Criando uma API REST para modelos de aprendizagem de máquina

Déborah Mesquita

TDC Recife 2019

Sobre o que vamos conversar

- Aprendizagem de máquina do ponto de vista do desenvolvimento de software
- A API do scikit-learn e como o Pipeline pode nos ajudar a organizar o código dos sistemas
- Etapas e exemplo de como colocar um modelo em produção

Quem sou eu

- Cientista de Dados, eye4fraud
- Generalista
- Vencedora do Machine Learning Award da Microsoft Imagine em 2016
- Technical Writer

Por que estou aqui falando sobre isso hoje

- Trabalhava mais com provas de conceito para fugir da complexidade do deploy
- Só que agora não tá dando pra fugir mais

O que vamos construir

- "Será que em palestras anteriores alguém já abordou esse tema?"
- Seria legal ver palestras relacionadas a uma determinada palestra
- **Vamos criar um sistema para fazer isso com as palestras do TDC**

Etapas

1. Extrair os dados
2. Análise Exploratória
3. Treinar modelo
4. Deploy

Extrair os dados

- Scrapy
- Dados de 2014 a 2019 dos TDC de SP, BH, Porto Alegre e Floripa

```
import scrapy

  with open("trilhas.txt", "r") as file:
      start_urls = eval(file.readline())

  def parse(self, response):
      ano = url_words[4]
      cidade = url_words[5]
      trilha = url_words[-1]
```

Análise exploratória

- *sem tempo irmao*
- O scraping **não conseguiu** capturar descrições de cerca de 6.000 palestras
- Usando apenas palestras com título e descrição e removendo duplicados: 3.229 palestras

Treinar modelo

- Similaridade de cosenos
- TfidfVectorizer do scikit-learn: converte documentos em uma matriz de features TF-IDF

Treinar modelo

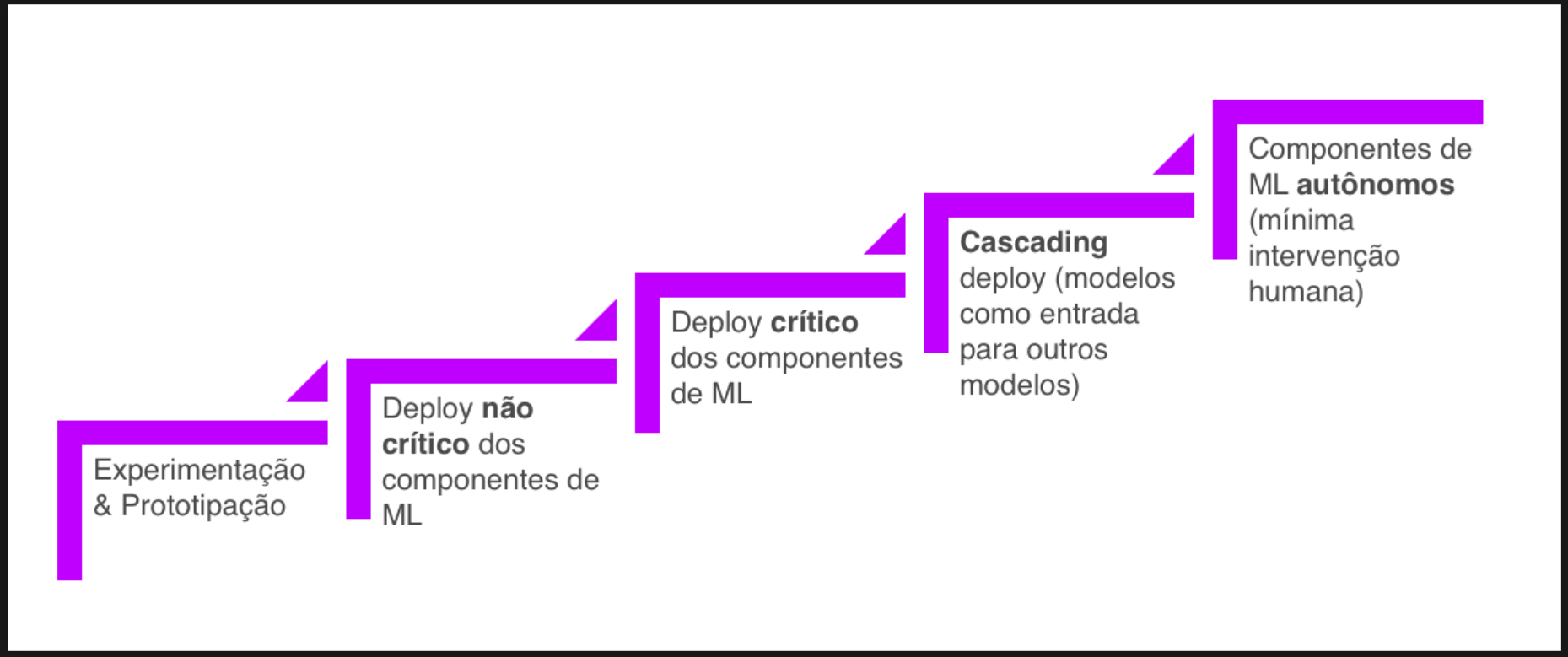
- Pipeline dos dados
  1. Juntar título e descrição
  2. Extrair o lemma de cada palavra e remover stopwords
  3. Criar matriz com o TfidfVectorizer
  4. Computar similaridade de cosenos

Treinar modelo

Jupyter notebook

Partindo do notebook para o código em produção

Taxonomia dos desafios de Eng. de Software para sistemas de ML



Evolução do uso de ML em ambientes de software comerciais, adaptado de [1]

Partindo do notebook para código em produção

- Criar aplicações em aprendizagem de maquina traz dificuldades além das encontradas em outros domínios de desenvolvimento de software
- A Microsoft identificou três dificuldades principais [2]

Dificuldades de Eng. de Software para sistemas de ML

1. Descobrir, disponibilizar, gerenciar e versionar **dados** é inerentemente mais complexo e diferente do que fazer o mesmo com código

Dificuldades de Eng. de Software para sistemas de ML

2. Construir e reutilizar modelos requer conhecimento de ML específico e que muitos engenheiros de software não possuem

Dificuldades de Eng. de Software para sistemas de ML

3. Geralmente é mais difícil manter limites entre módulos de ML do que entre componentes de engenharia de software

Partindo do notebook para código em produção

- Uma das características mais importantes e diferenciais é que estamos lidando com inteligência
- Isso significa que os modelos podem **cometer erros**

Como a API do scikit-learn vem para nos ajudar

- Scikit-learn: implementações do estado da arte de vários algoritmos famosos de machine learning
- Também inclui ferramentas para avaliar e selecionar modelos e procedimentos para pré-processar dados

Como a API do scikit-learn vem para nos ajudar

- API consistente e extensível, onde podemos **definir estimators e transformers customizáveis**, produzindo código de leitura fácil

A API do scikit-learn

- estimator: interface para construir e treinar modelos
- predictor: interface para realizar predições
- transformer: interface para transformar dados

Estimators

- Estimator: interface para construir e treinar modelos, expõe o método `fit()`
- Encapsula o modelo e o estimator para reduzir a complexidade [3]

```
from sklearn.ensemble import RandomForestClassifier()

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

Estimators

- Os algoritmos clássicos de ML não são os únicos implementados como estimators
- Rotinas de pré-processamento (como a vetorização de documentos por exemplo) também implementam a interface
- Com isso o código fica consistente e **compor as pipelines se torna mais fácil**

Predictors

- Predictor: interface para realizar predições, expõe o método `predict()`
- Recebe um array X_test e retornar as predições

```
y_pred = clf.predict(X_test)
```

- Usado por estimators supervisionados e por alguns não-supervisionados também

Predictors

```
from sklearn.cluster import KMeans

km = KMeans(nclusters=10)
km.fit(X_train)
clust_pred = mk.predict(X_test)
```

Transformers

- Transformer: interface para transformar dados, expõe o método `transform()`
- Recebe um array X_rest e retorna uma versão transformada de X_test
- Muitas vezes é preciso salvar estado do treinamento para transformar o dado em que será feita a predição, nesse caso podemos usar o método `fit()` do transformer

API avançada: pipelines

- Pipelines: encadeiam vários estimators em um só
- Útil porque o workflow de ML tipicamente envolve uma sequência fixa de etapas de processamento e algumas transformações necessitam de um tipo de treinamento

API avançada: pipelines

- Cada step da pipeline precisa implementar a interface transform, e o ultimo step pode ser um predictor, um transform ou os dois
- `pipeline.fit()`: chama o `fit()`, depois o `transform()` e passa para o próximo step

Custom transformer

- O scikit-learn fornece vários transformers, mas muitas vezes precisamos de transformers customizados
- Como os objetos no scikit-learn são definidos por interface, basta implementar os métodos do transformer para obter um novo transformer
- Os custom transformers podem ser utilizados nas pipelines, o que facilita muito nossa vida

Criando custom transformers e custom estimators

- Se precisamos dos dados de treinamento para transformar dados durante a predição, fazemos isso no `fit()`; transformações estáticas são implementadas no método `transform()`
- Se vamos precisar de um outro objeto para realizar as transformações, passamos ele como parâmetro no `__init__()`
- Se vamos precisar de resultados de predição. implementamos o método `predict()`

Aplicando tudo isso no projeto

Transformer para lemmas das palavras

- Estático (independe dos dados de treinamento)
- Precisamos do modelo pt_br do spacy

```
class LemmatizationTransformer(BaseEstimator,
                               TransformerMixin):
  def __init__(self, nlp=None) -> None:
      self.nlp = nlp

  def transform(self, X: list) -> list:
      X_output = []
      X_docs = list(self.nlp.pipe(X,
                        disable=["parser","ner"]))
      for doc in X_docs:
          tokens = [t.lemma_ for t in doc if ...]
          clean_doc = " ".join(tokens)
          X_output.append(clean_doc)
      return X_output
```

Predictor para as similaridades

- O `cosine_similarities` gera uma matriz de scores entre os vetores passados como parâmetros
- Como os vetores que vamos comparar (palestras extraídas das páginas) não mudam quando precisamos fazer uma predição, vamos salvar essa matriz no método `fit()`

Predictor para as similaridades

```
class CosineSimilaritiesEstimator(BaseEstimator,
                                  ClusterMixin):
    def fit(self, X, y=None) -> None:
        self.X_corpus = X
        return self

    def predict(self, X: 'sparse matrix') -> pd.Series:
        cosine_similarities = pd.Series(cosine_similarity(
                                        self.X_corpus, X
                                        ).flatten())
        return cosine_similarities
```

!!! ESCREVA TESTES !!!

ESCREVA TESTES

```
class TransformersTest(unittest.TestCase):
    def test_lemmatization(self):
        nlp = spacy.load("pt_core_news_sm")
        lt = LemmatizationTransformer(nlp)

        self.assertListEqual(["jogar"],
                             lt.fit_transform(["eu jogo"]))

        self.assertListEqual(["jogar"],
                             lt.transform(["eu quero jogar"]))

        self.assertListEqual(["gostar jogar"],
                             lt.transform(["eu gosto de jogar"]))
```

Montando a pipeline

Montando a pipeline

```
data_process_pipeline = Pipeline(
    [
        ("get_lemmas", LemmatizationTransformer(nlp)),
        ("tfidf", TfidfVectorizer())
    ]
)

prediction_pipeline = Pipeline(
    [
        ("data_process", data_process_pipeline),
        ("estimator", CosineSimilaritiesEstimator())
    ]
)
```

Encapsulando o modelo

Encapsulando o modelo

```
class Model(object):
    def __init__(self):
        self.pipeline = prediction_pipeline
        self.version = version

    def train(self, documents: pd.DataFrame) -> None:
        self.documents = documents
        X_train = self._get_all_text_fields(documents)

        self.pipeline.fit(X_train)
```

Encapsulando o modelo

```
    def predict(self, X: dict, n_results: int) -> list:
        similarity_scores = self.pipeline.predict(X)
        similar_documents = similarity_scores
                                        .nlargest(n_results)
        results = []
        for i,score in similar_documents.iteritems():
            item = self.documents.iloc[i].to_dict()
            [...]
            results.append(item)
        output = {"predictions":results,
                                "version":self.version}

        return output
```

Encapsulando o modelo

```
def serialize(self, fname):
    with open(fname, "wb") as f:
        dill.dump(self.pipeline, f)
        dill.dump(self.documents, f)

@staticmethod
def deserialize(fname):
    model = Model()
    with open(fname, "rb") as f:
        model.pipeline = dill.load(f)
        model.documents = dill.load(f)

        return model
```

Criando a API

Criando a API

- flask_restful

```
from flask import Flask, jsonify
from flask_restful import Resource, Api, reqparse
from text_similarity_model.model import Model

app = Flask(__name__)
api = Api(app)

model = Model.deserialize("../models/model_1.0.0.pkl")
```

Criando a API

```
class Predict(Resource):
    def post(self):
        title = args["title"]
        description = args["description"]
        n_results = 3
        data_to_predict = {"title":title,
                           "description":description}

        palestras = model.predict(X=data_to_predict,
                                  n_results=n_results)
        return palestras

api.add_resource(Predict, "/predict")
```

Criando a API

- Deploy em um VPS (gunicorn) [4]

```
$ sudo ufw allow 5000
$ gunicorn --bind 0.0.0.0:5000 wsgi:app
```

Consumindo a API

- http://45.179.88.108:5000/version
- http://45.179.88.108:5000/predict

{"titulo":"Criando e fazendo deploy de uma API REST para model

Considerações finais

- Sistemas de machine learning são difíceis de manter
- Usar pipelines facilita bastante a criação de código compreensível e reutilizável
- Ferramenta do scikit que não utilizamos e que é muito útil também: model selection (GridSearchCV e RandomizedSearchCV)

Do machine learning like the great engineer you are, not like the great machine learning expert you aren't [5]

Referências

1. Lwakatare, Lucy Ellen, et al. "A taxonomy of software engin
2. Amershi, Saleema, et al. "Software engineering for machine
3. Buitinck, Lars, et al. "API design for machine learning sof
4. https://www.digitalocean.com/community/tutorials/how-to-ser
5. https://developers.google.com/machine-learning/guides/rules
6. https://towardsdatascience.com/elmo-contextual-language-emb
7. https://realpython.com/kickstarting-flask-on-ubuntu-setup-a
8. https://drivendata.github.io/cookiecutter-data-science/

Obrigada!

- Repo: https://github.com/dmesquita/similaridades-palestras
- deborahmesquita.com