Desafio BK

Segue aqui, algumas anotações sobre os arquivos e a resolução do problema.

Escolha do modelo

Recentemente resolvi problemas similares à esse, e o modelo que usei que obtive melhores resultados foi o CatBoost(https://catboost.ai/), não me prendi muito na acurácia do modelo, mas se fosse para testar coisas novas para alguma possível melhora de desempenho, testaria LGBM e XGBoost. Essa biblioteca (Catboost) faz automaticamente alguns tratamentos que fiz de maneira manual, porém, acho que faz parte da avaliação o código de pré processamento.

Métricas

Para problemas de classificação binária como esses geralmente uso acurácia, e f1 score que consiste na média harmônica entre precisão e recall.

Usar somente a acurácia pode ser insuficiente. Caso o dataset esteja desbalanceado, a acurácia pode ficar alta porém por ter mais dados de uma categoria, o recall ou a precisão podem ficar baixos, por isso uso o f1_score também (não seria necessário nesse caso pois dataset foi equilibrado para o treino).

Guideline de código

Geralmente tento escrever meu código python baseado nas guidelines da PEP8 (https://www.python.org/dev/peps/pep-0008/) para evitar polêmicas do tipo “tal formatação fica melhor”, então, se eu escrevi algo diferente desse padrão (provavelmente sim) eu errei mesmo.

Estrutura dos arquivos:

-bk.py
-notebooks
—1.Data Discovery
—2.Preparando Código de data loading
—3.Model Training
-models
-data
-prod_code
—run_predictions.py

Notebooks

Aqui estão todos os arquivos usados para descobrir os dados e treinar o modelo. O primeiro arquivo desse diretório é “1. Data Discovery”, usei esse arquivo para olhar o dataset, e decidir quais colunas seriam usadas na predição, as respostas daquelas questões 1 e 2 do desafio estão ali dentro. O arquivo “2. Preparando o código de data loading..” basicamente foi onde eu escrevi e testei as funções que seriam posteriormente

usadas para carregar e tratar os dados tanto na etapa de treino quanto em produção, essas funções foram movidas para uma biblioteca chamada bk.py que estão na raiz do diretório. O arquivo “3. model training”, foi onde carreguei os dados, treinei e salvei o modelo. Em resumo, esse diretório tem os arquivos que compõem a etapa de solução do problema de predição.

Models

Esse diretório contém os modelos treinados e o scaler. Geralmente dentro de um projeto esses arquivos ficam no .gitignore para não deixar o repositório pesado, porém como não ficarei retreinando os modelos mantive no repositório mesmo.

Data

Esse diretório contém os dados que seriam usados no treino e teste do modelo. Assim como o conteúdo do “./models”, este diretório também deveria estar no .gitignore, mas por serem poucos dados decidi manter aqui mesmo.

Predictions

Aqui coloquei as predições do arquivo “./data/Abandono_teste.csv”.

Prod_code

Aqui tem o arquivo python que executa as predições do modelo para o arquivo passado por linha de comando. Esse arquivo basicamente executa uma função de poucas linhas, e a ideia disso é deixar fácil de colocar em produção em algum serviço de nuvem, por ex: cloud functions ou amazon lambda functions. As adaptações para fazer isso seriam: No lugar de buscar o modelo de um path local, buscar de um bucket da nuvem; Adaptar as funções de carregamento de dados para pegar os dados de um banco de dados diretamente ou de um upload de arquivo.

Para executar predições basta entrar no diretório ./prod_code/, e executar o seguinte comando: “python3 run_predictions.py ../data/Abandono_teste.csv /destino/destino.csv” o primeiro caminho é o caminho dos arquivos de teste para execução da predição, e o segundo caminho é o destino das predições.

Arquivos da raiz

bk.py

O Arquivo “bk.py”, é uma biblioteca que contém as principais funções usadas no pipeline do modelo. Ele pode parecer estranho e desnecessário já que é um código pequeno, porém ele trás varias vantagens. Irei citar algumas delas:

- Reusabilidade de código

- Certeza que o pré processamento dos dados terá o mesmo tratamento tanto no treino quanto na execução do modelo, já que eles usam a mesma função para fazer isso.
- Testável, caso o projeto tome maiores proporções, fica fácil de desenvolver testes automatizados para essas funções, já que estão separadas em um arquivo.

Aqui, caso o projeto crescesse, poderíamos separar as funções em mais arquivos, e achar uma solução melhor aos paths estáticos.

Fluxo dos dados