TREINAMENTO EM ML - Processo Sistemático de Desenvolvimento

DEFININDO O PROBLEMA

Podemos definir um problema utilizando 3 passos:

- O que é o problema? Descreva o problema nas formas informal e formal e tente associar com problemas já existentes ou já resolvidos.
- Por que o problema precisa ser resolvido? Liste a motivação, os benefícios e como a solução pode ser utilizada, com o maior detalhe possível.
- Como eu poderia resolver o problema? Descreva, a mão, como este problema poderia ser resolvido com o maior número de detalhes possíveis.

PREPARAÇÃO DOS DADOS

Podemos preparar os dados segundo estes passos:

- Seleção de Dados (Considere quais dados estão disponíveis, quais dados precisam ser retirados e quais dados estão faltando.).
- Pré-Processamento dos Dados (Organize os dados com operações de filtragem, formatação, limpeza e amostragem).
- Transformação dos Dados (Transforme os dados para o formato necessário indicado pelos algoritmos de ML. Utilize normalização, escala, decomposição, agregação, etc).

ESCOLHA DOS ALGORITMOS E AVALIAÇÃO DAS MÉTRICAS

Se o problema for de classificação de padrões, devemos observar se é um problema multiclasse ou biclasse. Se é um problema de classificação balanceado ou não balanceado, etc.

Escolha de 10 a 20 algoritmos diferentes e realize o treinamento avaliando as métricas necessárias. Bom observar que existem métricas diferentes para classificadores multiclasse e biclasse. Assim, as métricas adequadas devem ser escolhidas. Se o problema for de regressão ou clusterização, existem métricas específicas.

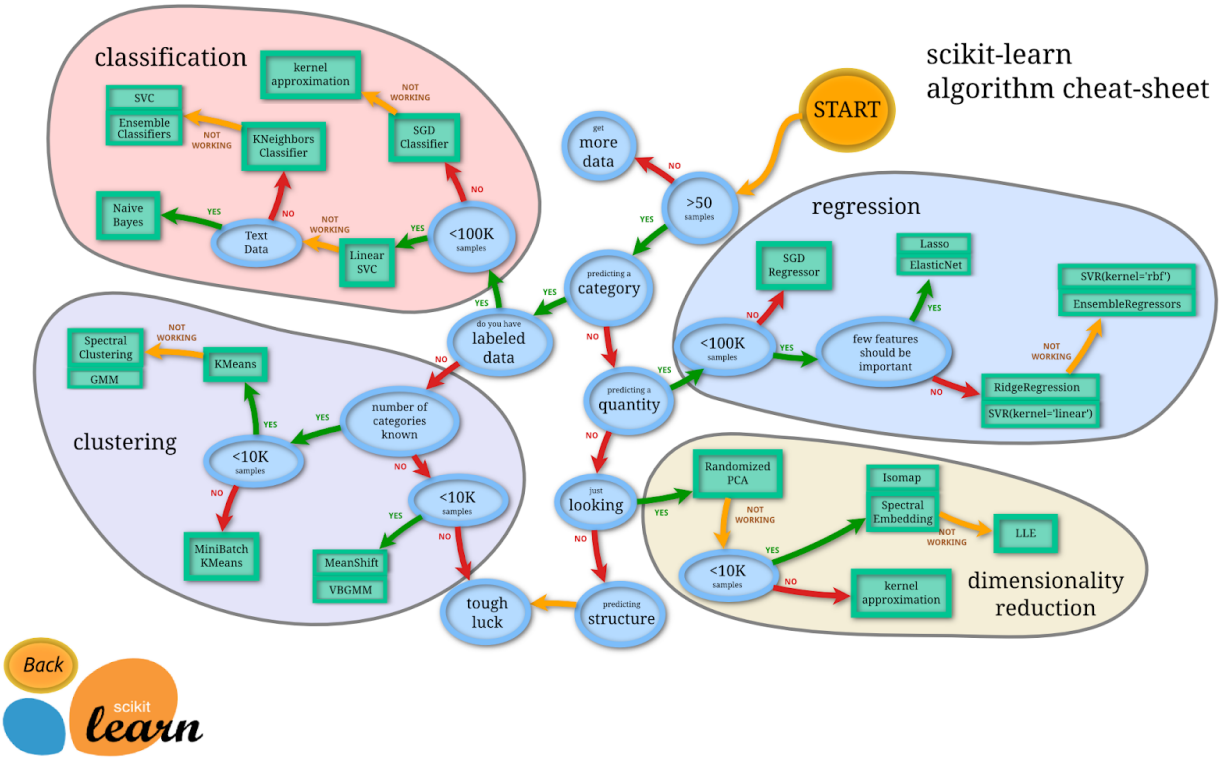

Figura 1 - Como escolher o algoritmo certo para o problema.
(https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

Devemos utilizar, também, as metodologias de Cross Validation ou Validação Cruzada para verificarmos profundamente se o modelo está adequado ou não. Utilizamos CV = 10 que indica 10 subdivisões dos dados originais. A métrica final do treinamento será a média das 10 métricas calculadas. Os algoritmos melhores avaliados serão utilizados para tunning nas etapas posteriores. Aqui é bom observarmos que para um mesmo conjunto de dados, o rendimento dos algoritmos são diferentes, portanto, escolher um algoritmo dito “padrão” pode ser desastroso! A escolha do melhor algoritmo é apenas o ponto de partida e não o fim do projeto. O algoritmo que foi melhor no treinamento pode não ser o melhor em produção. Assim sendo, é interessante pegarmos alguns outros algoritmos da mesma família para garantirmos o melhor resultado possível.

MELHORANDO OS RESULTADOS

Para melhorar os resultados podemos utilizar as seguintes técnicas:

- Tuning do Algoritmo (modificar os parametros do algoritmo).
- Ensemble Methods (associação de vários algoritmos para um resultado comum).
- Extreme Feature Engineering (experimentos com agregação, decomposição e redução de dimensionalidade (PCA, MDS, ISOMAP, etc).

Ensemble pode funcionar bem quando há vários modelos bons especializados em diferentes partes do problema. As três estratégias principais são: Bagging, Boosting e Blending.

- Bagging (ou Bootstrapped Aggregation) é quando o mesmo algoritmo apresenta resultados diferentes quando submetido a subconjuntos diferentes do dataset original.
- Boosting (Algoritmos diferentes são treinados com o mesmo dado de treinamento).
- Blending (Stacked Aggregation) acontece quando a saída de um algoritmo é a entrada do outro algoritmo predizendo ao final o sistema global.

De uma maneira geral, se os algoritmos tradicionais não obtiverem bons resultados, técnicas Ensemble devem ser utilizadas. Porém, estas técnicas são mais complexas.

EXEMPLO DE CÓDIGO EM PYTHON3.6

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from imblearn.metrics import classification_report_imbalanced
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn import datasets
from sklearn.model_selection import KFold
from sklearn.preprocessing import Normalizer

classifiers = {"RF": RandomForestClassifier(n_estimators=100),
           "KNN": KNeighborsClassifier(),
           "DTREE": DecisionTreeClassifier(),
           "GNB": GaussianNB(),
           "LRG": LogisticRegression(),
```

```
            "ABC": AdaBoostClassifier(),
            "MLP": MLPClassifier(max_iter=500,alpha=1),
            "KDA": QuadraticDiscriminantAnalysis(),
            "SVM1": SVC(kernel="linear", C=0.025),
            "SVM2": SVC(gamma=2, C=1),
            "GPC": GaussianProcessClassifier(1.0 * RBF(1.0))
            }
nfolds = 10
kf = KFold(n_splits=nfolds,shuffle=True)
transformer = Normalizer()
iris = datasets.load_iris()
X = iris.data
X = transformer.fit_transform(X)
y = iris.target
dfcol = ['FOLD','ALGORITHM','PRE', 'REC', 'SPE', 'F1', 'GEO', 'IBA', 'ACC']
df = pd.DataFrame(columns=dfcol)
i = 0
fold = 0
for train_index, test_index in kf.split(X):
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
   for name, clf in classifiers.items():
      clf.fit(X_train, y_train)
      y_pred = clf.predict(X_test)
      res = classification_report_imbalanced(y_test, y_pred)
      aux = res.split()
      score = aux[-7:-1]
      df.at[i, 'FOLD'] = fold
      df.at[i, 'ALGORITHM'] = name
      df.at[i, 'PRE'] = score[0]
      df.at[i, 'REC'] = score[1]
      df.at[i, 'SPE'] = score[2]
      df.at[i, 'F1'] = score[3]
      df.at[i, 'GEO'] = score[4]
      df.at[i, 'IBA'] = score[5]
      df.at[i, 'ACC'] = accuracy_score(y_test, y_pred)
      i = i + 1
```

```
        print(str(fold) + " " + str(name))
    fold = fold + 1
df.to_csv('results_iris.csv', index=False)

t = pd.Series(data=np.arange(0, df.shape[0], 1))
dfr = pd.DataFrame(columns=['ALGORITHM', 'PRE', 'REC', 'SPE', 'F1', 'GEO', 'IBA',
'ACC'],
                   index=np.arange(0, int(t.shape[0] / nfolds)))
df_temp = df.groupby(by=['ALGORITHM'])
idx = dfr.index.values
i = idx[0]
for name, group in df_temp:
    group = group.reset_index()
    dfr.at[i, 'ALGORITHM'] = group.loc[0, 'ALGORITHM']
    dfr.at[i, 'PRE'] = group['PRE'].astype(float).mean()
    dfr.at[i, 'REC'] = group['REC'].astype(float).mean()
    dfr.at[i, 'SPE'] = group['SPE'].astype(float).mean()
    dfr.at[i, 'F1'] = group['F1'].astype(float).mean()
    dfr.at[i, 'GEO'] = group['GEO'].astype(float).mean()
    dfr.at[i, 'IBA'] = group['IBA'].astype(float).mean()
    dfr.at[i, 'ACC'] = group['ACC'].astype(float).mean()
    i = i + 1

dfr.to_csv('media_results_iris.csv', index=False)
```