Trabalho Final: Algoritmo de Classificação

Michel Pires, Centro Federal de Educação Tecnológica de Minas Gerais July 16, 2024

Observações:

1. O trabalho deve ser realizado em equipes de até 6 alunos. Cada aluno deve ter uma ou mais funções específicas, as quais deverão ser detalhadas e explicadas quando questionadas pelo professor no dia da apresentação.
2. A entrega deve ser realizada por equipe, sendo necessário escolher um representante para essa tarefa. Considera-se uma entrega completa quando incluir uma documentação abrangente que detalhe o problema e a sua solução, as discussões sobre as estruturas de dados utilizadas e os motivos de sua escolha, além dos resultados de desempenho e execução coletados, bem como as conclusões da equipe ao finalizar a tarefa.
3. O trabalho entregue deve obrigatoriamente ser executável em sistema operacional Linux, sendo responsabilidade da equipe validar e corrigir possíveis erros de compilação e execução. A compilação deve seguir os padrões previamente estabelecidos na disciplina.
4. As linguagens de programação aceitas são C e C++.
5. A entrega deve ser realizada completamente via Git, com a documentação fornecida no formato README.md ou Wiki, ambos como parte do repositório apresentado.

A criação de um algoritmo de classificação utilizando listas, pilhas, filas e tabelas hash pode ser bem ilustrada com o algoritmo *Lazy Associative Classification* (LAC). O LAC é um método de classificação baseado em regras de associação que utiliza uma abordagem "preguiçosa" (i.e., *lazy*) para a classificação, realizada no momento da predição[1] e não como parte de um processo de treinamento antecipado. Durante o treinamento, o algoritmo cria apenas sua base de pesquisa, identificada como base de classes e base de características (*features*). Veja na descrição da Figura 1 um exemplo de como esse processo é realizado.

Considerando a abordagem da Figura 1, suponha que as entradas são vetores ou linhas em um arquivo CSV, representados, por exemplo, da seguinte forma: linha 1:[1, 2, 1, 0, 1, 4, 1], linha 2:[2, 2, 2, 5, 1, 1, 3], linha 3:[0, 0, 1, 1, 1, 2, 1], e assim por diante. De forma geral, o último número é utilizado para representar a classe da entrada; então, temos que a linha 1 e a linha 3 são da classe 1 e a linha 2 é da classe 3. Nesse padrão, também temos que a primeira coluna dessas linhas é representada pelos valores 1, 2 e 0, ou seja, (col1, 1), (col1, 2) e (col1, 0) ou simplesmente (1, 1), (1, 2) e (1, 0).

[1] Momento da entrada dos dados

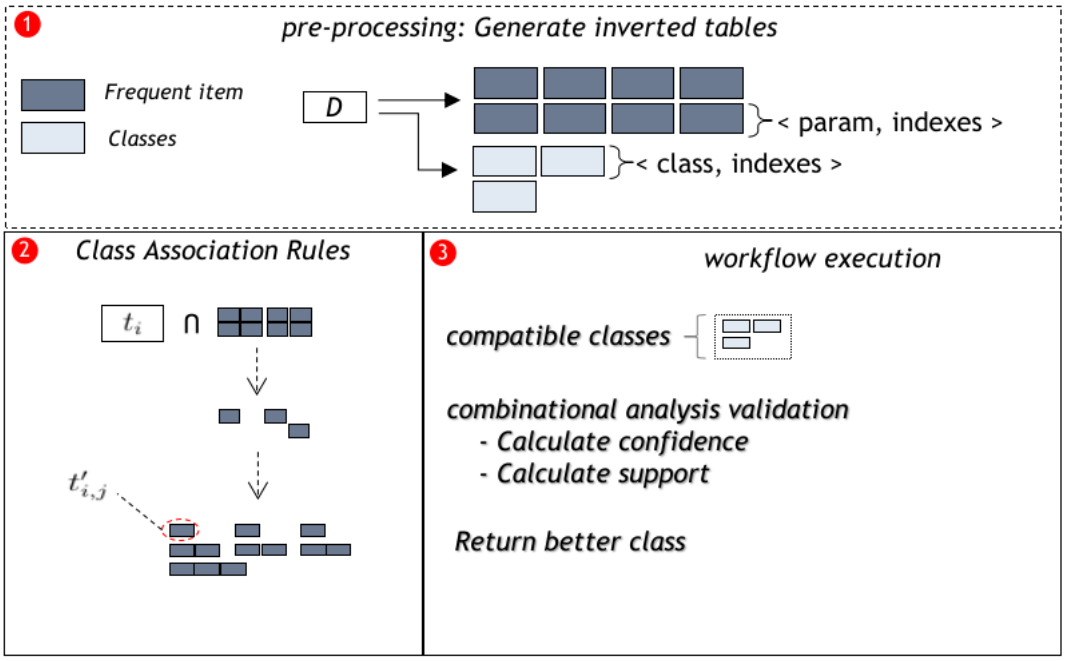

Figure 1: Visão geral do algoritmo *Lazy Associative Classification*. Em (1), os dados de treinamento são organizados nas chamadas tabelas invertidas. Nesse ponto, cada *feature* é indexada pelas linhas (i.e., instâncias de dados) nas quais aparece no conjunto de entradas. A ideia de indexação pelas linhas também é aplicada para associar as classes. Em (2), dada uma nova entrada (i.e., um vetor de *features* ou instância de dados), esta é consultada na tabela de *features* para determinar com quais linhas se relaciona. Uma vez identificadas as linhas de relação, uma etapa de análise combinatória, a partir das *features* identificadas, é iniciada em (3) para calcular as métricas de suporte e confiança dessas associações com todas as classes definidas pelo problema.

Para construir as tabelas invertidas (passo 1 da Figura 1), utilizamos o conceito de tuplas para indicar a coluna e a *feature* a ser mapeada. Assim, dada uma entrada de treinamento, esta deve ser primeiramente adaptada ao formato de tupla antes de ser processada. Veja um exemplo: considere como entrada o vetor [3, 3, 1, 1, 2, 3, 1], com o 1 da coluna 7 representando a classe. Essa entrada deve ser processada considerando o formato: [(c1, 3), (c2, 3), (c3, 1), (c4, 1), (c5, 2), (c6, 3), 1] ou simplesmente [(1, 3), (2, 3), (3, 1), (4, 1), (5, 2), (6, 3), 1]. Nesse sentido, utilizamos as *features* como chaves de uma tabela, enquanto a linha que as compõe é o valor. Já para a classe, utilizamos apenas o valor da mesma como chave e a linha para a composição da indexação. Ambos os contextos são definidos a partir de um modelo $< chave, valor >$. Veja na Figura 2 uma pequena representação de como criar as tabelas de *features* e classes na etapa de preparação do algoritmo.

Uma vez obtidas as tabelas invertidas, inicia-se a fase de classificação. Nessa fase, cada nova entrada, representada por um vetor de características sem a definição de classe, é avaliada por meio de um processo de análise combinatória. Primeiramente, cada elemento desse vetor é investigado na tabela de *features*. Como resultado, obtêm-se todas as características conhecidas pelo algoritmo, descartando aquelas que não são encontradas na tabela (vide

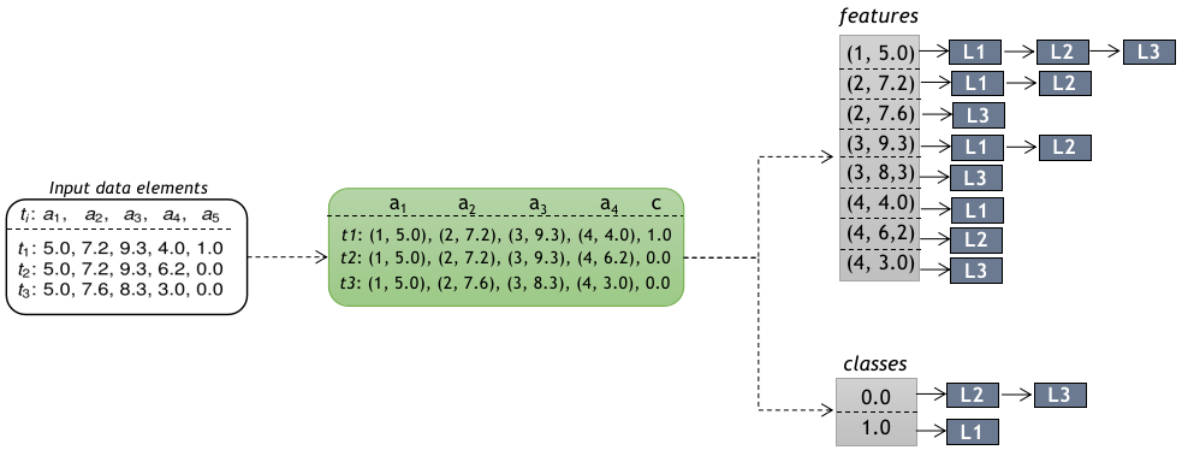

Figure 2: Visão geral da execução da fase de treinamento e/ou criação das tabelas invertidas para pesquisa e classificação.

etapa 2 da Figura 1). Em seguida, através de um processo combinatório das *features*, são avaliadas as linhas correspondentes. Nesse ponto, a avaliação começa para cada *feature* individualmente, depois em combinações de duas a duas, três a três e assim por diante, até que o tamanho da combinação seja igual ao tamanho da entrada. Observe a seguir o fluxo de execução para a classificação de uma entrada:

1. Obtenha as linhas relacionadas a cada *feature* da combinação realizada. Quando essa combinação tem tamanho 1, pegue as linhas diretamente da tabela. Caso contrário, execute a interseção das linhas das *features* da combinação para obter a lista de linhas onde todas estão presentes.

2. Com o vetor de linhas criado, inicie a avaliação do suporte e confiança.

3. Para calcular a confiança, pegue o número de linhas que a combinação apresenta e faça a interseção com as linhas de cada classe.

4. Se o valor da confiança for maior que zero, calcule o suporte.

5. Para calcular o suporte, divida o valor da confiança pelo número total de linhas na tabela de *features*. Armazene esse suporte, somando seu valor à relevância da classe c.

6. Repita os passos 1-5 até finalizar todas as combinações possíveis.

7. Após finalizar as combinações, ordene os valores de suporte em ordem decrescente e determine a classe que apresentou o maior valor de suporte. Essa classe será a classe definida para a entrada.

Considerando os passos acima apresentados, observe no exemplo gráfico da Figura 3 uma exemplificação desse processo de avaliação e classificação.

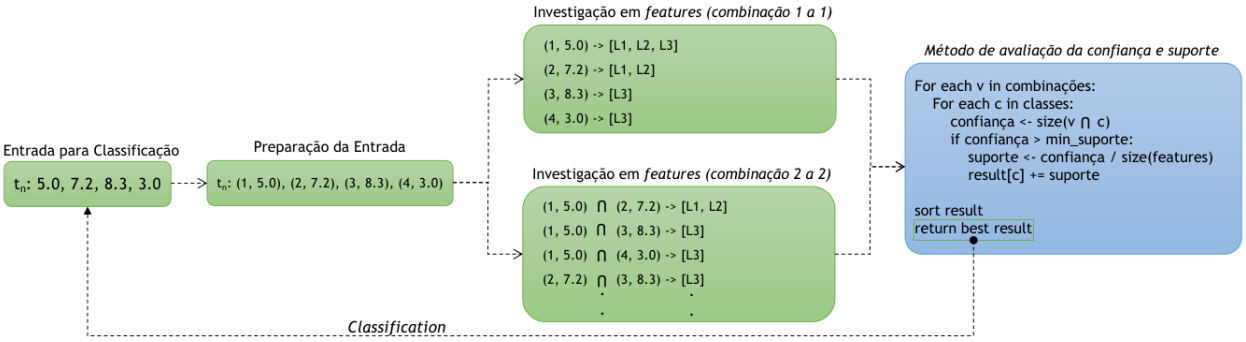

Figure 3: Visão geral do processo aplicado pelo LAC para classificar novas entradas de dados. Nesse processo, a execução do primeiro FOR é realizada para cada análise combinatória de forma separada, ou seja, realiza-se a análise para combinações uma a uma, depois duas a duas e assim por diante. Isso evita o estouro de memória, uma vez que o número de combinações pode ser considerável. Contudo, há maneiras de otimizar esse processo, utilizando sobreposição de cálculos.

1 Considerações a serem feitas

- O programa deverá ler dois arquivos: um chamado "treinamento" e outro chamado "teste". Para ambos os arquivos, a classe de cada linha será definida, mas para a realização dos testes, essa coluna deve ser desconsiderada na hora da leitura.
- O programa deve ler o arquivo de treinamento e definir em memória as tabelas apresentadas anteriormente, ou seja, a tabela de *features* e a tabela de classes.
- A leitura e execução do arquivo de testes deve ser realizada linha a linha. Nesse contexto, leia uma linha, faça todo o processamento de classificação para ela antes de ler a próxima.
- Como saída do programa, deve-se apresentar um arquivo chamado ‘output.txt‘. Nele, cada linha terá o número da linha e a classe atribuída a ela após a execução. Além disso, a última linha deve mostrar o número de acertos obtidos. Esse processo deve ser realizado da seguinte forma: (1) ao ler a linha a ser classificada, remova a classe e armazene-a para comparação; (2) ao término da execução, compute acertos e erros de classificação a partir da classe armazenada no início. Mostre, ao final, o número de erros e acertos alcançados pelo algoritmo, comumente chamados de *loss* e *accuracy*.

2 Documentação

A documentação a ser produzida deve conter, pelo menos, as seguintes partes:

- Um detalhamento mínimo que explique as fases de especificação, projeto e implementação. Nesta etapa, deve-se incluir uma ampla discussão sobre as estruturas utilizadas e o motivo de sua escolha.

- Para cada algoritmo adotado como parte da solução, uma descrição de seu projeto e análise de complexidade. Essa etapa deve ser realizada apenas para algoritmos já existentes e adotados como parte da solução apresentada.
- Para os arquivos utilizados para teste, uma descrição da saída esperada e as avaliações realizadas.
- Parte do ‘README.md‘ deve conter todas as instruções necessárias para a execução do trabalho com arquivos de entrada diferentes dos adotados durante os testes.
- O repositório Git deve conter todo o projeto, bem como uma descrição completa sobre sua execução, projeto e implementação. Deve-se observar também uma discussão sobre as estruturas adotadas e o motivo de tais decisões.

3 Considerações Finais

As equipes serão avaliadas em uma apresentação presencial realizada em sala de aula. Nesta apresentação, cada equipe deve expor todas as características adotadas no trabalho e os motivos que levaram ao projeto do algoritmo conforme exposto. Além disso, todas as equipes devem estar preparadas para a execução de seu algoritmo, a fim de coletar o tempo gasto na execução. A equipe que apresentar o algoritmo mais rápido para um volume de K entradas não precisará realizar a prova final. Nesse contexto, será considerado apenas o tempo gasto para processar o arquivo de testes apresentado como entrada, ou seja, o tempo máximo que o algoritmo apresentado gasta para classificar todas as entradas oferecidas.

Data das apresentações: 04 e 05 de setembro.
Data da entrega por todas as equipes: 03 de setembro até as 23h:59m.
Valor: 25 pontos.