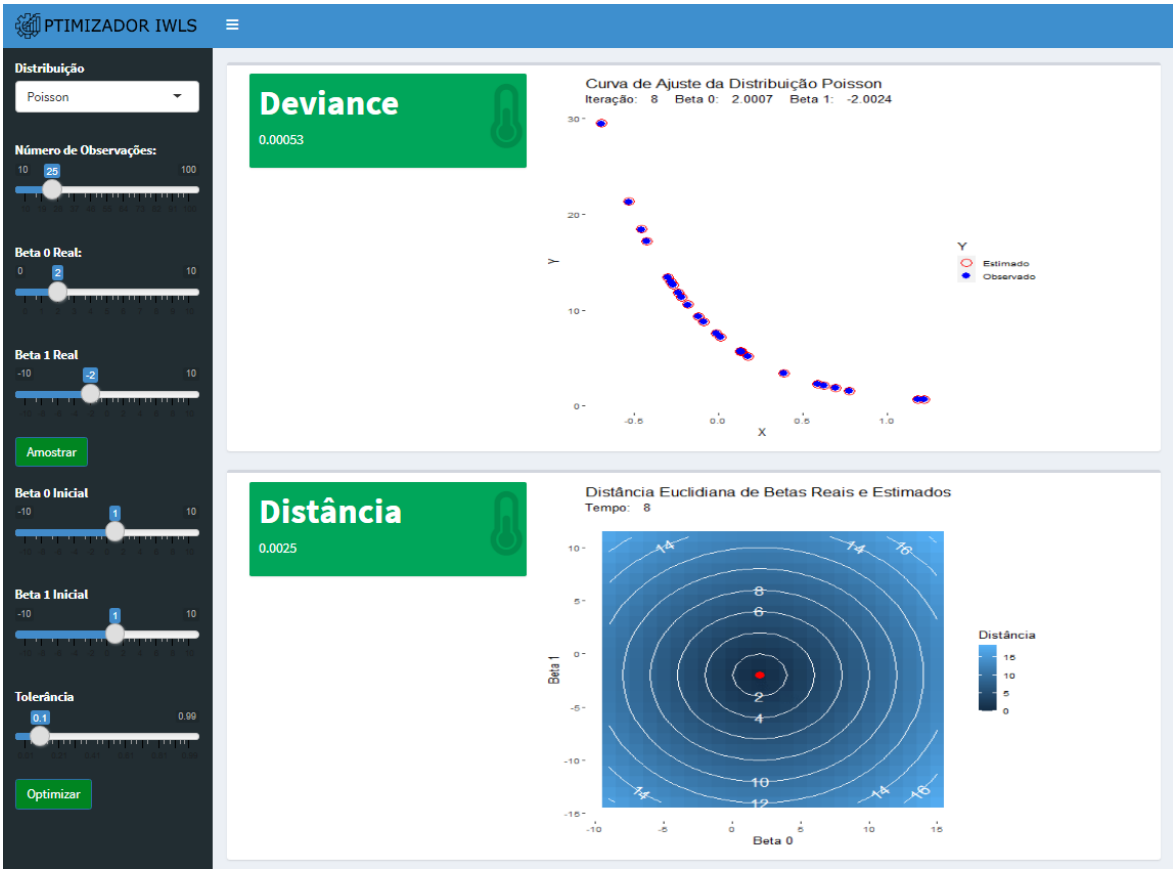
- A segunda simulação foi desenhada para investigar a precisão mensurada pela Deviance da curva ajustada com os betas otimizados pelo algoritmo IWLS considerando a variação da tolerância e do número de observações da amostra da distribuição $Poisson(\theta_i)$.
- Nos eixos X e Y do gráfico foram considerados 100 valores de tolerâncias para critério de parada do algoritmo e tamanho amostral da curva real.
- A escala da deviance foi transformada para escala logarítmica porque mesmo em um intervalo pequeno de tolerância o algoritmo apresentou Deviances próximas de zero até 10^{40}.



- Foi possível identificar 4 regiões distintas de Deviances ao longo do eixo Tolerância.
  - A primeira, com tolerâncias entre 0.11 e 0.21 observou-se na maioria dos casos Log 10 da deviance menor que 0, ou seja, Deviances inferiores a 1.
  - A segunda, com tolerâncias entre 0.21 e 0.61 observou-se na maioria dos casos Log 10 da Deviance entre 0 e 10, ou seja Deviances entre 1 e 10^{10}.
  - A terceira, com tolerâncias entre 0.61 e 1.01 observou-se na maioria dos casos Log 10 da Deviance entre 10 e 20, ou seja, Deviances entre 10^{10} e 10^{20}.
  - Na quarta, com tolerância entre 1.01 e 1.11 observou-se na maioria dos casos Log 10 da Deviance entre 20 e 40, ou seja, Deviances entre 10^{20} e 10^{40}.
- A análise do tamanho amostral indicou uma tendência de queda da **Tolerância** ao longo do eixo **Número de observações**. Portanto foi possível observar que para tamanhos amostrais maiores, é necessário uma tolerância menor para verificar os mesmos resultados de Deviance.
- Finalmente é possível observar que para todas as tolerâncias maiores que 1 o valor das Deviances explode.

Aplicativo Optimizador IWLS

- O aplicativo interativo e animado Optimizador IWLS foi construído com o pacote shiny da linguagem R para ilustrar o passo a passo do algoritmo. O objetivo da construção deste aplicativo foi investigar a mecânica de funcionamento do algoritmo e possibilitar a exploração dos parâmetros com intuito de analisar qualidade e defeitos do algoritmo.



- Link do Aplicativo:
  - Optimizador IWLS

Desenho

- O aplicativo foi projetado com uma barra na lateral esquerda que contém os botões para ajuste fino dos parâmetros do algoritmo e inicialização da amostragem e optimização.
- Do lado direito foi inserida a janela do painel principal que contém:
  - Na caixa superior o gráfico animado de ajuste da curva em cada iteração do algoritmo juntamente com a caixa de valor da Deviance final do algoritmo. O gráfico exibe os valores ajustados pelos betas em cada iteração na cor vermelha no formato de pontos vazados e os valores observados na cor azul no formato de pontos sólidos menores.
  - A caixa inferior da janela painel principal contém o gráfico animado da distância euclidiana dos vetores de betas reais e estimados em cada iteração do algoritmo e a caixa de valor com a distância euclidiana final dos vetores de betas reais e estimados. O gráfico ilustra também as curvas de nível da distância dos vetores de betas e utiliza o gradiente das cores pretas até azul para ilustrar o vale com valor mínimo zero onde os betas reais e estimados são exatamente iguais.

- Dois botões foram inseridos para permitir que o usuário altere somente parâmetros da optimização e acione **Optimizar** utilizado a mesma amostra já selecionada anteriormente.

Funcionalidade

- Inicialmente o aplicativo está no modo de espera não apresentando nenhum resultado porque os botões **Amostrar** e **Optimizar** não foram acionados. O acionamento do botão **Amostrar** seguido do botão **Optimizar** inicializam a optimização com os parâmetros padrões do aplicativo.
- No que diz respeito a amostragem são oferecidos ao usuário os serviços de controle do tamanho da amostra e betas reais. Já para optimização são oferecidos os serviços de tunagem dos betas iniciais e tolerância do IWLS.
- A construção dos gráficos animados no formato gif gasta em torno de 20 a 30 segundos dependendo dos parâmetros escolhidos.
- As caixas de valor da Deviance e da Distância Euclidiana dos vetores de betas mudam da cor verde para vermelho se a distância ou a Deviance são muito grandes.

Referências e Links

- GLM Wikipedia
- Aplicativo Optimizador IWLS
- Projeto GitHub