Redes Neurais Artificiais - IFES - PPCOMP

Exercicio 01

Implementação do Perceptron

Validação com Dataset Breast Cancer - Comparação com outros Classificadores (*)

(*) Utilizada a implementação do PerformanceEvaluator desenvolvido na disciplina de Reconhecimento de Padrões

In [49]:
```
import time
import sklearn
import numpy as np

from sklearn.base import BaseEstimator,ClassifierMixin

from sklearn.datasets import load_breast_cancer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.metrics import mean_squared_error

# Classificadores
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

In [50]:
```
print('Versão do scikit-learn {}.'.format(sklearn.__version__))
```

```
Versão do scikit-learn 0.21.2.
```

In [51]:
```
# Datasets
dX_AllDatasets={}
dy_AllDatasets={}

# Breast Cancer
data = load_breast_cancer()
X,y = data.data,data.target
dX_AllDatasets['breast_cancer']=X
dy_AllDatasets['breast_cancer']=y

print(X.shape, y.shape)
```

```
(569, 30) (569,)
```

In [52]:
```
class PerceptronPPCOMPClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self):
        return

    def predict(self, X):
        r = np.dot(X, self.w) + self.b
        if np.isscalar(r):
            if r>=0.0:
                return 1.0
            else:
                return 0.0
        else:
            for i in range(len(r)):
                if r[i]>=0.0:
                    r[i]=1.0
                else:
                    r[i]=0.0
            return r

    def fit(self, X, y, e=100,learn_r=0.001):
        # Inicializa pesos (w) e bias (b)

        # Inicializacao com Zeros (0)
        #self.w = np.zeros((X.shape[1], )) # X.shape[1] = total de caracteristic
        #self.b = 0.0

        # Inicialização com valores aleatorios
        #self.w = np.random.normal(size=X.shape[1])
        self.w = np.random.random((X.shape[1], ))
        self.b = np.random.random()

        for f in range(e):
            error_conv = 0 # avaliar convergencia
            for xi, yi in zip(X, y):
                err = yi - self.predict(xi)
                if err != 0:
                    self.w += learn_r*err*xi # w <- w + α(y − f(x))x
                    self.b += learn_r*err
                    error_conv+=1
            if error_conv == 0:
                break
        return self
```

In [53]:
```
class PerformanceEvaluator():
  def __init__(self, X, y,cv,scaler):
    self.X=X
    self.y=y
    self.cv=cv
    self.scaler=scaler
  def score(self, pipe):
    scores=cross_val_score(pipe, self.X,self.y, cv=self.cv) # (Stratified)KFold
    return scores
  def evaluate(self, clfs):
    best_overal=0
    for name,clf in clfs:
        if self.scaler==True:
            pipe = Pipeline(steps=[('scaler', StandardScaler()),
                   ('classifier', clf)])
        else:
            pipe = clf
        t_inicio = time.time()
        scores=self.score(pipe)
        t_fim = time.time()
        print('Mean: %0.7f Std: %0.7f(+/-) Best: %0.7f Time: %.2f(s) [%s]' % (sc
        if (scores.mean()>best_overal):
            best_overal=scores.mean()
            best_pipe=pipe
            best_clf_name=name
    print('Best Estimator: ',best_clf_name)
    ### Matriz de Confusão ilustrativa para o melhor estimator
    X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, test_siz
    best_pipe.fit(X_train,y_train)
    y_p=best_pipe.predict(X_test)
    conf_mat = confusion_matrix(y_test,y_p)
    print(conf_mat)
```

In [54]:
```
print ("Avaliação de Multiplos Classificadores x implementação do Perceptron (Per

# Classificadores de interesse com respectivos hyper-parametros
clfs = [
    ('PerceptronPPCOMPClassifier',PerceptronPPCOMPClassifier()),
    ('RandomForestClassifier',RandomForestClassifier(100)),
    ('KNeighborsClassifier',KNeighborsClassifier(n_neighbors=3)),
    ('MLP',MLPClassifier(max_iter=500,early_stopping=True,hidden_layer_sizes=(10
]

### Parametros complementaras ###
# cross-validation folds
cv = 5
# habilita ou nao scaler (standard scaler)
scaler = False
#################################

for key in dX_AllDatasets.keys():
    print("\n" +"="*40)
    print(key)
    print("-"*40)
    X,y=dX_AllDatasets[key],dy_AllDatasets[key]
    pe = PerformanceEvaluator(X,y,cv,scaler)
    pe.evaluate(clfs)
```

```
Avaliação de Multiplos Classificadores x implementação do Perceptron (Perceptro
nPPCOMPClassifier)

========================================
breast_cancer
----------------------------------------
Mean: 0.8559446 Std: 0.0284887(+/-) Best: 0.8938053 Time: 0.86(s) [PerceptronPP
COMPClassifier]
Mean: 0.9632628 Std: 0.0207368(+/-) Best: 0.9911504 Time: 0.46(s) [RandomForest
Classifier]
Mean: 0.9192920 Std: 0.0239354(+/-) Best: 0.9469027 Time: 0.02(s) [KNeighborsCl
assifier]
Mean: 0.8855406 Std: 0.0312450(+/-) Best: 0.9217391 Time: 0.25(s) [MLP]
Best Estimator:  RandomForestClassifier
[[43  1]
 [ 4 66]]
```

In [ ]: