

## Cheat Sheet : SciKit Learn

Para SciKit learn os dados precisam ser numéricos e para isso, utiliza-se "numpy" ou "scipy" para serem armazenados como matrizes.

### PRE-PROCESSAMENTO

#### TRAIN-TEST DATA

```
from sklearn.model_selection  
import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_  
state=0)
```

#### PREPARAÇÃO DE DADOS

```
from sklearn.preprocessing  
import StandardScaler
```

```
get_names = df.columns
```

```
scaler = preprocessing.StandardScaler()
```

```
scaled_df = scaler.fit_transform(df)
```

```
scaled_df = pd.DataFrame(scaled_df, columns=get_names)
```

" "

Padronização

```
from sklearn.preprocessing  
import Normalizer
```

```
pd.read_csv("File-name.csv")
```

```
x_array = np.array(df['Column'])
```

```
normalized_X = preprocessing.normalize([x_array])
```

Normalização

Opcão de Normaliza-  
ção L<sub>1</sub> e L<sub>2</sub>.

É usada para dimen-  
sionar o conjunto de  
dados de entrada em  
uma escala de 0 a 1  
aplicar a norma da  
unidade.

#### STANDARD SCALER:

Calcula a média e  
o desvio padrão  
em um conjunto de  
treinamento, e depois  
pode replicar a  
mesma transforma-  
ção no conj. teste.

# TRABALHANDO NO MODELO

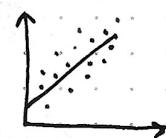
## Aprendizado Supervisionado

### Regressão Linear

```
from sklearn.linear_model
import LinearRegression
```

```
new_lr = LinearRegression(normalize=True)
```

**REGRESSÃO LINEAR:**  
Ajusta um modelo linear com coeficientes  $w$  para minimizar a soma residual dos quadrados entre os valores observados no conjunto e os previstos pela approximação linear.



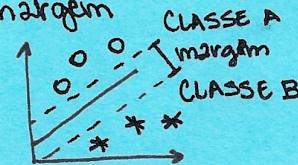
### Support Vector Machines

```
from sklearn.svm
import SVC
```

```
new_svc = SVC(kernel='linear')
```

### SVM:

Separar os pontos de dados usando um hiperplano com a maior quantidade de margem



### Naive Bayes

```
from sklearna.naive_bayes
import GaussianNB
```

```
new_gnb = GaussianNB()
```

### KNN

```
from sklearn
import neighbors
```

```
Knn = neighbors.KNeighborsClassifier(n_neighbors=1)
```

### FÓRMULA:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma^2}\right)$$

→ CLASSIFICADOR

→ Aplica após o treinamento.

## Aprendizado Não Supervisionado

### Principal Component Analysis (PCA)

```
from sklearn.decomposition
import PCA
```

```
new_pca = PCA(n_components=0.95)
```

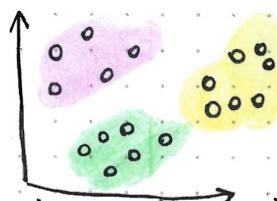
### K-Means:

```
from sklearn.cluster
import KMeans
```

```
Kmeans = KMeans(n_clusters=5, random_state=0)
```

### TCA:

Redução de dimensionalidade usando decomposição de valor singular para projetá-la para um espaço dimensional inferior



## TRAIN - TEST DATA SUPERVISIÃO

new\_lr.fit(X, y)

Knn.fit(X\_train, y\_train)

new\_svc.fit(X\_train, y\_train)

## TRAIN - TEST DATA NÃO SUPERVISIÃO

KMeans.fit(X\_train)

tpca\_model\_fit = new\_tpca.fit\_transform(X\_train)

PÓS

PROCESSAMENTO

## PREDIÇÃO

### Supervisionado

y\_predict = new\_svc.predict(np.random((3, 5)))

y\_predict = new\_lr.predict(X\_test)

y\_predict = Knn.predict\_proba(X\_test)

### Não Supervisionado

y\_pred = KMeans.predict(X\_test)

## AJUSTE DO MODELO

### Pesquisa de grade (Grid Search)

```
from sklearn.grid_search  
import GridSearchCV
```

```
params = {"n_neighbors": np.arange(1, 3), "metric":  
["euclidean", "cityblock"]}
```

```
grid = GridSearchCV(estimator=Knn, param_grid=params)
```

```
grid.fit(X_train, y_train)
```

```
print(grid.best_score_)
```

```
print(grid.best_estimator_.n_neighbors)
```

### GRID SEARCH:

Técnica de ajuste que tenta calcular os valores ótimos de hiperparâmetros.  
Como funciona?  
São otimizados por pesquisa de grade de validação cruzada sobre uma grade de hiper-

para-metros

## Otimização de Parâmetros Aleatórios (Randomized Parameter Optimization)

```
from sklearn.grid_search  
import RandomizedSearchCV  
  
params = {"n_neighbors": range(1, 5), "weights":  
          ["uniform", "distance"]}  
  
rsearch = RandomizedSearchCV(estimator=Knn,  
                             param_distributions=params, cv=4, n_iter=8,  
                             random_state=5)  
  
rsearch.fit(X_train, y_train)  
print(rsearch.best_score_)
```

Métodos implementados:

- fit
- score
- score\_samples
- predict
- transform
- predict\_proba
- decision\_function
- inverse\_transform

Esse métodos são otimizados por pesquisa de validação cruzada

## AVALIANDO A PERFORMANCE

### CLASSIFICAÇÃO

#### ① Matriz de Confusão

```
from sklearn.metrics  
import confusion_matrix  
  
print(confusion_matrix(y_test, y_pred))
```

Aqui, já utiliza os testes para avaliar a performance do modelo!

#### ② Acurácia

```
Knn.score(X_test, y_test)
```

```
from sklearn.metrics  
import accuracy_score
```

```
accuracy_score(y_test, y_pred)
```

## REGRESSÃO

### ① Média do erro absoluto (Mean Absolute Error)

```
from sklearn.metrics  
import mean_absolute_error
```

```
y_true = [3, -0.5, 2]
```

```
mean_absolute_error(y_true, y_predict)
```

### ② Média do erro quadrático (Mean squared error)

↳ também conhecido como: "risco quadrático")

```
from sklearn.metrics  
import mean_squared_error
```

```
mean_squared_error(y_true, y_predict)
```

### ③ R<sup>2</sup> Score

```
from sklearn.metrics  
import r2_score
```

```
r2_score(y_true, y_predict)
```

## CLUSTERING

### ① Homogeneidade.

```
from sklearn.metrics  
import homogeneity_score
```

```
homogeneity_score(y_true, y_predict)
```

### ② V-measure

```
from sklearn.metrics  
import v_measure_score
```

```
metrics.v_measure_score(y_true, y_predict)
```

R<sup>2</sup>-Score:

conhecido como coeficiente de determinação.

é uma medida de ajuste de um modelo estatístico linear generalizado.

Varia entre 0 e 1.

## CROSS-VALIDATION

```
from sklearn.cross_validation  
import cross_val_score.
```

```
print(cross_val_score(Knn,x_train,y_train,cv=4))
```

```
print(cross_val_score(new_lr,X,y_cv=2))
```