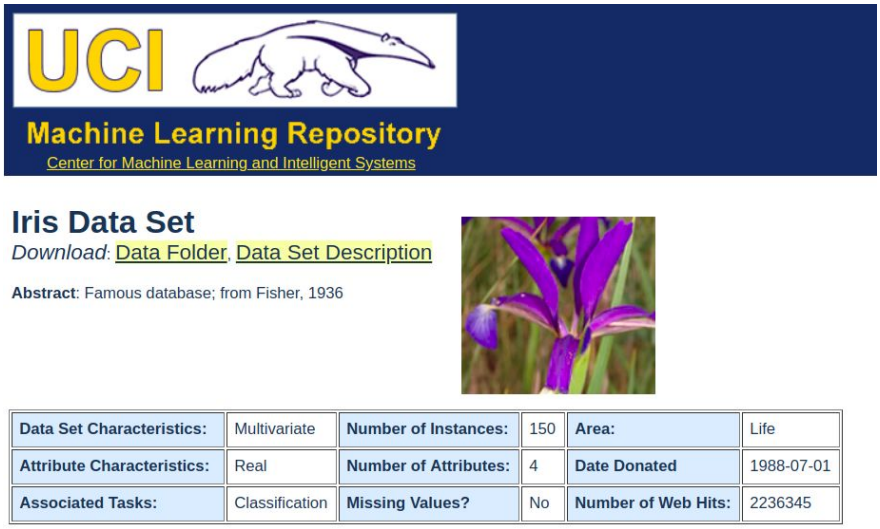
CLASSIFICAÇÃO DE PADRÕES

Cada problema tem o seu domínio próprio e suas características. Estas características se refletem no conjunto de dados deste domínio. Para exemplificar, vamos supor que queiramos criar um sistema de classificação de uma variedade específica de planta.



A UCI fornece um conjunto de dados para pesquisas chamado UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php). Este site possui vários tipos de conjuntos de dados. Para o nosso sistema de classificação, iremos utilizar o **Iris Dataset** (https://archive.ics.uci.edu/ml/datasets/iris).

Os atributos deste conjunto de dados são:

- Comprimento da pétala (cm)
- Largura da pétala (cm)
- Comprimento da sépala (cm)
- Largura da sépala(cm)

Classes: Iris Setosa, Iris Versicolour e Iris Virginica.
Número de classes: 3
Número de atributos: 4
Número de instâncias: 150 (50 de cada classe).

Instâncias	**Comprimento da Pétala**	**Largura da Pétala**	**Comprimento da Sépala**	**Largura da Sépala**	**Classe**
1	5.1	3.5	1.4	0.2	**Setosa**

2	7.0	3.2	4.7	1.4	**Versicolor**
3	6.3	3.3	6.0	2.5	**Virginica**
4	5.8	2.7	5.1	1.9	**Virginica**
...	...	...	...	...	**...**
150	6.3	2.8	5.1	1.5	**Virginica**

Tabela 1: Representação de um conjunto de dados com atributos e suas respectivas classes.

Problema a ser resolvido

Nesta caso, o problema é construir um componente de machine learning que seja capaz de classificar novas plantas submetidas ao sistema. Para isso utilizaremos técnicas de treinamento supervisionado.

Solução do Problema

Para solucionar este problema, vamos criar um classificador automático mas, para criar um classificador razoavelmente preciso, precisamos medir os seus acertos e seus erros.
Para tanto, vamos utilizar métricas.
As principais métricas para classificação de padrões são:

ACC - Accuracy (https://en.wikipedia.org/wiki/Accuracy_and_precision#In_binary_classification)
F1-score (https://en.wikipedia.org/wiki/F1_score)
AUC - Area Under the Curve (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5)
GEO - Geometric mean (https://en.wikipedia.org/wiki/Geometric_mean)
IBA - Index of Balanced Accuracy (https://core.ac.uk/download/pdf/61392839.pdf)
PRE - Precision (https://en.wikipedia.org/wiki/Precision_and_recall)
REC - Recall (https://en.wikipedia.org/wiki/Precision_and_recall)
SPE - Specificity (https://en.wikipedia.org/wiki/Sensitivity_and_specificity)

Como avaliar estas métricas?

	ACC	**F1**	**AUC**	**GEO**	**IBA**	**PRE**	**REC**	**SPE**

Melhor valor	100%	100%	100%	100%	100%	100%	100%	100%

Tabela 2: Quanto mais próximo de 100% for, melhor será o desempenho do classificador.

De acordo com a tabela 2, os melhores valores podem chegar a 100%, o que é muito raro. Tem-se de levar em conta, também, que uma métrica isolada pode falsear as expectativas de resultados. Por exemplo, considere uma amostra de 100 instâncias sendo que 95 sejam da classe A e 5 da classe B. Se um classificador acertar todos os elementos da classe A e errar todos os elementos da classe B, ele terá uma acurácia igual a 95%, o que **parece** ser muito bom! Mas imagine que a classe A corresponda a mamografias normais, ou seja, as que não possuem nenhum tipo de tumor e a classe B sejam as mamografias com tumor. Neste caso, o classificador acertaria todas as mamografias sem tumor e erraria todas as mamografias com tumor, o que é muito grave! Portanto, neste caso, 95% de acurácia não quer dizer muita coisa.

Agora que sabemos como avaliar as métricas, precisamos aprender como preparar os dados e como preparar os testes para verificar que o classificador construído realmente irá fazer o serviço pesado da operação (produção).

Preparação dos Dados para Validação Cruzada (Cross Validation)

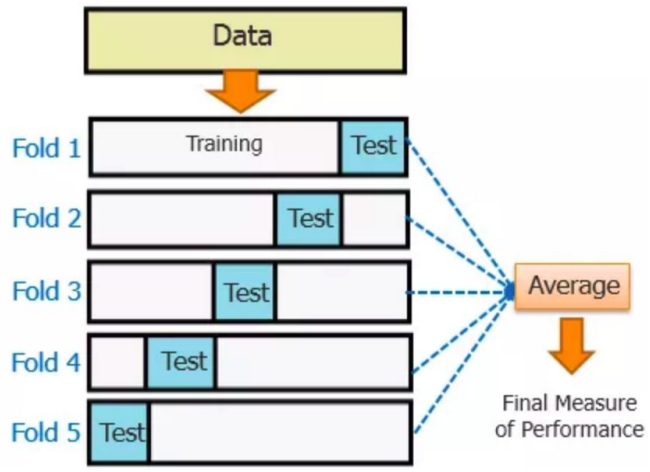


Figura 1 : Validação Cruzada (https://blog.contactsunny.com/data-science/different-types-of-validations-in-machine-learning-cross-validation)

Vamos entender agora o que é o processo de Validação Cruzada. A figura 1 mostra a representação de um conjunto de dados (Dataset). Digamos que seja o **Iris Dataset**. Este

conjunto de dados deve ser dividido em 5 ou 10 pastas. A amostragem aleatória pode ser feita com ou sem reposição, dependendo da quantidade de dados disponíveis. Como o Iris Dataset só possui 150 instâncias, iremos fazer a amostragem com reposição, ou seja, o mesmo elemento pode fazer parte de todas as pastas. Para cada pasta, dividimos os dados em dados de teste e dados de treinamento podendo ser 75% e 25% respectivamente. Esta divisão também é feita de forma aleatória em cada pasta. Após este procedimento, executamos o treinamento do classificador escolhido (explicado mais adiante) e verificamos as métricas de todas as pastas. O resultado global do classificador será a média de todos os resultados.

Escolha do Algoritmo de Classificação

Na escolha do algoritmo de classificação, vários fatores devem ser observados tais como a quantidade de dados disponíveis para o treinamento, a potência da máquina necessária para o treinamento, a velocidade do processo de treinamento e, o mais importante, sua compatibilidade com os dados. É muito comum verificarmos que para o mesmo conjunto de dados, um algoritmo atinja métricas próximas a 100% enquanto outros algoritmos tenham resultados desanimadores. Isso tudo para o mesmo conjunto de dados! Sendo assim, é uma boa prática testarmos vários classificadores para o mesmo problema.
Vejamos alguns classificadores na página:
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html .

```
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
```

Aqui temos 9 classificadores diferentes, sendo que um deles possui duas configurações (SVC).

Fazendo a codificação em Python 3.6 temos:

```python
# -*- coding: utf-8 -*-
import pandas as pd
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))
               }
kf = KFold(n_splits=10,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', 'AUC', '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, 'AUC'] = roc_auc_score(y_test, y_pred)
        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)
```

E o resultado foi:

FOLD,ALGORITHM,PRE,REC,SPE,F1,GEO,IBA,AUC,ACC
0,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,DTREE,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,LRG,0.60,0.73,0.90,0.64,0.68,0.64,,0.7333333333333333
0,ABC,0.91,0.87,0.95,0.87,0.90,0.81,,0.8666666666666667
0,MLP,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,KDA,0.95,0.93,0.98,0.93,0.95,0.91,,0.9333333333333333
0,SVM1,0.36,0.53,0.83,0.41,0.43,0.37,,0.5333333333333333
0,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
0,GPC,0.95,0.93,0.98,0.93,0.95,0.91,,0.9333333333333333
1,RF,0.96,0.93,0.99,0.94,0.96,0.92,,0.9333333333333333
1,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,DTREE,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,LRG,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,ABC,0.91,0.73,0.96,0.73,0.81,0.69,,0.7333333333333333
1,MLP,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,SVM1,0.02,0.13,0.87,0.03,0.00,0.00,,0.13333333333333333
1,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
1,GPC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
2,RF,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,KNN,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,DTREE,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,GNB,0.90,0.87,0.93,0.86,0.89,0.80,,0.8666666666666667
2,LRG,0.42,0.60,0.80,0.47,0.48,0.41,,0.6
2,ABC,0.87,0.87,0.92,0.87,0.89,0.80,,0.8666666666666667
2,MLP,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,KDA,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,SVM1,0.07,0.27,0.73,0.11,0.00,0.00,,0.26666666666666666
2,SVM2,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
2,GPC,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
3,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
3,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
3,DTREE,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
3,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
3,LRG,0.85,0.73,0.87,0.70,0.76,0.60,,0.7333333333333333
3,ABC,0.94,0.93,0.96,0.93,0.94,0.89,,0.9333333333333333
3,MLP,0.94,0.93,0.97,0.93,0.95,0.90,,0.9333333333333333
3,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0

3,SVM1,0.07,0.27,0.73,0.11,0.00,0.00,,0.26666666666666666
3,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
3,GPC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,DTREE,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,LRG,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,ABC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,MLP,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,SVM1,0.04,0.20,0.80,0.07,0.00,0.00,,0.2
4,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
4,GPC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,RF,0.95,0.93,0.98,0.93,0.95,0.91,,0.9333333333333333
5,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,DTREE,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,LRG,0.36,0.53,0.83,0.41,0.43,0.37,,0.5333333333333333
5,ABC,0.95,0.93,0.98,0.93,0.95,0.91,,0.9333333333333333
5,MLP,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,SVM1,0.36,0.53,0.83,0.41,0.43,0.37,,0.5333333333333333
5,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
5,GPC,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
6,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
6,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
6,DTREE,0.95,0.93,0.98,0.93,0.95,0.91,,0.9333333333333333
6,GNB,0.94,0.93,0.96,0.93,0.94,0.89,,0.9333333333333333
6,LRG,0.90,0.87,0.91,0.85,0.87,0.78,,0.8666666666666667
6,ABC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
6,MLP,0.94,0.93,0.96,0.93,0.94,0.89,,0.9333333333333333
6,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
6,SVM1,0.07,0.27,0.73,0.11,0.00,0.00,,0.26666666666666666
6,SVM2,0.94,0.93,0.96,0.93,0.94,0.89,,0.9333333333333333
6,GPC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
7,RF,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,KNN,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,DTREE,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,GNB,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,LRG,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,ABC,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,MLP,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333

7,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
7,SVM1,0.36,0.53,0.83,0.41,0.43,0.37,,0.5333333333333333
7,SVM2,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
7,GPC,0.94,0.93,0.94,0.93,0.93,0.88,,0.9333333333333333
8,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,KNN,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,DTREE,0.94,0.93,0.96,0.93,0.94,0.89,,0.9333333333333333
8,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,LRG,0.84,0.73,0.82,0.70,0.73,0.56,,0.7333333333333333
8,ABC,0.90,0.87,0.91,0.86,0.88,0.78,,0.8666666666666667
8,MLP,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,KDA,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,SVM1,0.04,0.20,0.80,0.07,0.00,0.00,,0.2
8,SVM2,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
8,GPC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
9,RF,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
9,KNN,0.87,0.87,0.91,0.87,0.89,0.79,,0.8666666666666667
9,DTREE,0.96,0.93,0.99,0.94,0.96,0.92,,0.9333333333333333
9,GNB,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
9,LRG,0.36,0.47,0.92,0.38,0.42,0.39,,0.4666666666666667
9,ABC,1.00,1.00,1.00,1.00,1.00,1.00,,1.0
9,MLP,0.96,0.93,0.99,0.94,0.96,0.92,,0.9333333333333333
9,KDA,0.87,0.87,0.91,0.87,0.89,0.79,,0.8666666666666667
9,SVM1,0.02,0.13,0.87,0.03,0.00,0.00,,0.13333333333333333
9,SVM2,0.96,0.93,0.99,0.94,0.96,0.92,,0.9333333333333333
9,GPC,0.87,0.87,0.91,0.87,0.89,0.79,,0.8666666666666667

Para calcularmos o resultado final dos classificadores, precisamos apenas fazer o cálculo da média de cada fold(0 a 9) para cada algoritmo. O código abaixo faz isso.

```
64  df.to_csv('results_iris.csv', index=False)
65
66  t = pd.Series(data=np.arange(0, df.shape[0], 1))
67  dfr = pd.DataFrame(columns=['ALGORITHM', 'PRE', 'REC', 'SPE', 'F1', 'GEO', 'IBA', 'ACC'],
68                     index=np.arange(0, int(t.shape[0] / nfolds)))
69  df_temp = df.groupby(by=['ALGORITHM'])
70  idx = dfr.index.values
71  i = idx[0]
72  for name, group in df_temp:
73      group = group.reset_index()
74      dfr.at[i, 'ALGORITHM'] = group.loc[0, 'ALGORITHM']
75      dfr.at[i, 'PRE'] = group['PRE'].astype(float).mean()
76      dfr.at[i, 'REC'] = group['REC'].astype(float).mean()
77      dfr.at[i, 'SPE'] = group['SPE'].astype(float).mean()
78      dfr.at[i, 'F1'] = group['F1'].astype(float).mean()
79      dfr.at[i, 'GEO'] = group['GEO'].astype(float).mean()
80      dfr.at[i, 'IBA'] = group['IBA'].astype(float).mean()
81      dfr.at[i, 'ACC'] = group['ACC'].astype(float).mean()
82      i = i + 1
83
84  dfr.to_csv('media_results_iris.csv', index=False)
```

A tabela abaixo mostra estas médias para cada algoritmo por métrica.

	ALGORITHM	PRE	REC	SPE	F1	GEO	IBA	ACC
0	ABC	0.9049999999999999	0.873	0.933	0.8699999999999999	0.8949999999999999	0.812	0.8733333333333333
1	DTREE	0.937	0.9259999999999999	0.9629999999999999	0.9269999999999999	0.942	0.8920000000000001	0.9266666666666667
2	GNB	0.9780000000000001	0.9719999999999999	0.986	0.9719999999999999	0.9780000000000001	0.959	0.9733333333333334
3	GPC	0.9650000000000001	0.958	0.975	0.958	0.9640000000000001	0.9339999999999999	0.9600000000000002
4	KDA	0.9709999999999999	0.9649999999999999	0.9790000000000001	0.9649999999999999	0.97	0.945	0.9666666666666668
5	KNN	0.9769999999999998	0.9719999999999999	0.9799999999999999	0.9719999999999999	0.974	0.954	0.9733333333333334
6	LRG	0.724	0.733	0.8870000000000001	0.6689999999999999	0.703	0.6249999999999999	0.7333333333333333
7	MLP	0.9720000000000001	0.9649999999999999	0.978	0.9649999999999999	0.9700000000000001	0.945	0.9666666666666668
8	RF	0.9729999999999999	0.966	0.984	0.966	0.9729999999999999	0.9490000000000001	0.9666666666666668
9	SVM1	0.145	0.30100000000000005	0.7989999999999999	0.17500000000000002	0.1	0.1	0.3
10	SVM2	0.9829999999999999	0.9789999999999999	0.9879999999999999	0.9789999999999999	0.982	0.968	0.9800000000000001

Análise dos Resultados e Escolha do Melhor Modelo

Na linha 9 podemos observar que o algoritmo SVM-1 não alcançou um bom desempenho. O algoritmo LRG (linha 6) apresentou melhores resultados mas ainda está bem abaixo do desempenho dos outros algoritmos. O algoritmo ABC (linha 0) apresentou resultados inferiores aos demais algoritmos. Sendo assim, qualquer dos outros algoritmos poderiam ser escolhidos, uma vez que apresentaram bons resultados. O critério, a partir deste ponto, poderia ser qual algoritmo é mais rápido para fazer o treinamento. Pode parecer irrelevante este critério mas, dependendo do tamanho do conjunto de dados, isso pode representar uma diferença de muitas horas de processamento.

Deploy do Componente de Inteligência

Agora que verificamos qual algoritmo se adaptou melhor no trabalho de classificação podemos fazer o Deploy do componente de inteligência para ser utilizado por um site, por um webserver, por um programa desktop, um aplicativo, etc.

Um dos melhores resultados para este dataset foi o algoritmo **GaussianNB** (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html). Assim sendo, iremos criar o modelo de deploy baseado neste algoritmo.

O código abaixo cria um classificador baseado no algoritmo **Gaussian Naive Bayes (GaussianNB)**. O modelo é salvo no arquivo **'iris.ml.gzip' .** O resultado final da classificação é gravado no arquivo **'resultados_iris_modelo_salvo.csv' .** Bom ressaltar que o exemplo classificado está na mesma linha do arquivo de resultados.

	0	1	2	3	
57	0.7591654715238996	0.3718361493178283	0.5112747053120139	0.15493172888242845	1
58	0.7630185275970008	0.3352657166714095	0.5318007919615461	0.15029152816304564	1
59	0.7246023348632824	0.37623582771747355	0.5434517511474618	0.19508524400165295	1
60	0.7692307692307693	0.3076923076923077	0.5384615384615384	0.15384615384615385	1
61	0.7392346162730675	0.37588200827444107	0.5262348115842176	0.18794100413722054	1
62	0.7889275245573255	0.2892734256710194	0.5259516830382169	0.13148792075955423	1
63	0.7308141200229883	0.3474362209945354	0.5630862891980403	0.16772783082494813	1
64	0.7591170716092961	0.3931141977976712	0.48800383174883327	0.1762236059093009	1
65	0.7694544446831321	0.3560162355996581	0.5053133666575793	0.1607815257546843	1
66	0.706318918233044	0.3783851347677022	0.5675777021515532	0.1891925673838511	1
67	0.7567649730125051	0.35228714260926963	0.5349545498881502	0.13047671948491468	1
68	0.7644423782009608	0.27125374710356676	0.5548372099845683	0.18494573666152278	1
69	0.7618518793947621	0.34011244615837594	0.5305754160070665	0.14964947630968542	2
70	0.6985796007419844	0.37889063091090686	0.5683359463663602	0.2131259798873851	1
71	0.770118538251249	0.3534970339513929	0.5049957627877042	0.16412362290600388	1
72	0.7414330662236145	0.29421947072365656	0.5766701626183669	0.17653168243419393	1
73	0.7365989486022553	0.33811099280103524	0.5675434522017377	0.14490471120044368	2
74	0.7674169845534854	0.34773582112579804	0.515608286496873	0.15588157498742672	1

Tabela com Matriz de Features e Resultados da Classificação

```
1  from imblearn.metrics import classification_report_imbalanced
2  from sklearn.externals import joblib
3  from sklearn.naive_bayes import GaussianNB
4  from sklearn import datasets, model_selection
5  from sklearn.preprocessing import Normalizer
6  import numpy as np
7  import sklearn
8
9  print (sklearn.__version__)
10
11 test_size = 0.25
12 transformer = Normalizer()
13 iris = datasets.load_iris()
14 X = iris.data
15 X = transformer.fit_transform(X)
16 y = iris.target
17 X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=10)
18 #Classificador com o melhor resultado em termos de F1
19 model = GaussianNB()
20 model.fit(X_train, Y_train)
21 print('Salvando Componente de Inteligência...')
22 #save model
23 filename = 'iris.ml.gzip'
24 joblib.dump(model, filename)
25 print('Modelo Salvo: iris.ml.gzip')
26
27 loaded_model = joblib.load(filename)
28 predictions = loaded_model.predict(X)
29 '''
30 y_pred é o resultado da predicao. Em produção, bastaria colocar no lugar do X o vetor
31 a ser classificado e o y_pred seria a resposta do classificador.
32 '''
33 y_pred = predictions.reshape(len(predictions),1)
34 #Verifica Metrica do Modelo Salvo
35 metricas = classification_report_imbalanced(y, y_pred)
36 print(metricas)
37 np.savetxt('resultados_iris_modelo_salvo.csv',predictions,delimiter=',',fmt='%.1i')
```

Para utilizarmos o classificador criado, bastaria carregarmos a linha 27 acima e depois inserir a matriz de features do dataset no lugar de X. As métricas são calculadas na linha 35 e o arquivo de resultados é salvo na linha 37.

Exercício - Criar um classificador de padrões para o dataset (https://archive.ics.uci.edu/ml/datasets/Activity+recognition+with+healthy+older+people+using+a+batteryless+wearable+sensor).