Trabalho Prático II

Igor Lacerda Faria da Silva

igorlfs@ufmg.br

Introdução

O objetivo deste trabalho é aprofundar o conhecimento do algoritmo de *boosting*, implementando-o "do zero". Mais especificamente, foi implementado o *AdaBoost* com *stumps* (árvores de decisão com apenas um nó). Como exemplo, foi analisado o *dataset* `Tic-Tac-Toe Endgame`, que contém todos as instâncias de "Jogo da Velha" em que o jogador inicial é x, além do resultado do jogo. No entanto, o programa é robusto: é possível usar outros bancos de dados apenas trocando alguns parâmetros.

Foi usada validação cruzada (*5-fold*) para avaliar o desempenho do modelo. Este relatório apresenta a evolução do erro variando-se a quantidade de *stumps*.

Desenvolvimento

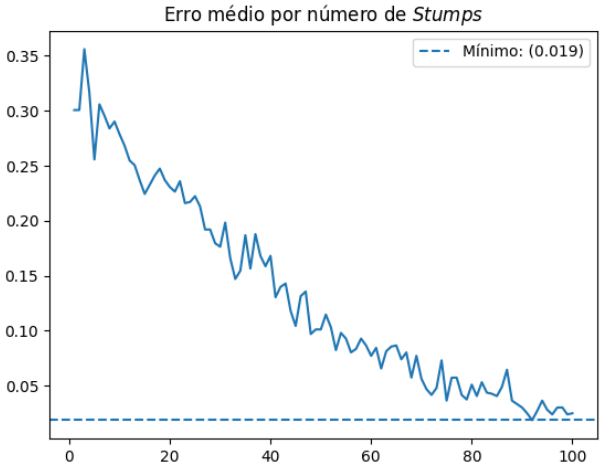

Figura 1: Média do erro pelo número de *stumps*, após a validação cruzada. O desempenho do modelo, de forma geral, foi satisfatório. Para até 100 stumps, o erro chegou a ser menor 2%.

Implementação

O trabalho foi implementado em *Python*. Foram usadas classes para representar os *stumps* e um objeto `Dataset`, que generaliza o programa. A cada `Stump` é associado: um vetor de indices onde o *stump* erra no treino (para facilitar o cálculo dos erros), um nome descritivo, uma *busca* para ser realizada no `Dataset` e um alfa, para se realizar as predições. Por outro lado, o `Dataset` contém uma *string* para representar o nome da coluna que contém as *labels*, os possíveis valores que x pode assumir, os valores "verdadeiro" e "falso" para a *label* do conjunto. O próprio *dataframe* é armazenado, assim como um vetor com as *labels* "ajustadas" para -1 e 1.

O principal método do programa realiza a validação cruzada, com auxílio do `KFold` do *sklearn*. Ele gera os 56 $(2*3*9+2)$[1] *stumps* para cada *split* com a função `gen_stumps`, que faz as buscas nos *dataframes* de treino e, principalmente, armazena os índices de erro. Em seguida, é realizado o *boosting* propriamente dito, na função `boosting`. A implementação teve como inspiração os *slides* da matéria e não fez nenhuma alteração relevante. Por fim, é calculado o erro, em `calculate_test_error`. Para fazer isso, foram usadas as *queries* para se obter o valor esperado e a comparação com este poder ser realizada.

Foram realizados testes simples com base nos exemplos dos *slides*, que se encontram no arquivo `src/test_util.py`. Eles podem ser rodados com o comando `pytest`. O arquivo principal (`src/app.py`) roda a função de validação cruzada e faz um gráfico da evolução dos erros, como o adicionado neste relatório.

[1] *label* (verdadeiro ou falso) * possíveis valores de x ('x','o','b') * número de posições + "chutar" tudo falso ou verdadeiro.