Algoritmos de aprendizagem de máquina com software em produção— análise de projetos da Lei Rouanet

LAPPIS

Feb 25 · 9 min read

Implementar sistemas com modelos de aprendizado de máquina é sempre um desafio, o trabalho com os dados é o coração da aplicação e deve ter cuidados especiais. Cada situação tem suas peculiaridades e, no projeto Salic-ML, não foi diferente. Com o escopo definido ao redor do mundo da lei Rouanet, o Salic-ML tem como objetivo agilizar processos pelos quais um projeto cultural deveria passar caso se candidatasse ao incentivo fiscal.

O uso de aprendizado de máquina e técnicas estatísticas nos dados disponibilizadas pelo Ministério da Cultura permitia calcular métricas que serviam como indicadores de possíveis anormalidades de um projeto, aumentando a rapidez na busca

de eventuais anomalias. A partir desta situação, surgiu o projeto Salic-ML, nosso objeto de estudo. O objetivo seria criar uma plataforma na qual, a partir da análise dos dados e por meio de métricas de cada projeto cultural, indicasse, aos usuários, um *score* que indicava o quão complexo seria a análise do ponto de vista financeiro.

Tal projeto foi financiado pelo Ministério da Cultura em parceria com o LAPPIS, e a aplicação será integrada ao SALIC.

O projeto completo está sendo desenvolvido como software livre, e pode ser acessado em Salic-ML

Ciência de dados

Assim como grande parte das equipes envolvidas na área, Jupyter Notebooks foram utilizados para o trabalho de ciência de dados. O versionamento era feito por meio do Git e toda a organização das tarefas era feita por issues. Para o estudo dos dados, foram planejadas e executadas

1 — Entendimento do negócio

Pode parecer óbvio, mas é estritamente necessário ter uma boa visão do negócio em que se deseja aplicar ciência de dados. No caso do Salic-ML, foram necessários diversos encontros e reuniões presenciais, tanto entre o próprio time do Salic-ML e entre clientes e usuários finais, além de estudos por fora sobre assuntos do negócio como Leis de Incentivo Cultural (onde entra a Lei Rouanet) e o SALIC (Sistema de Apoio às Leis de

Incentivo à Cultura). Aplicar ciência de dados sobre um negócio no qual não se tem um bom entendimento é um grande risco em termos de tempo e esforço do time, portanto achamos importante deixar claro esse ponto óbvio, mas essencial.

2 — Mergulhando nos dados — Mineração de Dados

Ok, entendemos o negócio e entendemos o produto que queremos construir. O próximo passo, agora, é começar o processo de Mineração de Dados. O objetivo agora é descobrir padrões e ter *insights* sobre os dados disponíveis. Portanto vamos quebrar o processo de mergulhar nos dados em 4 passos:

Passo 1: Identificando as fontes de informação

Não adianta nada ter uma ideia incrível de produto e um algoritmo perfeito se não há dados disponíveis. Este passo é sobre descobrir quais dados temos acesso. No caso do Salic-ML, a única fonte sistemática de dados era o banco de dados do Salic, vamos chamá-lo de Salic-DB. Portanto, toda a informação a nossa disposição se dá num banco de dados relacional por meio de tabelas e colunas.

Passo 2: Selecionando dados relevantes

Iniciamos aqui um processo `ad-hoc` para descobrir tabelas e colunas relevantes neste banco de dados. Não havia uma documentação abrangente e, sozinhos, tivemos de descobrir como fatos do mundo do negócio se traduziam em tabelas e colunas e como o contrário também acontecia. Neste passo,

também verificamos quais tabelas tinham poucos dados ou dados inconsistentes, desatualizados.

Passo 3: Obtendo dados de forma sistemática

Neste passo o objetivo é trazer dados que estão guardados dentro de um banco de dados para dentro de um *Python Notebook*. Utilizamos duas formas para cumprir esta tarefa:

.csv: Utilizando a ferramenta de banco de dados DBeaver, nós executamos queries `SQL` sobre o banco de dados e fizemos o download destes dados num formato `.csv`. Estes arquivos `.csv` foram utilizados posteriormente como datasets de entrada para a pesquisa do time e para treinamento de algoritmos de Machine Learning

Open Database Connectivity (ODBC): Utilizamos o pacote python pyodbc como ferramenta de ODBC no Python. Assim, nós tínhams uma forma de se conectar direto com banco de dados dentro da nossa aplicação Python.

Passo 4: Descobrindo padrões insights sobre os dados

Com os dados em mãos agora é o objetivo é identicar padrões nos dados e termos insights sobre o conjunto de dados. Neste passo nós buscávamos validar ou invalidar hipóteses nossas ou simplesmente explorar um trecho dos dados através de gráficos e estatísticas simples.

Tecnologia	Propósito
Python3	Linguagem de programação utilizada para propósito geral do projeto.
jupyter-notebook	Utilizado para juntar código python + textos em markdown para comentar, explicar e discutir resultados. Também é uma forma acessível de disponibilizar código, figuras e textos no github, sem a necessidade de executar o projeto.
matplotlib	Utilizado para gerar e manipular gráficos.
scikit-learn	Implementações de algoritmos clássicos de machine learning.
pandas	Biblioteca utilizada para manipulação de tabelas.
numpy	Utilizado como uma forma de calcular estatísticas simples (média, desvio padrão) e representar dados matemáticos de forma eficiente.

Tabela 1: tecnologias envolvidas no processo de descobrimento de padrões e insights sobre os dados

O processo desenvolvido nos *jupyter-notebook* definem uma etapa fundamental do Salic-ML: as hipóteses validadas ou insights exploradas nestes cadernos são posteriormente transformadas em códigos que vão para a produção.

Arquitetura

O desafio do produto

Grande parte do tempo de projeto foi dedicado aos estudos sobre os dados disponíveis. Entretanto, toda a análise tinha, como objetivo final, um produto que pudesse auxiliar os técnicos do MinC a realizarem o seu trabalho de uma forma mais otimizada. Para isso, iniciou-se o trabalho da equipe de produto.

Por conta do tempo e do escopo definido, a arquitetura inicial envolvia apenas uma aplicação, com fundação no *framework*

web Python Django. A chamamos de salic-ml-web. Nela, seria possível navegar entre os diversos projetos disponíveis na base de dados e visualizar uma nota global de cada um deles, calculada a partir dos algoritmos implementados pela equipe de *data science*.

Para viabilizar este comportamento, era necessário integrar os algoritmos dos *Python Notebooks* na aplicação *Python Django*. A solução encontrada foi transformar o código dos estudos em um pacote *Python*, disponibilizado no PyPi e consumido no *salic-ml-web*.

A divisão da equipe em duas frentes ("Equipe ML" e "Equipe ML Produto") foi inspirada na abordagem de **divisão de serviços por subdomínio** (bem utilizada em arquiteturas de microserviços). O pacote *Python* continha os algoritmos resultantes dos estudos sobre os dados e a aplicação *Django* tinha maior preocupação em como estes algoritmos seriam entregues como produto ao Ministério.

Como resultado desta divisão interna, foi planejado o seguinte fluxo de trabalho:

1. "Equipe ML" implementava, no *salic-ml*, o código do algoritmo necessário para calcular uma determinada métrica.
2. "Equipe ML Produto" preocupava-se em:
1. Atualizar, no *salic-ml-web*, a versão utilizada do *salic-ml* para permitir o uso da nova *feature* implementada.
3. Mostrar os resultados da métrica em um *front-end* acessível

pelos técnicos do MinC.

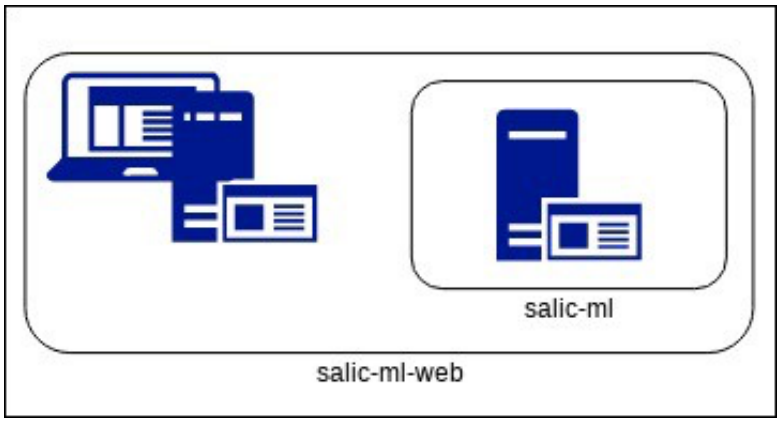

Imagem 1: arquitetura inicial, com frontend e o pacote acoplados no salic-ml-web

As vantagens deste modelo foram:

- Haviam duas equipes e criou-se dois “serviços”. Cada equipe ficou responsável por uma base de código.
- Não observou-se mudanças drásticas das responsabilidades de cada equipe, já que o time de ciência de dados permaneceu preocupado em criar novos algoritmos e a equipe de produto preocupava-se com outros aspectos (*DevOps*, *frontend*, …).

Entretanto, haviam desvantagens:

- O processo era cascateado: antes, a equipe de *data science* deveria implementar o algoritmo no *salic-ml* para uso posterior no *salic-ml-web*.
- A separação das equipes gera, naturalmente, eventuais falhas de comunicação. Alterações de última hora eram feitas de forma constante e isto gerava desgaste dos membros.

Para tentar amenizar estes problemas, técnicas de DevOps

foram utilizadas para otimizar o tempo de desenvolvimento de ambos os projetos. Integração e *deploy* contínuos permitiam encontrar problemas mais cedo e, consequentemente, corrigí-los o quanto antes.

Além dos pontos mencionados, havia uma preocupação em relação à integração com o SALIC, já implementado no Ministério. Para isso, em um determinado momento, trocou-se o conceito e a arquitetura interna do *salic-ml-web* para que ele se tornasse uma API baseada no *Django Rest Framework* e que o *frontend* fosse apenas um protótipo de homologação das informações, feito em *VueJS* (salic-ml-frontend). Esperava-se que, em algum momento, as informações resultantes dos algoritmos fossem repassadas para o SALIC e que o *frontend* fosse, no máximo, reutilizado na integração.

Imagem 2: arquitetura após primeira modificação, com frontend separado e o pacote acoplado no salic-ml-web

Tabela 2: tecnologias envolvidas na aplicação salic-ml-web

O desafio dos dados

Dada a implementação do pacote *salic-ml*, que espelhava o que foi feito nos *Python Notebooks*, era necessário alimentar os algoritmos com os mesmos tipos de *input* utilizados na fase de *data science*. Para isso, foi necessário mapear um *Docker Volume*, por meio do *Docker Compose*, para a pasta na qual o pacote era instalado dentro do contâiner. Nesta pasta, seriam inseridos os arquivos (.csv) contendo os dados exigidos pelos algoritmos.

Observou-se os seguintes problemas:

- Os .csv's eram significativamente grandes. Movimentá-los

exigia um certo tempo e nem sempre era garantido que funcionariam na primeira tentativa.
- Instanciar o objeto que continha os algoritmos custava alguns minutos, o que causava dificuldades no processo de desenvolvimento do produto. Cada modificação mínima no *frontend* resultava no recarregamento da aplicação e, consequentemente, na instanciação deste objeto.

Para tentar amenizar o segundo problema, além de serializar o objeto em um *Pickle File*, houve uma mudança arquitetural que removeria a necessidade do *salic-ml-web* instanciar o objeto que calculava os algoritmos. Mantendo a ideia do serviço único por equipe, criou-se o *lappis-learning*, uma aplicação *Python Flask* que implementava os algoritmos resultantes da análise dos dados e disponibilizava os resultados, em formato de JSON, por meio de uma API. Desta maneira, o objeto que leva mais tempo para ser instanciado fica completamente desacoplado da aplicação que exige uma maior cadência de modificações, facilitando o desenvolvimento.

Imagem 3: arquitetura resultante após a criação do lappis-learning

Aproveitando a modificação arquitetural, optou-se por eliminar o uso dos *.csv's*. A aplicação *lappis-learning* obtém os dados diretamente do banco de dados, eliminando a necessidade de se manter os arquivos que os continham.

Como o estágio de implementação do pacote *salic-ml* era avançado, a estratégia de transição envolvia implementar os novos algoritmos diretamente na nova arquitetura e, aos poucos, iríamos transferindo os já existentes.

Assim, era possível manter uma arquitetura híbrida. Alguns resultados eram obtidos do pacote *salic-ml*, que retornava *Python Dicts*, e outros do serviço *lappis-learning*, que retornava *JSONs*.

Como pontos de melhoria, é possível observar:

- É um verdadeiro malabarismo técnico manter a arquitetura híbrida do ponto de vista de fonte de dados. O cenário ideal seria a finalização da transição do consumo do *salic-ml* para o consumo exclusivo do *lappis-learning*.
- Mesmo com a separação do serviço exclusivo de análise de dados, a arquitetura interna deveria possibilitar um cache dos metadados que são calculados toda vez que se inicia o serviço

lappis-learning.
- A integração, até o momento no qual este texto está sendo escrito, não foi realizada.

Apesar das possíveis melhorias, também percebemos acertos:

- A separação dos serviços entre as equipes, que beneficiou o desenvolvimento como um todo. Haver responsabilidades exclusivas de cada equipe possibilitou que cada uma delas pudesse estar mais imersa em seus desafios e, consequentemente, apresentar melhores resultados.
- As otimizações internas, como o uso do *pickle* para agilizar a instanciação do objeto que continha os algoritmos.
- A transferência de conhecimento entre os membros foi constante. Seja sobre o contexto das leis de incentivo à cultura ou sobre questões técnicas, percebe-se que os membros da equipe (tanto do ML quanto do ML Produto) tiveram crescimento no conhecimento obtido, o que é relevante dado o contexto acadêmico no qual o projeto se inseria.

Do ponto de vista arquitetural, apresentamos uma solução que viabiliza o uso de algoritmos, tanto estatísticos quanto de *machine learning*, em uma aplicação *web*. Já do ponto de vista de produto, conseguimos construir uma plataforma na qual é possível enxergar anomalias em propostas de projetos submetidas ao Ministério da Cultura. O resultado, dentro do escopo e do contexto presentes na época, foi relevante e apresentou-se de forma madura aos clientes.