GESTÃO DE TRAFEGO URBANO UTILIZANDO PROCESSAMENTO DE IMAGEM EM TEMPO REAL E MODELOS DE PREDIÇÃO

João Pedro Romero Soares [1]; Daniel de Oliveira Mota [2]

[1] Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
[2] Professor da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Resumo. *A busca para melhorar a qualidade de vida em megacidades tem sido um desafio para pesquisadores da área de logística devido a alta quantidade de pessoas que habitam esses lugares. O alto número de pessoas em lugares pequenos gera o efeito denominado "caos urbano" que nada mais é do que um alto nível de congestionamento de carros em um horário de fluxo. Buscando lidar com este problema, por meio de modelos matemáticos alimentados por algoritmos de visão computacional, propõe-se construir uma plataforma móvel para a gestão de congestionamento (análise, predição e ação) que possa tornar perceptível a melhoria do fluxo de veículos e redução no tempo de atravessamento de uma região congestionada. Esta pesquisa buscou analisar e estudar a fórmula de Bayes a fim de conhecer o seu funcionamento e adaptá-la para prever o tempo que o semáforo deve ter para diminuir o trânsito naquele instante baseado em um histórico da localidade, considerando as variáveis: quantidade de veículos e tempo de atravessamento. A metodologia utilizada nesta iniciação científica foi à pesquisa exploratória e estudo de caso. Para analisar as fórmulas estatísticas foi utilizada a plataforma de programação Python, pois é uma plataforma simples de ser utilizada e com bastantes recursos na internet pelo fato de ser uma plataforma de fonte aberta (open source). Com o desenvolvimento da pesquisa, pôde ser observado que o código Bayes só era feito para classificar objetos baseado em dados qualitativos e não quantitativo o que fez com que ele fosse descartado. Buscando uma nova fórmula estática, foi encontrado o Algoritmo de Gibbs, também chamado de Amostragem de Gibbs, que é um algoritmo utilizado para gerar uma sequência de amostras da distribuição conjunta de probabilidades de duas ou mais variáveis aleatórias com o objetivo de aproximar a distribuição conjunta, gerando um valor esperado. O Algoritmo de Gibbs é um caso especial do algoritmo de Metropolis-Hastings e apesar de ser muito usado pela área de medicina veterinária para prever genes, pouco foi encontrado sobre este algoritmo na área de pesquisa de logística. Utilizando um banco de dados para teste foi constatado que o Algoritmo de Gibbs é sim capaz de prever uma variável,*

Introdução

Veículos autônomos, cidades inteligentes, aprendizado de máquinas são temas que tem se tornado recorrentes não só em assuntos acadêmicos, mas da população em geral. Trazendo estes tópicos ao contexto urbano, sabe-se que a mobilidade urbana é um dos grandes problemas discutidos no século XXI. Isso se dá, principalmente devido ao grande impacto na vida dos residentes de uma cidade com alto adensamento populacional, crescimento desordenado, precária política de transporte público, e intenso gerado de empregos. As cidades que tenham a maioria, senão todas, as características mencionadas são atualmente classificadas como "megacities". O problema abordado nesta proposta possui uma característica transitória, uma vez que existe uma lacuna fundamental entre um sistema de transporte totalmente autônomo e o sistema observado na atualidade (em todo o mundo). E a expectativa desta equipe é construir esta ponte através de uma maior coordenação entre os elementos que compõe o sistema de transporte urbano em uma megacidade.

Com este contexto em mente, o problema da falta de coordenação entre os sinais de transito em regiões urbanas de elevado índice de congestionamento será desafiado através deste projeto.

Material e Métodos

A metodologia utilizada nesta iniciação científica foi à pesquisa exploratória e estudo de caso. Para analisar as fórmulas estatísticas foi utilizada a plataforma de programação Python, pois é uma plataforma simples de ser utilizada e com bastantes recursos na internet pelo fato de ser uma plataforma de fonte aberta (open source) e foi utilizado o *Microsoft Office Excel* como banco de dados.

Os conceitos necessários sobre Bayes e o Algoritmo de Gibbs foram pesquisados em artigos científicos e foi utilizado banco de dados e casos já conhecidos para fazer testes e analisar os códigos

Resultados e Discussão

A ideia inicial do projeto era utilizar a formula de Bayes para prever o tempo do semáforo baseado no transito, utilizando de um banco de dados que funcionaria como o conhecimento a priori que pode estar relacionado ao evento. Porém ao testar e estudar o código de Bayes foi concluído que ele não seria útil para este projeto, pois Bayes funciona prevendo novos dados a partir de dados qualitativos e não quantitativos, que é o que buscamos.

Ao analisar outras formulas de estatística foi constatado que o Algoritmo de Gibbs é a formula matemática ideal para se utilizar neste projeto.

1. **Bayes**

O primeiro passo seria adaptar um código já existentes sobre Bayes começas a utilizar ele em um banco de dados já conhecido para saber se ele conseguiria prever e como ele fazia isso. Foi adaptado o código de Sunil Ray do artigo *6 passos fáceis para aprender o algoritmo Naive Bayes (com o código em Python).* (Figura 1)

Figura 1 - Imagem do código adaptado utilizado para testes

```
from sklearn.naive_bayes import GaussianNB
import numpy as np
import xlrd
book = xlrd.open_workbook("Novos_Dados_05-02_Lab.xlsx")
long = []
lat = []
clas = []
sit = 60
for j in range(1,4):
    for i in range(1, sit):
      long.append(book.sheet_by_index(j).cell(rowx=i,colx=3).value)
      lat.append(book.sheet_by_index(j).cell(rowx=i,colx=2).value)
      clas.append(book.sheet_by_index(j).cell(rowx=i,colx=5).value)
lo = np.array(long)
la = np.array(lat)
vel = np.array(clas)
lola = np.stack((la,lo),axis=-1)
temp = []
x_val = np.array([
            [-1418.63016,2795.2811], #+/- 0  ok - 1
            [-1418.62747,2795.27937], #+/- 15 ok - 1
            [-1418.62183,2795.27313], #+/- 25 ok - 2
            [-1418.60397,2795.19694], #+/- 50 ok - 2
            [-1418.61253,2795.18434], #+/- 60 ok - 2
            [-1418.64348,2795.14062], #+/- 70 ok - 2
            [-1418.66808,2795.10686], #+/- 80 ok - 2
            [-1418.95813,2794.888],   #+/- 90 ok - 3
            [-1419.10351,2794.80625], #+/- 100 ok - 3
                ])
model = GaussianNB()
model.fit(lola, vel)
predicted= model.predict(x_val)
print ("Teste com:",sit,"observações")
print (predicted)
```

A primeira parte do código serve para criar as variáveis iniciais (long, lat, class, temp, sit), importar as bibliotecas *numpy* (biblioteca de vetores) e *xlrd* (biblioteca do Excel), inicializar a variável sit com o numero máximo de linhas do banco de dados e a variável book como o nome da planilha que contem o banco de dados (Figura 2).

Figura 2 - Parte inicial do código nomeada de "Set Up"

```
Set Up

from sklearn.naive_bayes import GaussianNB
import numpy as np
import xlrd
book = xlrd.open_workbook("Novos_Dados_05-02_Lab.xlsx")
long = []
lat = []
clas = []
temp = []
sit = 60
```

Em seguida as variáveis são preenchidas pelas informações que estão no banco de dados e depois são convertidas para vetores utilizado a biblioteca numpy. (Figura 3)

Figura 3 - Código da parte de preenchimento das variáveis e conversão para vetores.

```
Preenchimento

for j in range(1,4):
    for i in range(1, sit):
      long.append(book.sheet_by_index(j).cell(rowx=i,colx=3).value)
      lat.append(book.sheet_by_index(j).cell(rowx=i,colx=2).value)
      clas.append(book.sheet_by_index(j).cell(rowx=i,colx=5).value)
lo = np.array(long)
la = np.array(lat)
vel = np.array(clas)
lola = np.stack((la,lo),axis=-1)
x_val = np.array([
            [-1418.63016,2795.2811], #+/- 0  ok - 1
            [-1418.62747,2795.27937], #+/- 15 ok - 1
            [-1418.62183,2795.27313], #+/- 25 ok - 2
            [-1418.60397,2795.19694], #+/- 50 ok - 2
            [-1418.61253,2795.18434], #+/- 60 ok - 2
            [-1418.64348,2795.14062], #+/- 70 ok - 2
            [-1418.66808,2795.10686], #+/- 80 ok - 2
            [-1418.95813,2794.888],   #+/- 90 ok - 3
            [-1419.10351,2794.80625], #+/- 100 ok - 3
                ])
```

Depois que os vetores foram criados e preenchidos, é só utilizar a biblioteca Gausiana para prever o resultado. (Figura 4)

Figura 4 - Parte final do código.

```
Operação

model = GaussianNB()
model.fit(lola, vel)
predicted= model.predict(x_val)
print ("Teste com:",sit,"observações")
print (predicted)
```

Para a realização de teste foi utilizado um banco de dados de um projeto anterior feito no Instituto Mauá de Tecnologia onde baseado na latitude e longitude, o código seria capaz de prever a velocidade que o carro estaria andando e analisamos o desempenho do código a partir da comparação entre o que código previa e oque era real.

Figura 5 - Formato do banco de dados

sats"	time	lat	long	velocity	Classificação
70	175110,5	-1418,63016	2795,2811	0	1
70	175110,6	-1418,63015	2795,2811	0	1
70	175110,7	-1418,63014	2795,28109	0	1
70	175110,8	-1418,63014	2795,2811	0	1
70	175110,9	-1418,63013	2795,28109	0	1
70	175111	-1418,63013	2795,28109	0	1
70	175111,1	-1418,63012	2795,2811	0	1
70	175111,2	-1418,63012	2795,28111	0	1
70	175111,3	-1418,63011	2795,28113	0	1
70	175111,4	-1418,6301	2795,28115	0	1
70	175111,5	-1418,6301	2795,28117	0	1
70	175111,6	-1418,63009	2795,28119	0	1
70	175111,7	-1418,63009	2795,28121	0	1
70	175111,8	-1418,63008	2795,28123	0	1
70	175111,9	-1418,63008	2795,28125	0	1
70	175112	-1418,63007	2795,28127	0	1
70	175112,1	-1418,63007	2795,2813	0	1
70	175112,2	-1418,63006	2795,28131	0	1
70	175112,3	-1418,63006	2795,28133	0	1
70	175112,4	-1418,63006	2795,28135	0	1
70	175112,5	-1418,63005	2795,28137	0	1
70	175112,6	-1418,63005	2795,28139	0	1

Como modo de testar o programa criamos um variável classificação que iria de 1 a 3(um ao três) e que dividisse o banco de dados da seguinte forma: se a velocidade fosse maior que 90 km/h seria um tipo 3, se estivesse entre 90 e 20 km/h seria do tipo 2 e se fosse menor que 20 seria do tipo 1 (Tabela 1). Para poder comparar foi retirado do banco de dados 10 velocidade diferentes e de lugares diferentes para ver se o código era capaz de prever corretamente

Tabela 1 – Critério da divisão, o código de cada divisão e a quantidade em cada uma delas.

Critério	Código	Quantidade
X > 90	3	3408
90 > X > 20	2	11252
20 > X	1	416
Total		15076

Tabela 2 – Dados utilizados para verificar a eficácia do código.

Coordenadas		Velocidade	Classificação
-1418.63016	2795.2811	0	1
-1418.62747	2795.27937	15	1
-1418.62183	-2795.27313	25	2
-1418.60397	-2795.16694	50	2
-1418.61253	-2795.18434	60	2
-1418.64348	-2795.14062	70	2
-1418.66808	-2795.10686	80	2
-1418.95813	-2794.888	90	3
-1419.10351	-2794.80625	100	3

No início havia muitos erros porem quanto mais dados você coloca melhora a previsão fica (Tabela 3) porém foi observado que a enorme diferença entre as divisões acarretou uma sobrecarga no histórico de apenas uma variável oque fez com que o código tivesse muitos referenciais para apenas uma variável. Como efeito colateral ele começou a prever para todas as velocidades o mesmo resultado oque não é possível dado à diferença entre eles(este efeito acontece com as amostras em ordem aleatória ou em uma ordem fixa). (Tabela 4)

Tabela 3 – Desempenho do código com 50 amostras até 2800 amostras

Observaçõe	1	1	2	2	2	2	2	3	3
50	1	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	1	1
200	1	1	1	1	1	1	1	1	1
300	1	1	1	1	1	1	1	1	1
400	1	1	2	2	2	2	2	2	2
500	1	1	1	2	2	1	1	1	1
600	1	1	1	2	2	2	2	1	1
700	1	1	1	2	2	2	2	2	2
800	1	1	1	2	2	2	2	2	2
900	1	1	1	2	2	2	2	2	2
1000	1	1	1	2	2	2	2	3	2
1200	1	1	1	2	2	2	2	3	3
1400	1	1	1	2	2	2	2	3	3
1600	1	1	1	2	2	2	2	3	3
1800	1	1	1	2	2	2	2	2	3
2000	1	1	1	2	2	2	2	2	3
2200	1	1	1	2	2	2	2	2	3
2400	1	1	1	2	2	2	2	2	2
2600	1	1	1	2	2	2	2	2	2
2800	1	1	1	2	2	2	2	2	2

Tabela 4 – Desempenho do código com 9600 amostras até 15077

9200	1	1	1	1	1	1	1	1	1
9400	1	1	1	1	1	1	1	1	1
9600	1	1	1	1	1	1	1	1	1
9800	1	1	1	1	1	1	1	1	1
10000	1	1	1	1	1	1	1	1	1
10200	1	1	1	1	1	1	1	1	1
10400	1	1	1	1	1	1	1	1	1
10600	1	1	1	1	1	1	1	1	1
10800	1	1	1	1	1	1	1	1	1
11000	1	1	1	1	1	1	1	1	1
11200	1	1	1	1	1	1	1	1	1
11400	1	1	1	1	1	1	1	1	1
11600	1	1	1	1	1	1	1	1	1
11800	1	1	1	1	1	1	1	1	1
12000	1	1	1	1	1	1	1	1	1
12200	1	1	1	1	1	1	1	1	1
12400	1	1	1	1	1	1	1	1	1
12600	1	1	1	1	1	1	1	1	1
12800	1	1	1	1	1	1	1	1	1
13000	1	1	1	1	1	1	1	1	1
13200	1	1	1	1	1	1	1	1	1
13400	1	1	1	1	1	1	1	1	1
13600	1	1	1	1	1	1	1	1	1
13800	1	1	1	1	1	1	1	1	3
14000	1	1	1	1	1	1	1	1	3
14200	1	1	1	1	1	1	1	1	3
14400	1	1	1	1	1	1	1	1	3
14600	1	1	1	1	1	1	1	1	3
14800	1	1	1	1	1	1	1	1	3
15000	1	1	1	1	1	1	1	1	3
15077	1	1	1	1	1	1	1	1	3

Ao equilibrar a quantidade de amostras de cada divisão para um mesmo número a eficiência do código muda drasticamente (Tabela 5).

Tabela 5 – Desempenho do código com 50 amostras para cada divisão ate 416 amostras.

Observações	1	1	2	2	2	2	2	3	3
50	1	2	2	2	2	2	3	3	3
100	1	2	2	2	2	2	2	3	3
150	1	2	2	2	2	2	2	3	3
200	1	2	2	2	2	2	2	3	3
250	1	2	2	2	2	2	2	3	3
300	1	2	2	2	2	2	2	3	3
350	1	1	1	2	2	2	2	3	3
400	2	2	2	2	2	2	2	3	3
416	2	2	2	2	2	2	2	3	3

Contudo ao pesquisar mais sobre o código de Bayes foi descoberto que ele apenas previa em que classe o objeto estaria e não o valor dele, oque não é o objetivo deste projeto e também que as variáveis têm que ser qualitativas e não quantitativas, oque não é o objetivo deste projeto também. Baseado em tudo oque foi feito ate agora foi considerado descarta o código de Bayes e procurar uma nova formula.

2. Amostrador de Gibbs

Procurando informações sobre códigos ou formulas similares a Bayes foi constatado que na engenharia ainda não é muito aplicado este conceito hoje em dia, mas na área da biomedicina é um conceito muito utilizado para prevenir genes.

Fazendo pesquisa sobre neste campo de estudo encontramos uma tese de mestrado na qual explicava todos os métodos para se simular genes através de monte carlo via cadeias de markov (Nascimento, 2009). Dentre estes métodos foi encontrado o amostrador de Gibbs que é atualmente a formula utilizada para realizar este trabalho.

O código de Gibbs usa a mesma base do código de Bayes (área de set up, de preenchimento e de conversão para vetor) porém na parte de conclusão torna-se mais complexa e detalhada. Este codigo foi baseado e adaptado do artigo *Gibbs sampling for Bayesian linear regression in Python* de Kieran R Campbell.

Para realizar os testes foi utilizando um banco de dados onde as variáveis eram os pesos atômicos de cada elemento químico e o índice de refração do vidro (Figura 6).

Figura 6 - Banco de dados para o teste do Amostrador de Gibbs

ID	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
1	1,52101	13,64	4,49	1,1	71,78	0,06	8,75	0	0
2	1,51761	13,89	3,6	1,36	72,73	0,48	7,83	0	0
3	1,51618	13,53	3,55	1,54	72,99	0,39	7,78	0	0
4	1,51766	13,21	3,69	1,29	72,61	0,57	8,22	0	0
5	1,51742	13,27	3,62	1,24	73,08	0,55	8,07	0	0
6	1,51596	12,79	3,61	1,62	72,97	0,64	8,07	0	0,26
7	1,51743	13,3	3,6	1,14	73,09	0,58	8,17	0	0
8	1,51756	13,15	3,61	1,05	73,24	0,57	8,24	0	0
9	1,51918	14,04	3,58	1,37	72,08	0,56	8,3	0	0
10	1,51755	13	3,6	1,36	72,99	0,57	8,4	0	0,11
11	1,51571	12,72	3,46	1,56	73,2	0,67	8,09	0	0,24
12	1,51763	12,8	3,66	1,27	73,01	0,6	8,56	0	0
13	1,51589	12,88	3,43	1,4	73,28	0,69	8,05	0	0,24
14	1,51748	12,86	3,56	1,27	73,21	0,54	8,38	0	0,17
15	1,51763	12,61	3,59	1,31	73,29	0,58	8,5	0	0
16	1,51761	12,81	3,54	1,23	73,24	0,58	8,39	0	0
17	1,51784	12,68	3,67	1,16	73,11	0,61	8,7	0	0
18	1,52196	14,36	3,85	0,89	71,36	0,15	9,15	0	0
19	1,51911	13,9	3,73	1,18	72,12	0,06	8,89	0	0
20	1,51735	13,02	3,54	1,69	72,73	0,54	8,44	0	0,07
21	1,5175	12,82	3,55	1,49	72,75	0,54	8,52	0	0,19
22	1,51966	14,77	3,75	0,29	72,02	0,03	9	0	0
23	1,51736	12,78	3,62	1,29	72,79	0,59	8,7	0	0
24	1,51751	12,81	3,57	1,35	73,02	0,62	8,59	0	0

A primeira parte do código é parecida com a outra, inicialmente importamos as bibliotecas, criamos as variáveis, inicializamos a variável de contagem sit e preenchemos os vetores.(Figura 7)

Figura 7 - Parte inicial do código onde ocorre a criação e preenchimento de variáveis e a conversão das mesmas para vetores.

```
import numpy as np
import pandas as pd
import xlrd
book = xlrd.open_workbook("Programa_Gibbs_Vidro.xlsx")
sh = book.sheet_by_index(0)
RI = []
Elemento = []
sit = 215
for i in range(1,sit):
      RI.append(sh.cell(rowx=i,colx=1).value)
      Elemento.append(sh.cell(rowx=i,colx=4).value)
y = np.array(RI)
x = np.array(Elemento)
```

A segunda parte do código é a parte onde é transcrito a formula matemática para a linguagem computacional. É nesta parte do código onde são previstas as variáveis necessárias para realizar a distribuição (Beta 0, Beta 1 e Tao)(Figura 8 e 9).

Figura 8 – Primeira parte do código onde são prevista as variáveis necessárias para construir o Amostrador de Gibbs

```
def sample_beta_0(y, x, beta_1, tau, mu_0, tau_0):
    N = len(y)
    assert len(x) == N
    precision = tau_0 + tau * N
    mean = tau_0 * mu_0 + tau * np.sum(y - beta_1 * x)
    mean /= precision
    return np.random.normal(mean, 1 / np.sqrt(precision))

def sample_beta_1(y, x, beta_0, tau, mu_1, tau_1):
    N = len(y)
    assert len(x) == N
    precision = tau_1 + tau * np.sum(x * x)
    mean = tau_1 * mu_1 + tau * np.sum( (y - beta_0) * x)
    mean /= precision
    return np.random.normal(mean, 1 / np.sqrt(precision))

def sample_tau(y, x, beta_0, beta_1, alpha, beta):
    N = len(y)
    alpha_new = alpha + N / 2
    resid = y - beta_0 - beta_1 * x
    beta_new = beta + np.sum(resid * resid) / 2
    return np.random.gamma(alpha_new, 1 / beta_new)
```

Figura 9 - Segunda parte do código

```
beta_0_true = -1
beta_1_true = 2
tau_true = 1

init = {"beta_0": 0,
        "beta_1": 0,
        "tau": 2}

hypers = {"mu_0": 0,
          "tau_0": 1,
          "mu_1": 0,
          "tau_1": 1,
          "alpha": 2,
          "beta": 1}

def gibbs(y, x, iters, init, hypers):
    assert len(y) == len(x)
    beta_0 = init["beta_0"]
    beta_1 = init["beta_1"]
    tau = init["tau"]

    trace = np.zeros((iters, 3)) ## trace to store values of beta_0, beta_1, tau

    for it in range(iters):
        beta_0 = sample_beta_0(y, x, beta_1, tau, hypers["mu_0"], hypers["tau_0"])
        beta_1 = sample_beta_1(y, x, beta_0, tau, hypers["mu_1"], hypers["tau_1"])
        tau = sample_tau(y, x, beta_0, beta_1, hypers["alpha"], hypers["beta"])
        trace[it,:] = np.array((beta_0, beta_1, tau))

    trace = pd.DataFrame(trace)
    trace.columns = ['beta_0', 'beta_1', 'tau']

    return trace
```

Na ultima parte do código esta o código de construção de gráficos para melhor visualização dos resultados. (Figura 10)

Figura 10 - Área do código destina a construção do gráfico

```
iters = 100000
trace = gibbs(y, x, iters, init, hypers)

traceplot = trace.plot()
traceplot.set_xlabel("Iteration")
traceplot.set_ylabel("Parameter value")

trace_burnt = trace[500:999]
hist_plot = trace_burnt.hist(bins = 30, layout = (1,3))

print(trace_burnt.median())
print(trace_burnt.std())
```

Para testar o desempenho do amostrador de Gibbs primeiro foi feito um teste Anova e regressão linear com os dados para saber se de é fato era possível utilizar este banco de dados e também para comparar com os resultados do código. (Figura 11). Foi concluído que com 94% de certeza é possível prever o índice de refração do vidro com regressão linear.

Figura 11- Tabelas com os testes Anova e regressão linear

RESUMO DOS RESULTADOS

Estatística de regressão	
R múltiplo	0,945932107
R-Quadrado	0,894787552
R-quadrado ajustado	0,8906817
Erro padrão	0,001004088
Observações	214

ANOVA

	gl	*SQ*	*MQ*	*F*	*F de significação*
Regressão	8	0,001757722	0,000219715	217,9298	7,90095E-96
Resíduo	205	0,000206679	1,00819E-06		
Total	213	0,001964401			

	Coeficientes	*Erro padrão*	*Stat t*	*valor-P*	*95% inferiores*	*95% superiores*	*Inferior 95,0%*	*Superior 95,0%*
Interseção	1,453267306	0,067039348	21,67782568	6,01E-55	1,321092292	1,58544232	1,321092292	1,58544232
Na	0,001395259	0,000655051	2,129999758	0,034364	0,000103758	0,00268676	0,000103758	0,00268676
Mg	0,001844231	0,000675466	2,730308044	0,006878	0,000512479	0,003175983	0,000512479	0,003175983
Al	3,26236E-05	0,000698337	0,04671618	0,962785	-0,001344219	0,001409466	-0,001344219	0,001409466
Si	0,000168506	0,000677398	0,248754232	0,8038	-0,001167055	0,001504066	-0,001167055	0,001504066
K	0,001382866	0,000689985	2,004197244	0,046365	2,24892E-05	0,002743243	2,24892E-05	0,002743243
Ca	0,003116768	0,000668372	4,663221895	5,61E-06	0,001799003	0,004434533	0,001799003	0,004434533
Ba	0,002982806	0,000675999	4,412442852	1,65E-05	0,001650004	0,004315607	0,001650004	0,004315607
Fe	0,000426273	0,000778689	0,547424472	0,584683	-0,001108992	0,001961538	-0,001108992	0,001961538

Para testar o Amostrador de Gibbs foi realizado testes onde o código iria fornecer os Beta 0 e os Beta 1 da média e do desvio padrão para calcular Beta 0 e Beta 1 de Gibbs através do Excel e depois disso era feito um erro padrão comparando o resultado esperado com o obtido(Figura 12). Cada Beta 0 e Beta 1 foi calculado numa media de 10 resultados gerado pelo código (Figura 13) e como este código só tem como entrada um variável foi necessário realizar este experimento para cada uma das variáveis.

Figura 12 - Formato da tabela de teste

Erro	RI	yi	B0	B1	Ca				
0,023029	1,52101	1,485983	1,4981106746	-0,00139	8,75		Erro medio	3,03%	
0,035972	1,51761	1,463019	1,4419992033	0,002685	7,83				
0,015716	1,51618	1,492352	1,4896573197	0,000346	7,78		Nova base	Ui	Ci
0,005554	1,51766	1,526089	1,4733834820	0,006412	8,22		B0	1,49	0,04
0,041213	1,51742	1,454883	1,4834971629	-0,00355	8,07		B1	0,003	0,004
0,090642	1,51596	1,37855	1,4550136851	-0,00948	8,07				

Figura 13 - Tabela de média para a determinação de Beta 0 e Beta 1 da media e da desvio padrão.

Base	Ui	Ci		1	Ui	Ci		6	Ui	Ci
B0	1,48	0,03		B0	1,519815	0,018375		B0	1,519108	0,021147
B1	0,003	0,003		B1	-0,00135	0,012049		B1	-0,00176	0,013646
Média	Ui	Ci		2	Ui	Ci		7	Ui	Ci
B0	1,520	0,018		B0	1,519734	0,020509		B0	1,520523	0,02146
B1	-0,0011	0,012		B1	-0,00066	0,013334		B1	-0,00117	0,01403
Nova base	Ui	Ci		3	Ui	Ci		8	Ui	Ci
B0	1,52	0,018		B0	1,518757	0,017521		B0	1,525171	0,017638
B1	-0,0011	0,012		B1	0,0006	0,011675		B1	-0,00423	0,011447
				4	Ui	Ci		9	Ui	Ci
				B0	1,518678	0,019193		B0	1,520795	0,019213
				B1	-0,0006	0,012496		B1	-0,00173	0,012461
				5	Ui	Ci		10	Ui	Ci
				B0	1,515777	0,020868		B0	1,522791	0,019154
				B1	0,001586	0,013519		B1	-0,00269	0,012489

No final do teste foi observado que:

- Regressão linear é menos precisa na maioria dos casos(Figura 14). Enquanto a regressão linear chega a 94% de acerto há elementos em que o amostrador de Gibbs tem uma probabilidade de acerto de 99,55%.
- Ambos diferem em relação ao elemento irrelevante. Na regressão linear aponta que o Al não altera os dados, mas ao realizar o teste com o amostrador de Gibbs é possível prever utilizando o Al com 98,55% de certeza. O amostrador de Gibbs não consegue prever utilizando o Si, ele tem apenas 65,83% de chance de acertar o resultado.

É possível utilizar este algoritmo no projeto porém o grande empecilho atual é o fato de que ele só prevê utilizando apenas 1 variável enquanto para o projeto seria necessário que funcionasse utilizando 2 variáveis.

Figura 14 - Analise final, comparação entre regressão linear e amostrador de Gibbs

	Regressão linear	Gibbs
Ca	94%	97,3%
Na	94%	95,19%
Mg	94%	99,08%
Si	94%	65,83%
K	94%	99,53%
Ba	94%	99,55%
Fe	94%	99,47%
Al	-	98,55%

Referências Bibliográficas

CAMPBELL, Kieran R. **Gibbs sampling for Bayesian linear regression in Python.** São Paulo 2018. Disponivel em; https://kieranrcampbell.github.io/blog/2016/05/15/gibbs-sampling-bayesian-linear-regression.html. Accesso em 29 de outubro de 2018

NASCIMENTO, Moysés. **O uso de simulação de monte carlo via cadeias de markov no melhoramento genetico**. Tese (Mestrado em Biologia), Universidade Federal de Viçosa, Viçosa, Minas Gerais, 2009.

RAY, Sunil. **6 passos fáceis para aprender o algoritmo Naive Bayes (com o código em Python).** São Paulo: 2018. Disponivel em: https://www.vooo.pro/insights/6-passos-faceis-para-aprender-o-algoritmo-naive-bayes-com-o-codigo-em-python/. Acesso em 29 de outubro de 2018