**Desenvolvimento**

*Ninguém pensa que a qualidade do código é importante até que sejam 4 da manhã de um sábado, você tenha 18 horas depurando uma falha e ainda não tenha corrigido o bug.*

Ter uma prática de desenvolvimento inadequada para projetos de ML pode se manifestar de diversas maneiras, podendo matar completamente um projeto. Embora geralmente não seja tão diretamente visível como algumas das outras causas principais, ter uma base de código frágil e mal projetada e práticas de desenvolvimento inadequadas podem tornar um projeto mais difícil de trabalhar, mais fácil de quebrar na produção e muito mais difícil de melhorar com o passar do tempo. Por exemplo, vejamos uma situação de modificação bastante simples e frequente que surge durante o desenvolvimento de uma solução de modelagem: mudanças na engenharia de recursos. Na figura 1.9, vemos dois cientistas de dados tentando fazer um conjunto de alterações em uma base de código monolítica. Neste paradigma de desenvolvimento, toda a lógica de todo o trabalho é escrita em um único notebook por meio de declarações e funções de variáveis com script. Julie, na base de código monolítica, provavelmente terá muita pesquisa e rolagem para fazer, encontrando cada local individual onde o vetor de recursos está definido e adicionando seus novos campos às coleções. Seu trabalho de codificação precisará ser correto e realizado ao longo do script nos locais corretos também. É uma quantidade de trabalho assustadora para qualquer base de código de ML suficientemente complexa (já que o número de linhas de código para engenharia de recursos e modelagem combinadas pode chegar a milhares se desenvolvido em um paradigma de script) e está sujeito a erros frustrantes na forma de omissões, erros de digitação e outros erros de transcrição. Enquanto isso, Joe tem muito menos edições para fazer. Mas ele ainda está sujeito ao ato de pesquisar na longa base de código e confiar na edição correta dos valores codificados. O verdadeiro problema com a abordagem monolítica surge quando eles tentam incorporar cada uma de suas alterações em uma única cópia do script. Como possuem dependências mútuas do trabalho um do outro, ambos terão que atualizar seu código e selecionar uma de suas cópias para servir de mestre do projeto, copiando as alterações do trabalho do outro. Esse processo longo e árduo desperdiça um tempo precioso de desenvolvimento e provavelmente exigirá muita depuração para ser corrigido. A Figura 1.10 mostra uma abordagem diferente para manter a base de código de um projeto de ML. Desta vez, uma arquitetura de código modularizada separa o forte acoplamento que está presente no grande script da Figura 1.9. Esta base de código modularizada é escrita em um ambiente de desenvolvimento integrado (IDE). Embora as alterações feitas pelos dois DSs sejam de natureza idêntica àquelas feitas na figura 1.9 (Julie está adicionando alguns campos ao vetor de recursos e atualizando as codificações para esses novos campos, enquanto Joe está atualizando o escalonador usado no recurso vetor), a quantidade de esforço e tempo gasto para que essas mudanças funcionem em conjunto é dramaticamente diferente. Com uma base de código totalmente modularizada registrada no Git, cada um deles pode verificar uma ramificação de feature do master, fazer pequenas edições nos módulos que fazem parte de suas features, escrever novos testes (se necessário), executar seus testes e enviar uma solicitação pull. Depois que seu trabalho for concluído - devido ao código baseado em configuração e à capacidade dos métodos em cada classe de módulo de agir sobre os dados de seu projeto por meio do aproveitamento da configuração do trabalho - cada ramificação de recurso não afetará a outra e deverá funcionar apenas como projetado. Julie e Joe podem cortar um branch de lançamento de ambas as alterações em uma única compilação, executar um teste de integração completo e mesclar com segurança no master, confiantes de que seu trabalho está correto. Eles podem, na verdade, trabalhar juntos de forma eficiente na mesma base de código, minimizando enormemente a chance de erros e reduzindo a quantidade de tempo gasto na depuração do código.



