Aula 5 - Software para gerenciamento de projetos de ciência de dados

Jayme Anchante

22 de junho de 2020

Software para gerenciamento de projetos de ciência de dados

Stack inicial para ciência de dados

Python

- pandas: processamento de dados
- scikit-learn: aprendizado de máquina
- matplotlib: visualização
- jupyter: interface gráfica para programação

R

- dplyr: processamento de dados
- caret: aprendizado de máquina
- ggplot2: visualização
- jupyter/RStudio: interface gráfica para programação

Fluxo usual: OSEMN

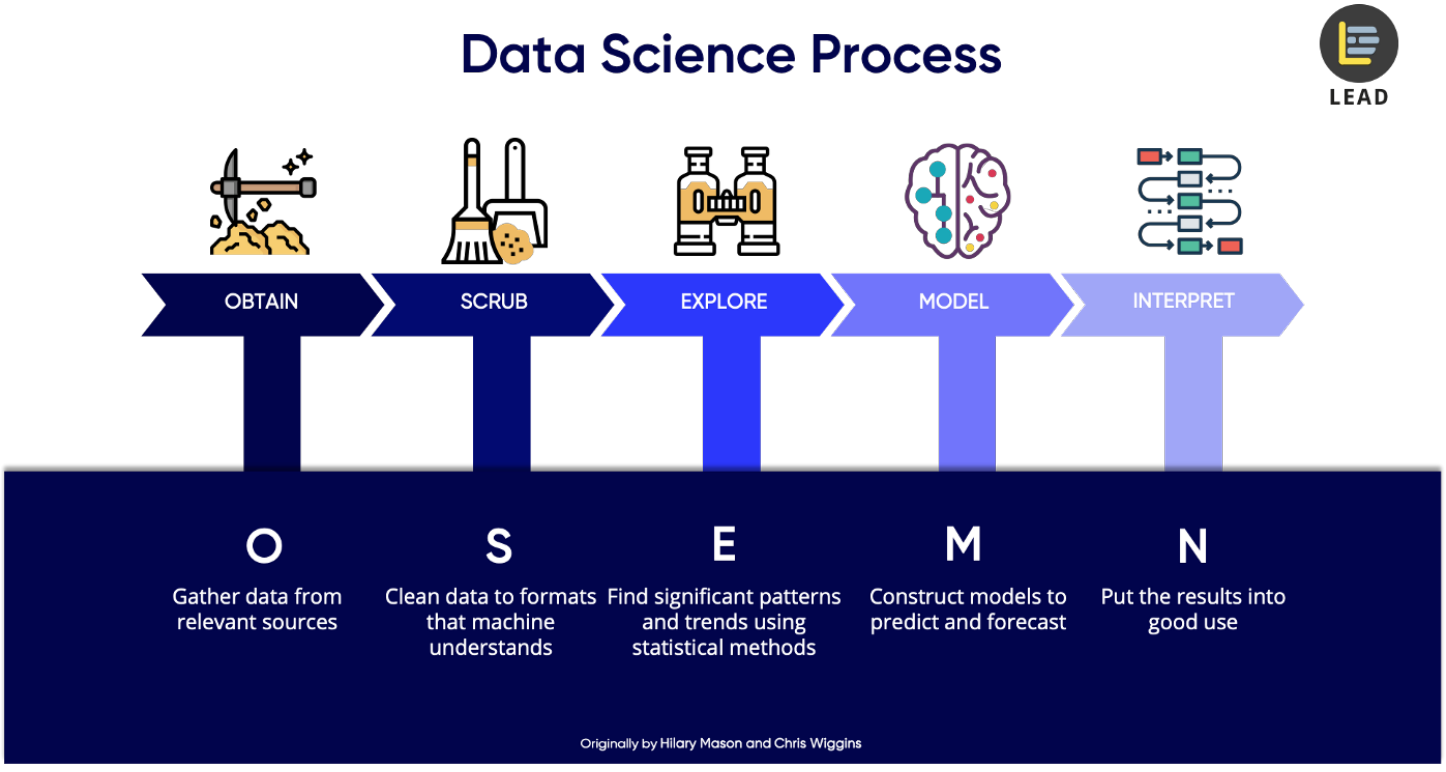


Figure 1: Paradigma OSEMN

Boas práticas

Lista de boas práticas

- Git: utilize versionamento de código
- Cloud: utilize um provedor de nuvem para seu ambiente de produção (aquele em que as coisas vão rodar no dia a dia), e se possível, faça desenvolvimento na nuvem
- CI/CD: faça integrações contínuas (forma automatizada de construir, empacotar e testar código) e implantação contínua (coloca seu programa numa infraestrutura de produção)
- Testes: faça testes de seu código para sua melhor manutenção, o seu "eu do futuro" irá agradecer
- **Cookiecutter para ciência de dados**
- **Docker**

Cookiecutter para ciência de dados

"Cookie cutter" é uma expressão em inglês que significa "mesmice" ou "falta de originalidade"

Cookiecutter para ciência de dados é uma estrutura de projeto lógica, razoavelmente padronizada, e flexível para fazer e compartilhar projetos de ciência de dados

Ciência de dados, ciência é reproduzível por definição. Reproducibilidade é muito importante em projetos de programação e ciência de dados. Muitas vezes encontramos repositórios com um ou vários arquivos tipo **jupyter** sem nenhuma menção a linguagem utilizada (projeto jupyter suporta mais de 50 linguagens), como fazer a instalação do ambiente para rodar o *notebook*/caderno e outras instruções

Qualidade de organização e de código também é importante, ninguém quer se deparar com arquivos confusos e conflitantes como "make_figures.py.old", "make_figures_working.py" ou "new_make_figures01.py"

Cookiecutter para ciência de dados - estrutura início

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
```

Figure 2: Estrutura

Cookiecutter para ciência de dados - estrutura fim

```
.
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- Make this project pip installable with `pip install -e`
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.testrun.org
```

Figure 3: Estrutura

Docker

Ferramenta de virtualização a nível do sistema operacional para construção, empacotamento e compartilhamento de software como aplicações e microsserviços.

Problemas que resolve:

- Fazer implantações automatizadas contínuas (CD)
- Compartilhar código com pessoas que utilizam outros sistemas operacionais
- Utilizar software sem modificar a versão original que veio no seu sistema operacional

Casos de uso:

- Todos na equipe de ciência de dados utilizadas a mesma versão da linguagem de programação (Python 3.7.2, R 4.0.2) com a mesma versão das bibliotecas (pandas 0.23.4, tidyverse 1.3.0)

Dados

Softwares

- Git Large File Storage (LFS)
- **DoltHub**
- **Great expectations**

Git LFS

Substitui grandes arquivos de áudio, vídeo, bases de dados por apontadores para estes arquivos em servidores dedicados a seu armazenamento

Git LFS - Provedores

A maioria dos provedores de *Git as a Service* oferecem uma implementação de Git LFS, tais como **GitHub**, **Azure Repos da Microsoft**, **BitBucket da Atlassian**, **GitLab**

Git LFS - uso

Baseado na **documentação do GitHub**. Após a **instalação**, rode os seguintes comandos no terminal:

```
git lfs install          # instala o LFS para seu usuário
git lfs track "*.png"    # selecione os tipos de arquivo
git add .gitattributes   # registre o .gitattributes
# comece o fluxo usual de git
git add images/*.png
git commit -m "Add image files"
git push origin master
```

DoltHub

Dolt é a experiência Git para bancos de dados SQL (relacionais), provendo versionamento para *schema*/esquema e para cada linha de dados, otimizando para colaboração

Além disso, o DoltHub é um local de hospedagem de bancos de dados públicos e privados, provendo ferramentas de hospedagem, interface para consultas.

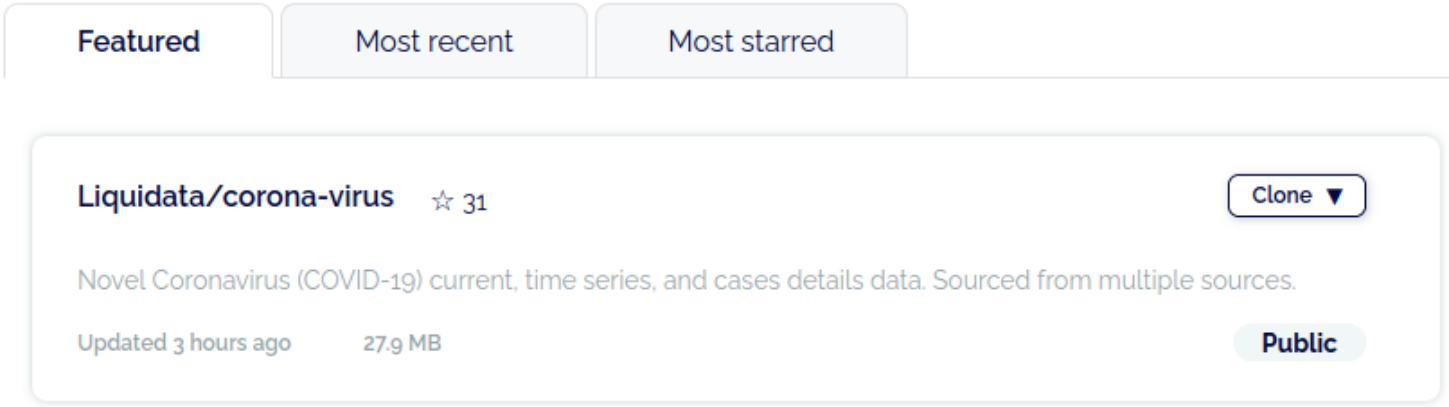


Figure 4: Bancos de dados públicos

Dolt - uso

Clonando (baixando) os dados:

```
$ dolt clone Liquidata/ip-to-country
cloning Liquidata/ip-to-country
23,716 of 23,716 chunks complete.
0 chunks being downloaded currently.
```

Mudanças na pasta:

```
$ ls -ltra
total 0
drwxr-xr-x   192B May  3 17:20 ../
drwxr-xr-x    96B May  3 17:20 ./
drwxr-xr-x   192B May  3 17:21 .dolt/
```

Dolt - inspecionando os dados

```
$ dolt sql
# Welcome to the DoltSQL shell.
# Statements must be terminated with ';'.
# "exit" or "quit" (or Ctrl-D) to exit.
ip_to_country> show tables;
+---------------+
| Table         |
+---------------+
| IPv4ToCountry |
| IPv6ToCountry |
+---------------+
```

Dolt - salvando resultados

```
$ dolt sql -q '\
  select Country,count(*)
  from IPv4ToCountry
  group by Country
  order by count(*) desc' -r csv > results.csv
$ head results.csv
Country,COUNT(*)
United States,56561
Brazil,10977
Russian Federation,10871
```

Great expectations

Saiba o que esperar dos seus dados - “Grandes Expectativas” ajudam times de dados a eliminar débitos de pipeline, por meio de testes para dados, documentação e perfilamento

Great expectations - configuração

```
Would you like to configure a Datasource? [Y/n]:

What data would you like Great Expectations to connect to?
    1. Files on a filesystem (for processing with Pandas or Spark)
    2. Relational database (SQL)
: 1

What are you processing your files with?
    1. Pandas
    2. PySpark
: 1

Enter the path (relative or absolute) of the root directory where the data files are stored.
: my_data

Give your new Datasource a short name.
 [my_data__dir]:

Great Expectations will now add a new Datasource 'my_data__dir' to your deployment,
by adding this entry to your great_expectations.yml:

  my_data__dir:
    data_asset_type:
      class_name: PandasDataset
      module_name: great_expectations.dataset
    batch_kwargs_generators:
      subdir_reader:
        class_name: SubdirReaderBatchKwargsGenerator
        base_directory: ../my_data
    class_name: PandasDatasource
```

Figure 5: Configuração

Great expectations - após adicionar um arquivo


```
{
  "data_asset_type": "Dataset",
  "expectation_suite_name": "npidata_pfile_20200511-20200517.warning",
  "expectations": [
    {
      "expectation_type": "expect_table_row_count_to_be_between",
      "kwargs": {
        "max_value": 20884,
        "min_value": 17087
      },
      "meta": {
        "BasicSuiteBuilderProfiler": {
          "confidence": "very low"
        }
      }
    },
    {
      "expectation_type": "expect_table_column_count_to_equal",
      "kwargs": {
        "value": 330
      },
      "meta": {
        "BasicSuiteBuilderProfiler": {
          "confidence": "very low"
        }
      }
    },
    {
      "expectation_type": "expect_table_columns_to_match_ordered_list",
      "kwargs": {
        "column_list": [
          "NPI",
          "Entity Type Code",
```


Figure 6: Grandes expectativas foram geradas

Execução de pipelines

Pipeline - conceito

Pipeline é tubulação.

Em vez de um único script (arquivo texto contendo instruções) ser executado manualmente, é necessário um tratamento mais elegante para soluções complexas do mundo real

Um pipeline de dados automatiza da tarefa de extração, transformação e carregamento ("exportação") de dados de um ou vários locais para um ou vários locais

Exemplos:

- Buscar dados do Google Analytics, realizar cálculos de agregação e colocar em um banco de dados da empresa
- Treinamento de um modelo de aprendizado de máquina que utiliza dados de transações financeiras
- Consolidação das informações contábeis do dia anterior

Software

- **Airflow**
- **dbt**
- **prefect**
- **Dagster**
- **Kedro**
- **Luigi**
- **pipecutter**
- **GNU Make**

pipecutter

```
import luigi
import pipecutter
from pipecutter.targets import JoblibTarget
from sklearn.ensemble import RandomForestClassifier

class TrainModel(luigi.Task):
    n_estimators = luigi.IntParameter()
    def output(self):
        return JoblibTarget(self.task_id + ".joblib")
    def run(self):
        model = RandomForestClassifier(
            n_estimators=self.n_estimators
            )
        self.output().dump(model)

pipecutter.run(TrainModel(n_estimators=100))
# -> Cria arquivo TrainModel_100_0b0ec0cdea.joblib
```

pipecutter - DAG

Direct acyclic graph (DAG, Grafo acíclico direto) é uma estrutura de dados em forma de grafo não linear com nós e arestas. No DAG, as arestas fluem em apenas uma direção.

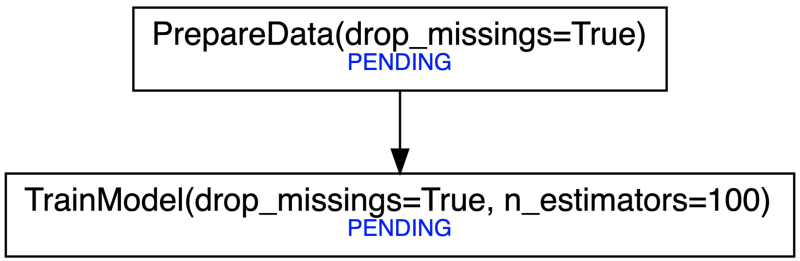


Figure 7: DAG simples gerado pelo pipecutter

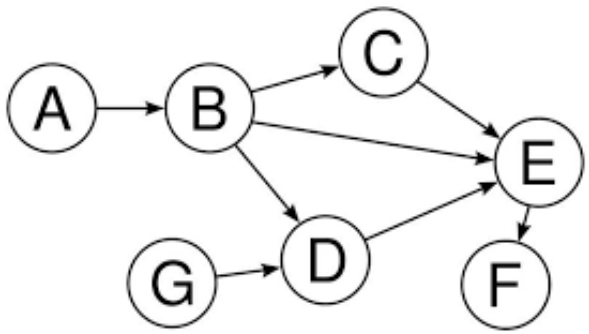


Figure 8: DAG mais complexo

GNU Make

Ferramenta de construção automatizada que cria programas executáveis e bibliotecas a partir de código fonte a partir da leitura de arquivos chamados *Makefiles*. Existem ferramentas específicas para certas linguagens e ambientes, mas Make ainda é muito utilizado em ambientes Unix-like.

Foir criado em 1976 por Stuart Feldman nos Laboratórios Bell. Hoje faz parte do ecossistema GNU mantido pela Fundação do Software Livre

GNU Make - estrutura

Target/alvo é o arquivo que vai ser (re)construído, *dependencies*/dependências são um ou mais arquivos são utilizados para construir o alvo, e *commands*/comandos são as instruções que devem ser executadas para (re)construir o alvo.

```
target:   dependencies ...
          commands
          ...
```

Supondo que uma dependência tenha uma data de modificação mais recente que o alvo (ou seja, sofreu uma alteração que ainda não refletiu no alvo), Make oferece uma interface rápida para atualizar o alvo (sem atualizar alvos que não precisem ser atualizados)

GNU Make - ciência de dados

Zachary Jones escreveu um **post** mostrando a utilização de Make para criar um pipeline de ciência de dados poliglota

Rodando apenas o comando `make` todas as etapas serão executadas, mas também é possível executar apenas a extração de dados com `make data`

```
all: data model
data: raw.csv
model: model.Rout
raw.csv: get_data.py
    python get_data.py
clean.csv: clean.sh raw.csv
    source clean.sh
model.Rout: model.R clean.csv
    R CMD BATCH model.R
```

Aprendizado de máquina

Software

- Usar as abstrações de pipelines (scikit-learn, spark)
- TensorFlow Hub, PyTorch Hub
- **MLflow**
- scikit-learn - **choosing the right estimator**

Pipeline em scikit-learn

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC())])
# The pipeline can be used as any other estimator
# and avoids leaking the test set into the train set
pipeline.fit(X_train, y_train)
Pipeline(steps=[("scaler", ..), ("svc", ..)])
pipeline.score(X_test, y_test)
0.88
```

Pipeline em spark

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
                      outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

pipeline = pipeline.fit(training)

prediction = model.transform(test)
```

TensorFlow Hub, PyTorch Hub

Tanto **TensorFlow** - plataforma de código aberto apoiada pela **Google** - quanto **PyTorch** - biblioteca para aprendizado profundo apoiada pelo **Facebook** - possuem um local para descoberta e publicação de modelos pré treinados

O **PyTorch Hub** e o **TensorFlow**

MLflow

Uma plataforma de código aberto para o ciclo completo de aprendizado de máquina

- Funciona com qualquer biblioteca de aprendizado de máquina e linguagem de programação
- Roda do mesmo jeito na nuvem
- Escalabilidade de um para vários usuários
- Escala para *big data* com Spark

MLflow - rastreamento

```
import mlflow
import mlflow.spark

with mlflow.start_run():
    mlflow.log_tag("estimatorName", "RandomForest")
    mlflow.log_param("n_estimators", 100)
    mlflow.log_metric("rmse", 3.14)
    mlflow.log_metric("nrows", 10084)
    mlflow.log_artifact(["age", "itemsPurchased"])
    mlflow.spark.log_model(spark_model)
```

MLflow - visualização

	Date	User	Source	Version	Parameters		Metrics		
					alpha	l1_ratio	mae	r2	rmse
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.288	0.742
☐	2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.288	0.742

Figure 9: Visualização de modelos

MLflow - reproducibilidade

```
# sklearn_elasticnet_wine/MLproject

name: tutorial

conda_env: conda.yaml

entry_points:
  main:
    parameters:
      alpha: float
      l1_ratio: {type: float, default: 0.1}
    command: "python train.py {alpha} {l1_ratio}"
```

The Conda file lists the dependencies:

```
# sklearn_elasticnet_wine/conda.yaml

name: tutorial
channels:
  - defaults
dependencies:
  - numpy=1.14.3
  - pandas=0.22.0
  - scikit-learn=0.19.1
  - pip:
    - mlflow
```

Figure 10: Ambiente Anaconda reproduzível

scikit-learn: escolhendo o estimator certo

A biblioteca scikit-learn oferece uma miríade de estimadores

É importante conhecer as bases teóricas de cada um (ou cada classe de estimadores, pelo menos) para saber qual utilizar em certa situação

Caso você não tenha ciência de todos ou do funcionamento geral, a biblioteca oferece um gráfico de fluxo conhecido como **escolhendo o estimador certo**

Escolhendo o estimador certo: fluxo

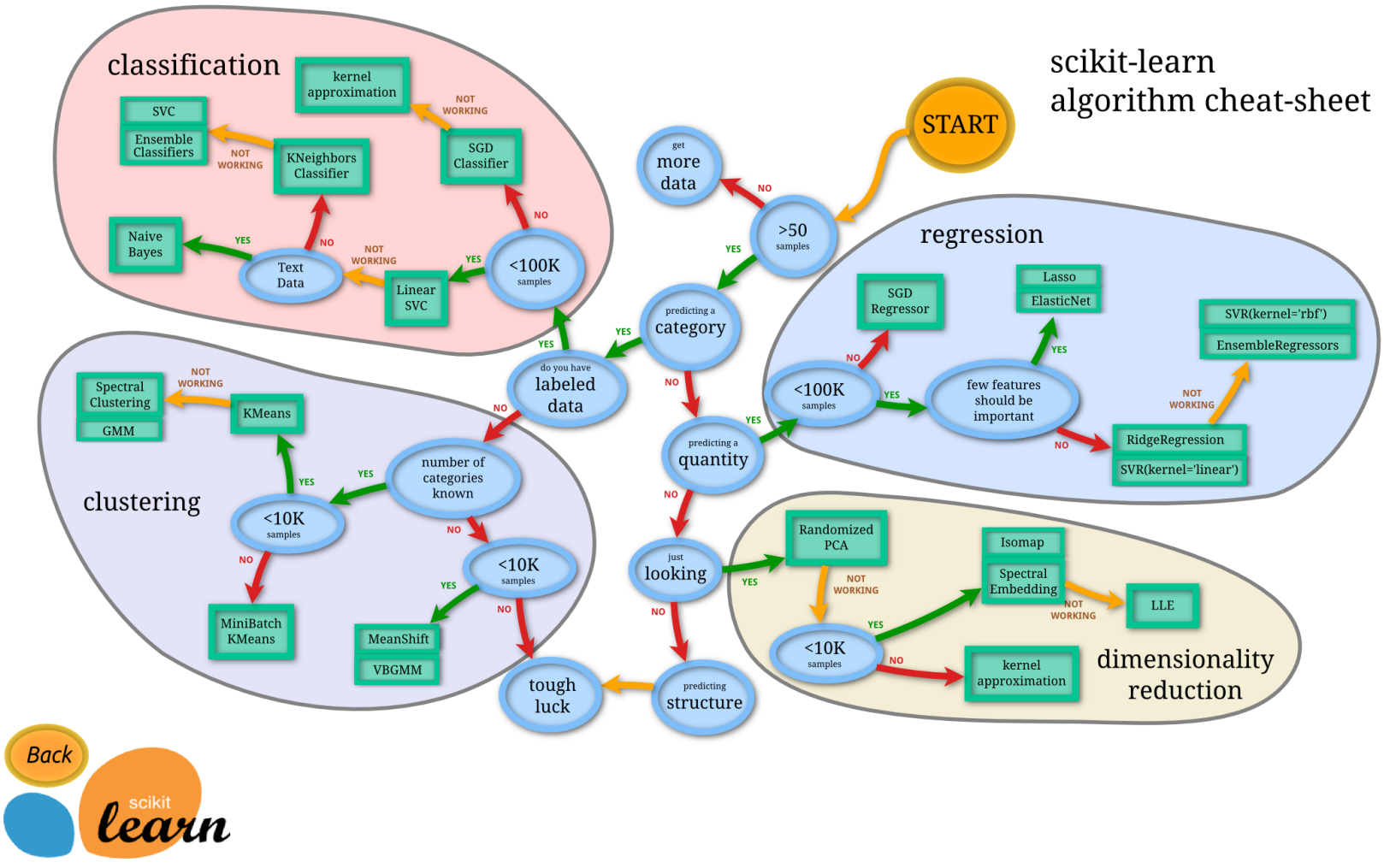
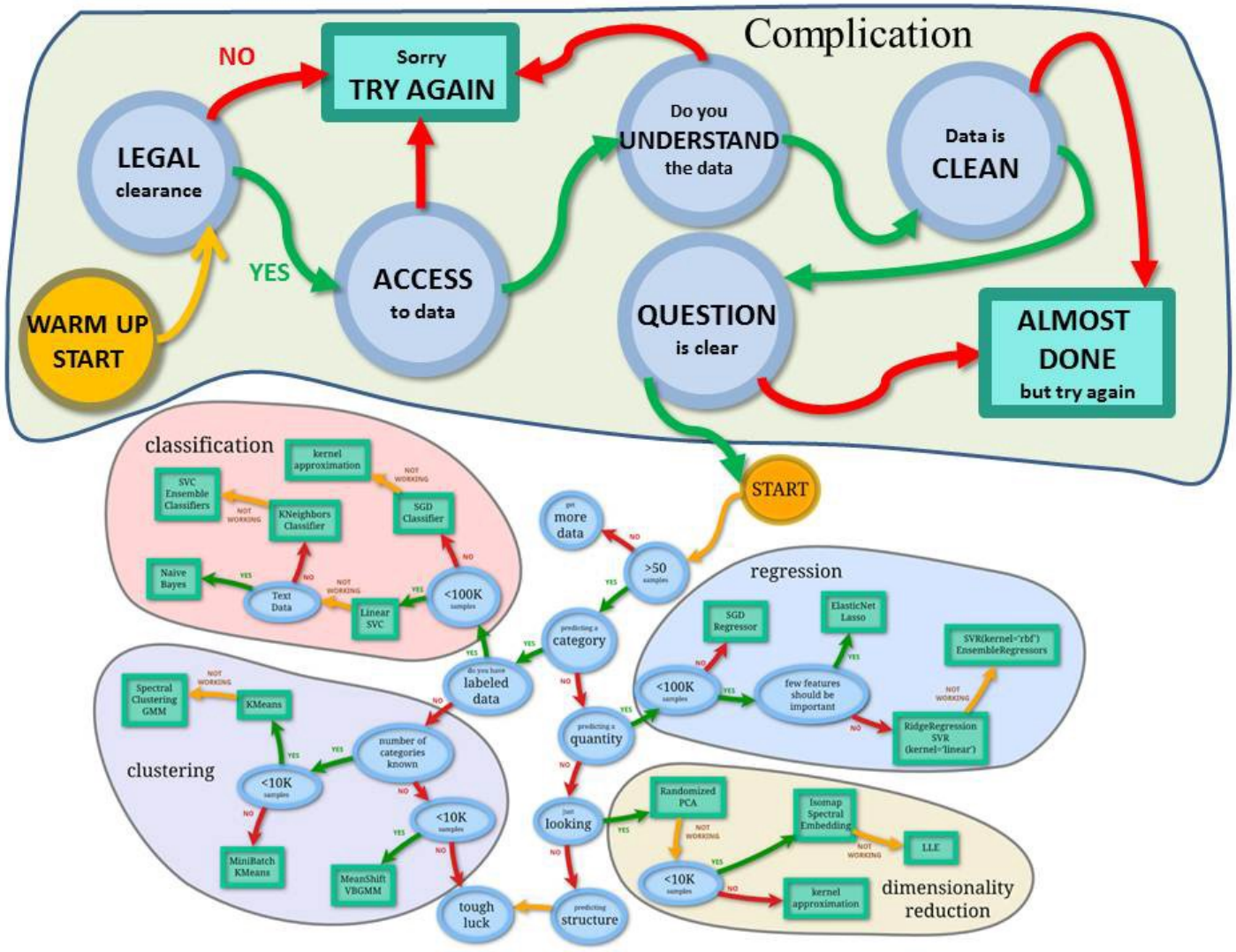


Figure 11: Escolhendo o estimador certo

Uma versão extendida do gráfico de fluxo do scikit-learn

Christophe Bourguignat escreveu o artigo "**An extended version of the scikit-learn cheat sheet**"



Permissão legal

Mesmo que os dados já estejam "dentro de casa", grandes organizações ou empresas regulamentadas (*fintechs*) possuem regras "rígidas" para acesso de dados

Se o projeto for feito para uma empresa externa, devem ser estabelecidas regras claras sobre armazenamento, acesso, governança, disponibilidade. É praxe as partes (empresas) e até mesmo colaboradores assinarem NDAs (*non disclosure agreements* ou acordos de não abertura)

Às vezes os dados já estão coletados, mas não há permissão dos usuários para certos novos usos. E.g. as transações dos usuários são armazenadas, mas pode recomendar produtos para os usuários (envio de email, push, sms)

Acesso aos dados

- Regras de acesso complexas
- Dados não centralizados, uso de vários sistemas
- Uso de VPN, autenticação, firewalls

Entendimento dos dados - competição

Em competições de aprendizado de máquina, as variáveis estão claramente diferenciadas entre dependente e independentes, não é necessário saber o que cada uma significa, às vezes isso é um requisito para garantir a confidencialidade e anonimização dos dados

Exemplo:

- rótulo: variável dependente que é 1 se o banner foi clicado e 0 caso contrário
- I1-I13: são as colunas cujo tipo é inteiro
- C1-C26: são as colunas cujo tipo é texto ou categóricas

Entendimento dos dados - "mundo real"

O entendimento dos dados pode ajudar em:

1. Criar modelos interpretáveis: a cada 1min a mais que o visitante passa o site, a chance de gerar uma compra aumenta em 5%
2. Ajudar na criação de características: colunas são informações extraídas em bancos de dados, arquivos e outros formatos, características são as informações que serão colocados para que os estimadores interpretem e vejam os padrões

Supondo que você precise criar um sistema de recomendação de produto, você pode usar:

- cod_item, cod_sku, cod_cat_item da tabela items
- cod_produto, cod_subproduto, cod_cat_subproduto da tabela produtos
- produto_nome, item_nome, id_produto, sku_produto da tabela transactions
- cod_sku, cod_inventario da tabela inventory

Limpeza dos dados

É reconhecida como uma das fases mais dispendiosas de tempo e esforço num projeto de ciência de dados

Alguns casos comuns são: dados de teste no início ou ao longo do período histórico (testes automatizados ou manuais que percorrem um site em produção), erros de digitação de dados manuais, dados não preenchidos (cadastro feito por pessoas no mundo físico), fraudes, outliers (pontos fora da curva), amostras que não são do interesse do projeto (pessoas com certas características ou certas transações)

Pergunta de negócio

Cientistas de dados amam resolver problemas complexos, alguns úteis, outros nem tanto

Antes de começar o projeto é importante todos os stakeholders estarem alinhados quanto aos objetivos a serem alcançados, o que será predito, qual a performance mínima exigida, qual o retorno sobre o investimento mínimo atrativo, como integrar a saída do modelo com os demais sistemas da organização e processos que as pessoas realizam

Chegou até aqui?

Então agora pode entrar no fluxo original do scikit-learn!

Figure 13: Pronto para o aprendizado de máquina