Previsão de terremotos utilizando *Support Vector Regression*

John Theo S de Souza

Resumo—Este artigo tem o objetivo de mostrar a aplicação de *Support Vector Regression* para a previsão do tempo de ocorrer um terremoto. O dataset utilizado foi disponibilizado pelo Laboratório Nacional de Los Alamos(LALN) através de uma competição no Kaggle [1] e consiste e dados colhetados em um experimento laboratorial de modo a simular um terremoto. Por possuir apenas duas propriedades(dado acústico e tempo de falha), foi feito (*feature engeneering*) de modo a obter mais parâmetros para aplicar o método de *Support Vector Regression* com um kernel *Radial Basis Function*

Palavras chave—Reconhecimento de Padrões, Support Vector Regression, terremoto

1 Introdução

Devido as consequências devastadoras que um terremoto traz, a previsão de um terremoto tornou-se um dos problemas mais importantes dentro das Ciências da Terra a ser pesquisado. Atualmente os principais focos de pesquisa são: **quando** um evento vai ocorrer, **onde** vai ocorrer e a **magnitude**. Este artigo foca utilizar uma técnica de reconhecimento de padrões conhecida como *Support Vector Regression* para prever **quando** um terremoto vai acontecer. Mesmo não sendo a melhor técnica para essa finalidade, foi escolhida para fins de estudo e para mostrar uma possível abordagem para o problema.

O *dataset* utilizado foi disponibilizado na competição *"LANL Earthquake Prediction"* do portal Kaggle [1] e consiste 600 milhões de linhas de dados compostos por dois valores: dado acústico e tempo até acontecer o terremoto. Sobre esse *dataset* aplicou-se a técnica de *Feature Engeneeting* de modo a obter mais valores para aplicar a *Support Vector Regression* com um kernel de base radial pois trata-se de um problema não linear.

As próximas sessões discutem a metodologia utilizada, bem como explica de forma sucinta as técnicas utilizadas.

2 Metodologia

A partir *dataset* obitdo do Kaggle [1], foi feita uma análise exploratória dos dados para ter uma noção dos dados e ajudar a formular uma heurística para abordar o problema. A partir dessa análise gerou-se novas features e aplicar o modelo de *Support Vector Regression* fornecido pela biblioteca SciKit-Learn [2]. Os hiper-parâmetros(C e gama) do modelo foram otimizados utilizando GridSearch, através de um biblioteca também fornecida pelo SciKit-Learn [2] e após validado no conjunto de validação cruzada. Os detalhes sobre como cada um dos passos foi feito, são encontrados nas próxima sessões.

2.1 Dataset

Fornecido em uma competição do portal Kaggle [1], os dados vieram de um experimento criado para o estudo da física relacionada aos terremotos.O experimento consiste de duas placas sendo pressionadas até que ocorra o deslise de uma em relação a outra(terremoto). Nesse experimento foram coletados dois dados: dado acústico que refere-se ao sinal produzido pelo atrito entre as placas e o tempo para falha, que refere-se ao tempo restante antes do próximo terremoto experimental.

Os dados de treinamento são um segmento único e contínuo de dados experimentais com 600 milhões de registros com o par de dados: dado acústico(*acoustic_data*) e tempo para falha(*time_to_failure*), onde ocorrem 16 eventos de terremotos. Os dados de testes consistem em uma pasta com diversos segmentos menores. Cada arquivo de teste é contínuo, mas eles juntos não representam um segmento contínuo do experimento, sendo assim as predições não devem contemplar seguir a ordem que os arquivos são listados na pasta.

Dado isso, o objetivo é prever um tempo para falha, para cada segmento na pasta de testes.

2.1.1 Exploração dos dados

O primeiro passo realizado, foi explorar os dados de treinamentos e teste de modo obter direcionamentos para as técnicas utilizadas.

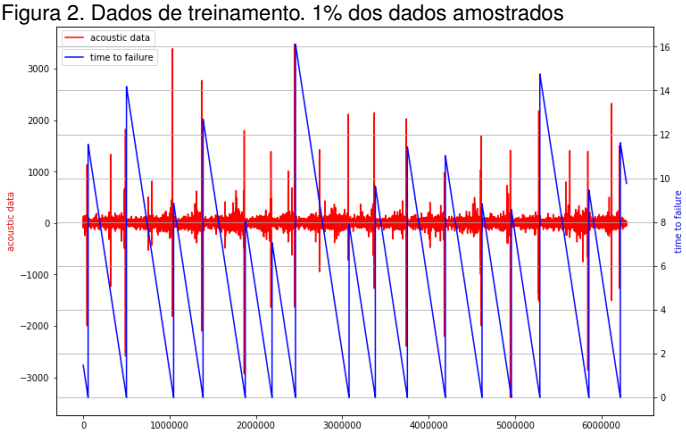
Figura 1. Dados de treinamento. Informações sobre os dados

	acoustic_data	time_to_failure
count	6.291454800000000e+08	6.291454800000000e+08
mean	4.519467573700124e+00	4.477084279060364e-01
std	1.073570724951096e+01	2.612789392471313e+00
min	-5.515000000000000e+03	9.550396498525515e-05
25%	2.000000000000000e+00	2.625997066497803e+00
50%	5.000000000000000e+00	5.349797725677490e+00
75%	7.000000000000000e+00	8.173395156860352e+00
max	5.444000000000000e+03	1.610740089416504e+01

A figura 1 nos dá um referência sobre a escala dos dados que vai nos ajudar posteriormente na implementação do modelo

Após isso foi feita uma amostragem de 1% dos dados de modo a obter uma visualização do comportamento dos mesmos. O resultado pode ser visto na figura 2 onde é possível perceber um padrão de que após um determinado pico no dado acústico, segue-se um evento de falha(terremoto experimental).

Figura 2. Dados de treinamento. 1% dos dados amostrados



Na figura 3, aproximou-se a visualização em torno de um evento de terremoto e nela podemos ver que, na verdade, a grande oscilação antes da falha não é exatamente o último. Tanto antes como depois do picos existem oscilações de diversas naturezas e somente depois de um tempo que de fato ocorre o terremoto.

Figura 3. Visualização dos dados em torno de uma falha

2.1.2 *Preparação dos dados*

Após analisar os segmentos de teste e verificar que cada um dele continha 150000 registros, dividiu-se o segmento de treinamento em segmentos compatíveis com o tamanho do teste, para que assim o treinamento fosse coerente. O resultado dessa divisão foi um conjunto de 4195 segmentos que serão utilizados para o treinamento.

Porém esses segmentos continuam apenas com os dois valores(acoustic_data e time_to_failure). Sendo assim fez-se necessária a aplicação de um método conhecido como *Feature Engineering*. Uma *feature* é uma representação numérica que deriva do dado cru e está atrelada ao modelo, de modo que algumas features são mais relevantes para alguns modelos. *Feature Engineering* é o processo de formular as *features* mais apropriadas dado um dataset, um modelo e uma tarefa. [3]

Para esse trabalho foi gerado um conjunto de *features* arbitrárias estatísticas arbitrárias de modo a acrescentar informação e poder aplicar de forma mais efetiva o modelo. Sendo assim o conjunto de *features* foi composto de valores como: média, mediana, desvio padrão, quantil, curtose, assimetria. O resultado foi um conjunto de treinamento com 4195 registros com 13 *features* a serem injetadas no modelo. Todos os valores foram então normalizados utilizando a biblioteca ***StandardScaler*** do *SciKit Learn* [2] de modo evitar distorções no aprendizado.

Para gerar o conjunto de dados de saída, a cada bloco de 150000 registros, utilizou-se o último valor como resultante. Esse conjunto será usado para treinar o modelo e validar seu *score*.

2.2 Support Vector Regression

A ideia por trás do *Support Vector Regression* é bem parecida com a do aplicada a *Support Vector Machine*. O diferencial encontra-se que ao posso que a *Support Vector Machine* busca encontrar um hiperplano que separe os dados com a maior margem possível(definida pelos vetores de suporte), na *Support Vector Regression* o objetivo é encontrar um hiperplano(com margens delimitadas pelos vetores de suporte) que contenha a maior quantidade de pontos possíveis, ou seja, não importa que o ponto não esteja exatamente sobre o hiperplano contanto que esteja dentro da margem [4].

Em termos matemáticos dado um conjunto de treinamento $D = (x1, y1), ..., (xM, yM)$ com M amostras onde $\mathbf{x}^i \in \Re^n$ são entrada multidimensionais e $y_i \in \Re^n$ são saídas contínuas unidimensionais, deseja-se encontrar uma função contínua $f : \Re \Rightarrow \Re$ que melhor descreva os pontos do conjunto de treinamento com a função: $y = f(x)$. [5]

Sendo assim, procuramos um estimativa para seguinte função:

$$f(x) = \langle \mathbf{w}^T, \mathbf{x} \rangle + b \text{ com } \mathbf{w} \in \Re^N, n \in \Re \tag{1}$$

sendo que os pontos que estiverem contidos no ϵ-tubo não são penalizados, $f(x) - y \leq \epsilon$. Tal problema pode ser visto como um problema de otimização e a função $f(x)$ pode ser aprendida minimizando o seguinte:

$$\begin{aligned} &\min_{w,\xi,\xi^*} \left(\tfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{M} (\xi_i + \xi_i^2) \right) \\ &\text{sendo} \begin{cases} y^i - \langle \mathbf{w}^t, \mathbf{x}^i \rangle - b \leq \epsilon + \xi_i^* \\ \langle \mathbf{w}^t, \mathbf{x}^i \rangle + b - y^i \leq \epsilon + \xi_i \\ \xi_i, \xi_i^* \geq 0, \forall_i = 1...M, \epsilon \geq 0 \end{cases} \end{aligned} \tag{2}$$

onde $\mathbf{w}$ é o hiperplano, ξ_i, ξ_i^* são as variáveis *slack*, b é o bias, ϵ é o erro permitido sem penalização e C é o fator de penalidade para erros maiores que ϵ. Logo resolvendo o dual e aplicando a *kernel trick* temos:

$$y = f(x) = \sum_{i=1}^{M} (\alpha_i - \alpha_i^*) k(\mathbf{x}, \mathbf{x}^i) + b \tag{3}$$

onde α_i são os multiplicadores Lagrangianos que definem os vetores de suporte(*support vectors*) e $k(\mathbf{x}, \mathbf{x}^i)$ é a função kernel que para esse artigo foi usada a *Radial Base Function*(RBF) da forma:

$$e^{-\gamma\|\mathbf{x}-\mathbf{x}^i\|^2} \text{ onde } \gamma = \frac{1}{2\sigma^2} \tag{4}$$

Logo ao final, temos os hiperpâmetros C (penalização) e γ (distribuição da gaussiana RBF) a serem injetados no modelo.

2.2.1 Definição dos hiperparâmetros

Uma das formas mais comuns de otimizar os hiper parâmetros é o *Grid Search*, que nada mais é do que uma busca exaustiva através de um conjunto de parâmetros especificados manualmente. O algoritmo de *Grid Search* deve ser guiado por alguma métrica de performance e tipicamente calculando usando um conjunto de validação cruzada. [6]

Sendo assim, o algoritmo de *Grid Serch* da biblioteca *Sci-Kit Learn* [2] irá fazer o produto cartesiano desses valores, aplicar ao modelo e retornar o que tiver o melhor *score*.

3 Resultados

Para realizar os ajustes nos parâmetros do modelo de modo chegar num melhor conjunto de valores, dividiu-se os dados de treino em duas partes na proporção de 80% para treino e 20% para validação cruzada, com isso foram sendo testados conjuntos de valores para C e γ até chegar ao conjunto de valores para esse artigo que foi:

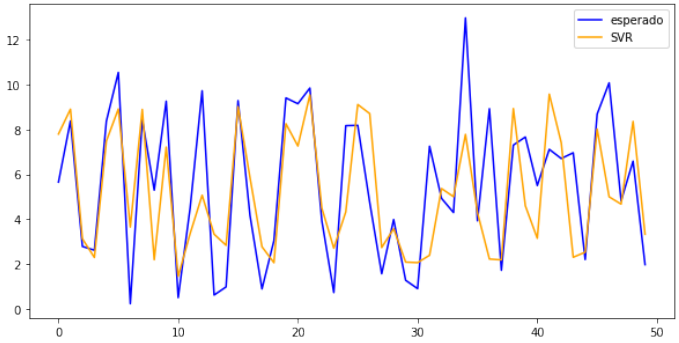
$$\gamma = [0.001, 0.005, 0.01, 0.02, 0.05, 0.1]$$
$$C = [0.1, 0.2, 0.25, 0.5, 1, 1.5, 2]$$

Ao submeter o resultado para a competição do Kaggle, é atribuido um score referente ao erro médio absoluto. O score do modelo implementado nesse artigo foi:

- Conjunto de treino: 2.061
- Conjunto de validação cruzada: 1.992
- Conjunto de teste: 1.559

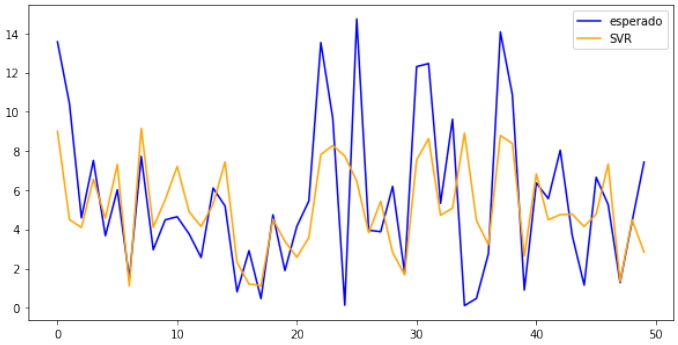
Na figura 4 foi amostrado os primeiros 50 pontos do resultado obtido com o SVR e o resultado esperado para o conjunto de **validação cruzada**

Figura 4. **Cross Validation** - Resultado Esperado vs Resultado obtido



Na figura 5 foi amostrado da mesma forma que anteriormente, porém para o conjunto de treino

Figura 5. **Treino** - Resultado Esperado vs Resultado obtido



Como esperado, devido ao score, a curva obtida no conjunto de treino diverge mais do esperado do que a obtida no conjunto de validação cruzada. Não foi possível gerar um gráfico para o cojunto de teste pois os valores são conhecidos apenas pela competição.

Isso mostra que o modelo não sofreu *overfitting* e é capaz de lidar com novos dados *outliers*.

4 Conclusão

Foi possível realizar uma regressão no dataset utilizando o SVR e obter uma $f(x)$ que se aproximou do resultado esperado no conjunto de validação cruzada. Como proposta futura sugere-se um cuidado maior com o processo de *feature engineering* de modo a validar a importância de cada *feature* para o modelo bem como adicionar mais features. Em outras implementações encontradas no GitHub onde é feito um processo de feature engineering para séries temporais, um conjunto maior de *features* é avaliado e com a ajuda do conjunto de validação cruzada são definidas as melhores *features* a serem mantidas no modelo.

FeatureTools [7] é um *framework* open source onde o processo de Feature Engineering é feito de forma automática desde que os dataset seja colocado no formato aceito pelo framework. Para esse trabalho não foi utilizado esse *framework*, pois desejava-se validar o método de aplicar um modelo de reconhecimento de padrões em cima de um dataset existente, mas para trabalhos mais aprofundados esse framework pode facilitar esse passo do processo.

Ao utilizar a heurística de GridSearch, acabou-se discretizando um espaço contínuo de possíveis valores a serem assumidos pelos hiperparâmetros. Abordagens como *Gradient Boosting* e Algoritmo Genético poderiam obter comportamentos melhores ao passo que trabalhariam com um espaço contínuo de valores para os hiperparâmetros.

Na figura 4 acredita-se que as grandes divergências em alguns pontos, deve-se ao fato de ao fazer a divisão do segmentos do conjunto de treinamento, não ter a certeza que todos os segmentos continham um ponto de pico. Isso atrelado a *features* menos complexas, pode ter levado a um aumento do erro em alguns pontos.

Por fim, tendo como objetivo utilizar um método de reconhecimento de padrões para abordar um dataset e realizar previsões, esse artigo cumpriu o que havia proposto mostrando que abordagens mais simplistas podem levar a resultados ótimos se bem modelado o problema.

APÊNDICE A
ALGORITMO DE PREPARAÇÃO DOS DADOS

```
#Definindo a funcao
def gen_features(X):
    strain = []
    strain.append(X.mean())
    strain.append(X.std())
    strain.append(X.min())
    strain.append(X.max())
    strain.append(X.kurtosis())
    strain.append(X.skew())
    strain.append(
        np.quantile(X,0.01))
    strain.append(
        np.quantile(X,0.05))
    strain.append(
        np.quantile(X,0.95))
    strain.append(
        np.quantile(X,0.99))
    strain.append(np.abs(X).max())
    strain.append(np.abs(X).mean())
    strain.append(np.abs(X).std())
    return pd.Series(strain)

#Lendo os dados
train_input = pd.read_csv(
    'train.csv',
    iterator=True,
    chunksize=150_000,
    dtype={
      'acoustic_data': np.int16,
      'time_to_failure':np.float64}
)

#Gerando os conjuntos
X_train = pd.DataFrame()
y_train = pd.Series()
for df in train_input:
    ch = gen_features(
        df['acoustic_data'])
    X_train = X_train.append(
        ch, ignore_index=True)
    y = df['time_to_failure']
            .values[-1]
    y_train = y_train.append(y)

#Normalizando os dados
scaler = StandardScaler()
scaler.fit(X_train)
scaled_train_X = pd.DataFrame(
    scaler.transform(X_train),
    columns=X_train.columns)
```

APÊNDICE B
TREINAMENTO COM *Support Vector Regression*

```
parameters = [{
    'gamma': [
        0.001,
        0.005,
        0.01,
        0.02,
        0.05,
        0.1],
    'C': [
        0.1,
        0.2,
        0.25,
        0.5,
        1,
        1.5,
        2]}]

svr = GridSearchCV(
    SVR(kernel='rbf', tol=0.01),
    parameters,
    cv=5)
svr.fit(
    scaled_train_X,
    y_train.values.flatten())
y_pred = svr
           .predict(scaled_train_X)
```

REFERÊNCIAS

[1] R.-L. Bertrand, C. Hulbert, and P. J. B. Rouet-Leduc, "LANL Earthquake Prediction." https://www.kaggle.com/c/LANL-Earthquake-Prediction, 2018.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] H. Liu, *Feature Engineering for Machine Learning and Data Analytics*. O'Reilly, 2018.

[4] A. J. Smola, B. Schölkopf, and S. Schölkopf, "A Tutorial on Support Vector Regression," 2003.

[5] A. Billard, N. Figueroa, and D. Lamotte, "Advanced machine learning practical 4: Regression (svr, rvr, gpr)," 2016.

[6] Chih-Wei Hsu, Chih-Chung Chang, C.-J. Lin, and Chih-Wei Hsu, "A Practical Guide to Support Vector Classification," *BJU international*, vol. 101, no. 1, pp. 1396–1400, 2008.

[7] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pp. 1–10, IEEE, 2015.