Trabalho Prático 2: Aprendizado de Máquina

Ricardo Dias Avelar
Matrícula: 2019054960

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brazil

ricardodiasmode@hotmail.com

1. Introdução

Esta documentação tem como objetivo facilitar a compreensão do programa que implementa um modelo de Boosting, no caso o AdaBoost, em um problema de classificação binária utilizando o dataset Tic Tac Toe Endgame. Ainda, vários testes foram feitos mudando parâmetros como número de estimadores, tipo de classificador (SVM ou Tree), profundidade de árvore, C do SVM, e por curiosidade o número de folds.

2. Implementação

O código tem algumas funções auxiliares. São elas:

```
def get_estimator(in_estimator_type, in_estimator_param):
```

Que dado uma string “in_estimator_type” e um valor “in_estimator_param”, é criado o classificador passado pela string com o parâmetro passado. O classificador criado é retornado pela função.

```
def get_data():
```

Pega os dados no diretório local e retorna uma tupla sendo [features, labels].

```
def k_fold_cross_validation(in_features, in_labels, fold_amount,
in_estimators, in_tree_depth, in_estimator_type):
```

É a função que chamamos para criar, treinar e avaliar nosso classificador. Retorna a acurácia após o teste. Para mudarmos facilmente os parâmetros de teste, além das features e labels, temos como parâmetro o número desejado de folds, estimators, altura da árvore e o tipo do classificador.

Finalmente, temos a classe AdaBoost que contém a função *fit*, que treina os estimators, e a função *predict*, que retorna a predição após o treinamento.

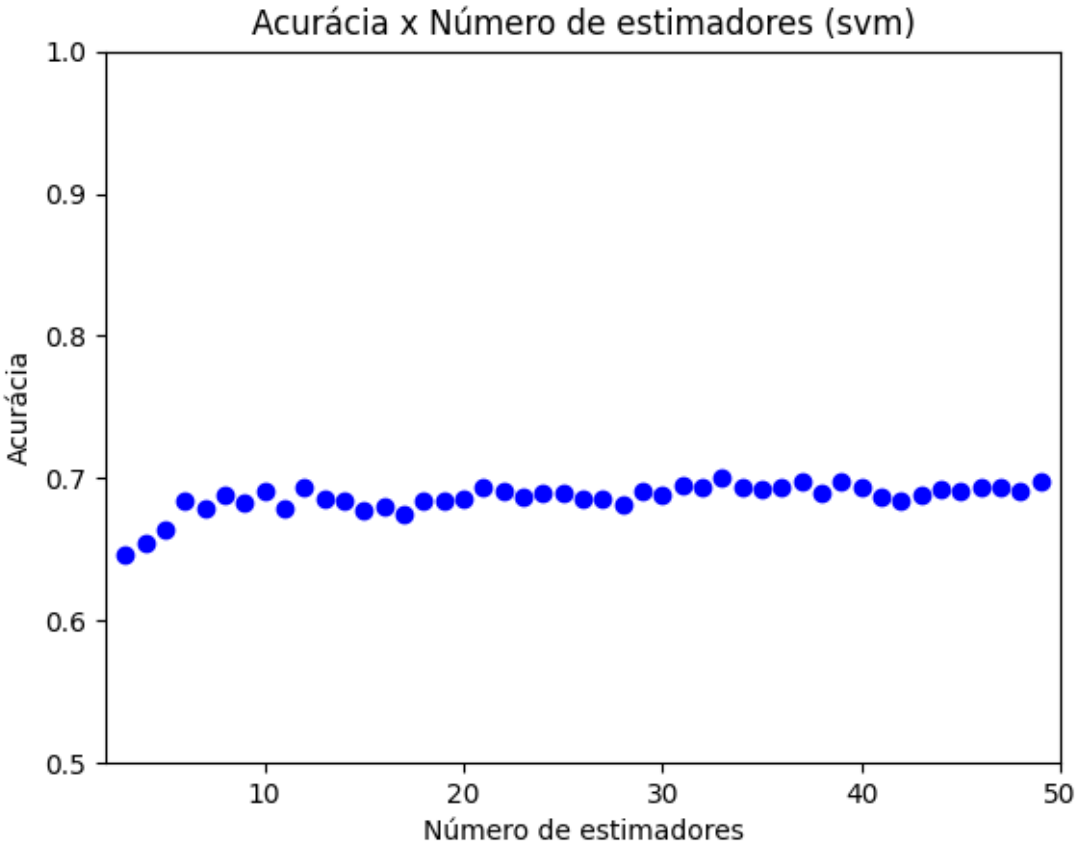
No fim do código temos quatro loops que chamam a função *k_fold_cross_validation* com parâmetros diferentes e criando gráficos para testarmos diferentes modelos.

No primeiro loop testamos o SVM sete vezes, alterando o número de estimators de 3 a 9, para testar a influência do número de estimators do boosting no SVM.

```
    for estimator_type in ['tree', 'svm']:
        for estimators in range(3, 10):
            tree_depth = 3
                accuracy.append(k_fold_cross_validation(features, labels,
n_folds, estimators, tree_depth, estimator_type))
            plt.plot(estimators, accuracy[estimators - 3], 'bo')
        plt.xlabel('Número de estimadores')
        plt.ylabel('Acurácia')
```

```
            plt.title('Acurácia x Número de estimadores' + ' (' +
estimator_type + ')')
        plt.axis([2, 10, 0.5, 1])
        plt.show()
```

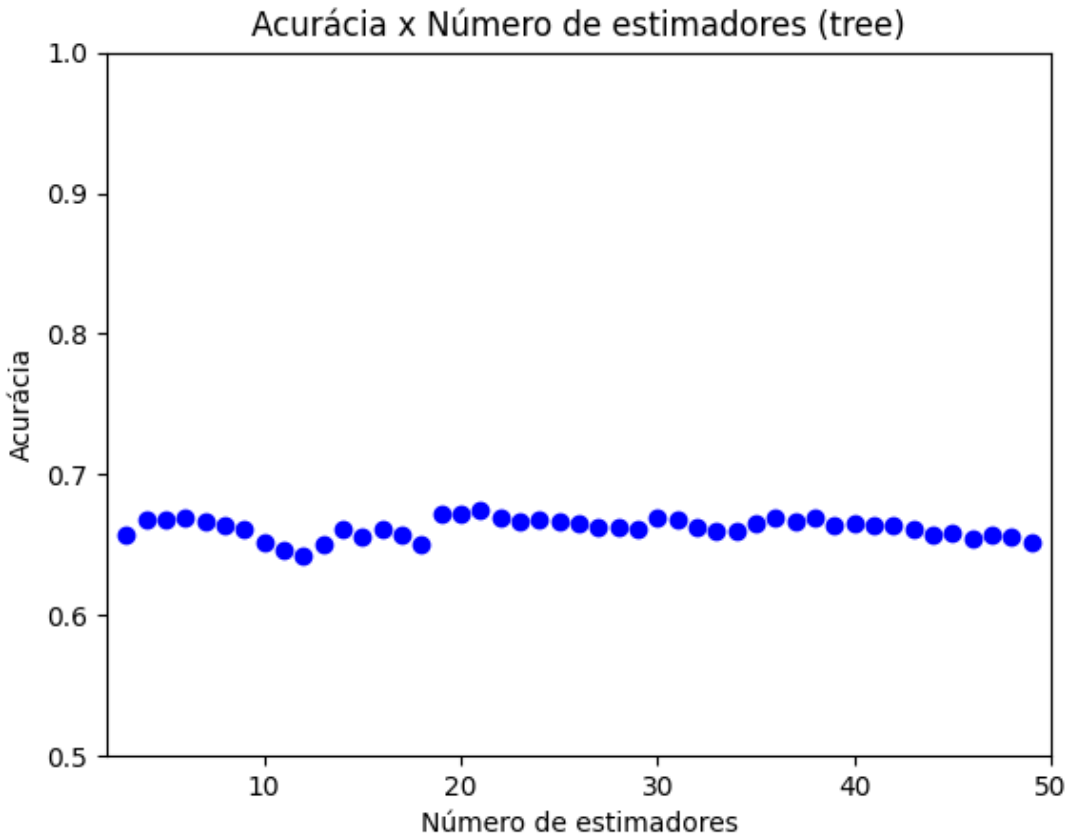
Aqui está o resultado obtido:



Percebe-se que o número de estimadores, depois que grande suficiente, não influencia significativamente na acurácia do modelo. Isso ocorre por dois motivos:

1. À medida que o número de estimators aumenta, o modelo tende a atingir um ponto de saturação. Isso ocorre porque, após um certo ponto, o modelo já aprendeu a maior parte das informações relevantes dos dados de treinamento. Dessa forma, aumentar o número de estimators além desse ponto não traz ganhos significativos em termos de acurácia.
2. O aumento do número de estimators também pode levar a um aumento na complexidade do modelo, o que pode levar ao overfitting.

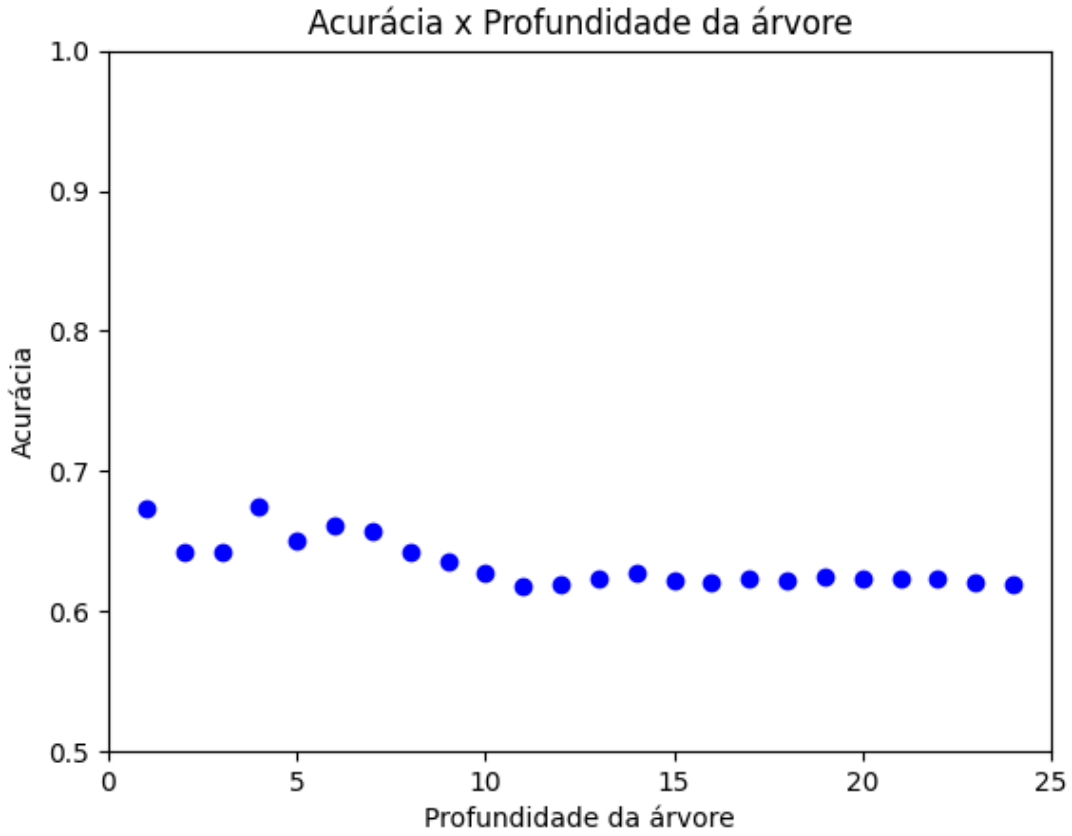
Ainda nesse loop, testamos a árvore nas mesmas condições. Por mais que tenha feito uma curva ligeiramente diferente, aumentar o número de estimators não causa aumento na acurácia pelos mesmos motivos.



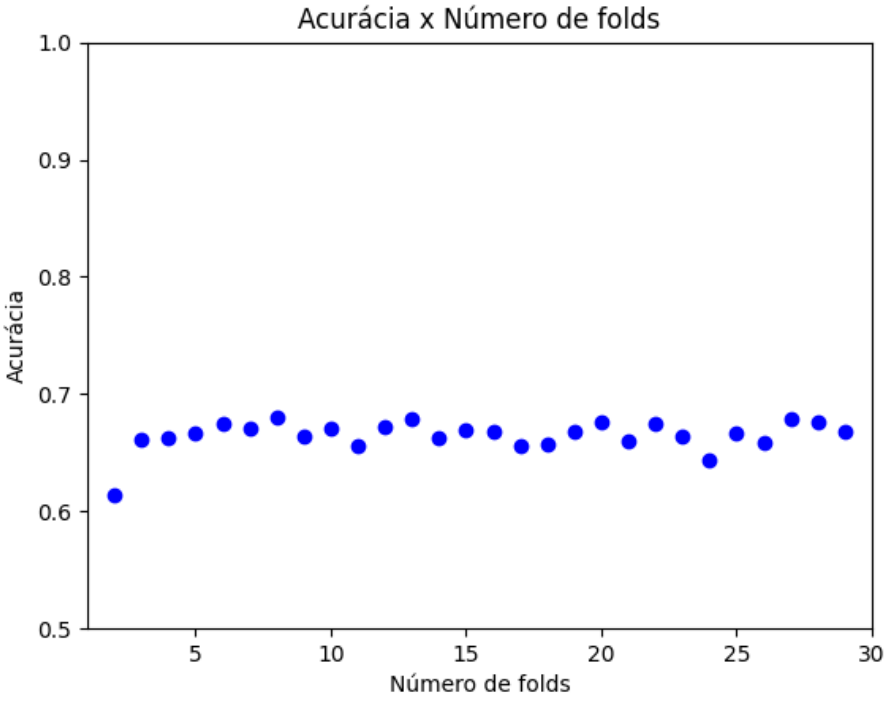
Agora, com o número de estimators fixado em 3 (que se provou um número razoável), testamos a eficiência de aumentar a profundidade da árvore.

```
      # Testando a acurácia em função da profundidade da árvore
      for tree_depth in range(1, 25):
         estimators = 3
         accuracy.append(k_fold_cross_validation(features, labels, n_folds,
estimators, tree_depth, 'tree'))
         plt.plot(tree_depth, accuracy[tree_depth - 1], 'bo')
      plt.xlabel('Profundidade da árvore')
      plt.ylabel('Acurácia')
      plt.title('Acurácia x Profundidade da árvore')
      plt.axis([0, 25, 0.5, 1])
      plt.show()
```

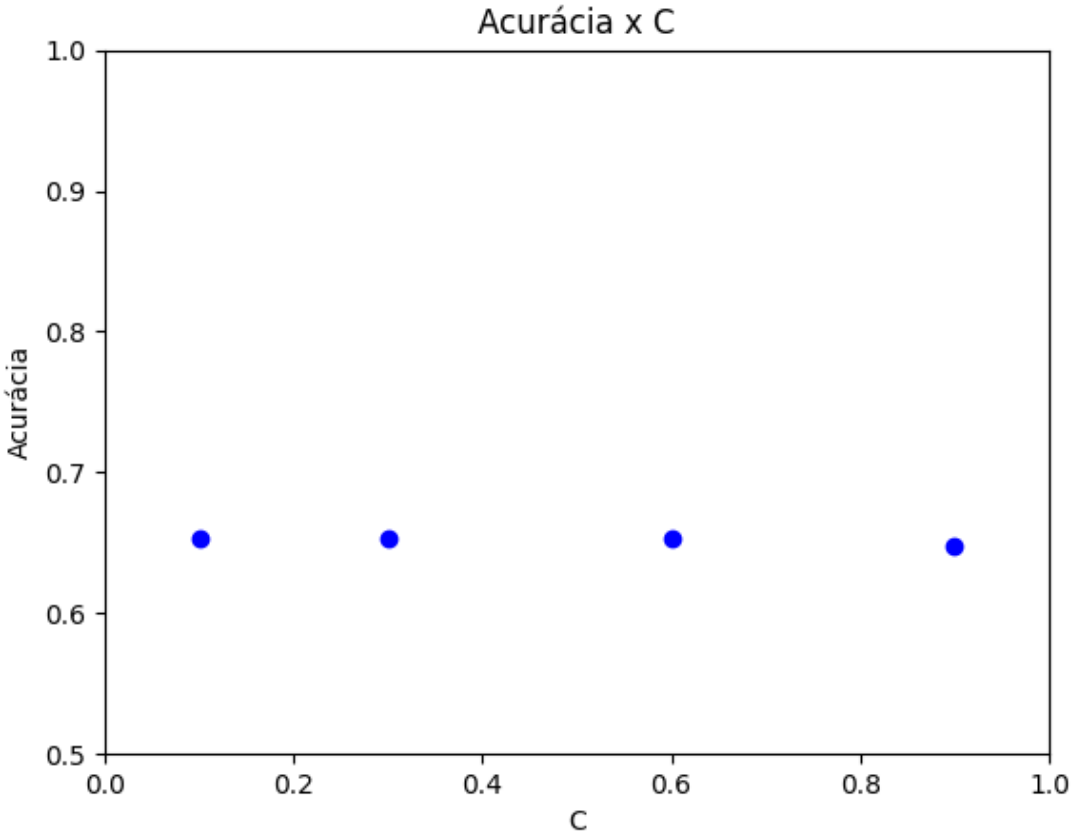
Aqui se torna interessante notar que aumentar demais a profundidade da árvore do estimador fraco reduz significativamente a acurácia do modelo. Isso ocorre pois o estimador começa a ser complexo demais e ser afetado fortemente por overfitting, que é justamente o que estamos tentando reduzir com boosting. Percebe-se que os valores mais efetivos foram 1 e 4.



No terceiro loop, a título de curiosidade, testaremos a acurácia em função do número de folds. Uma vez que realizei um shuffle nos dados antes do treinamento, aumentar o número de folds de 5 (o padrão que estamos testando, como especificado no trabalho) parece não fazer efeito razoável na acurácia. Isso pode acontecer por diversos motivos relacionados ao dado, como tamanho, sensibilidade a outliers e variabilidade.



Por fim, no último loop, variamos o C do SVM:



Percebe-se que a mudança do C nada influencia na acurácia do modelo. Isso pode ter ocorrido por alguns motivos diferentes:

1. O parâmetro C do SVM pode ter tido menos influência pelo fato dos pesos e as características das instâncias serem modificadas durante o treinamento dos estimadores fracos.
2. Como Boosting já realiza uma espécie de regularização, selecionando os estimadores mais fracos que contribuem para a combinação final, isso pode ter reduzido a necessidade de ajustar o parâmetro C para regularização adicional.
3. Os hiperparâmetros do boosting podem ter tido um impacto mais significativo na acurácia do modelo do que o parâmetro C do SVM. Portanto, alterações no parâmetro C podem ter sido ofuscadas pelo efeito dominante desses outros hiperparâmetros.