Redes Neurais Artificiais - IFES - PPCOMP

Exercicio 02

Comparação de Redes Neurais Rasas em Múltiplos Datasets

Perceptron (Atividade 1), Perceptron SciKit, MLP (1 Hidden Layer), Linear SVM, SGD (Hinge Loss)

Datasets: Breast Cancer, Dummy datasets (*)

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

In [72]:
```
import time
import sklearn
import numpy as np

from sklearn.base import BaseEstimator,ClassifierMixin

from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_digits
from sklearn.datasets import make_classification

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.neural_network import MLPClassifier
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
```

In [73]:
```
print('Versão do scikit-learn {}.'.format(sklearn.__version__))
```

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

In [78]:
```
# Datasets Binários
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

# Dummy Dataset 1 - sklearn.datasets.make_classification
# One informative feature, one cluster per class
X, y = make_classification(n_samples=1000,n_features=2, n_redundant=0, n_informat
                             n_clusters_per_class=1)
dX_AllDatasets['dummy_ds_1']=X
dy_AllDatasets['dummy_ds_1']=y

# Dummy Dataset 2 - sklearn.datasets.make_classification
# Two informative features, one cluster per class
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                             n_clusters_per_class=1)
dX_AllDatasets['dummy_ds_2']=X
dy_AllDatasets['dummy_ds_2']=y

# Dummy Dataset 3 - sklearn.datasets.make_classification
# Two informative features, two clusters per class
X, Y = make_classification(n_features=2, n_redundant=0, n_informative=2)
dX_AllDatasets['dummy_ds_3']=X
dy_AllDatasets['dummy_ds_3']=y

# Dummy Dataset 4 - sklearn.datasets.make_classification
# 10.000 Samples com 10% de "ruído"
X, y = make_classification(
    n_samples=10000,
    n_features=25,
    flip_y=0.1)
dX_AllDatasets['dummy_ds_4_10000_10_noise']=X
dy_AllDatasets['dummy_ds_4_10000_10_noise']=y

# Dummy Dataset 5 - sklearn.datasets.make_classification
# 10.000 Samples - Difícil separação
X, y = make_classification(
    n_samples=10000,
    n_features=25,
    class_sep=0.1) # class_sep padrão=1.0. Menor o valor, mais dificil a classif
dX_AllDatasets['dummy_ds_5_10000_hard_sep']=X
dy_AllDatasets['dummy_ds_5_10000_hard_sep']=y


# Dummy Dataset 6 - sklearn.datasets.make_classification
# 5.000 Samples - Ajuste na contribuição das features
X, y = make_classification(n_samples=5000,
    n_features=25,
    n_redundant=10, # 10 das 25 features serão combinações das outras
    n_repeated=5) # e 5 das 25 serão duplicadas
dX_AllDatasets['dummy_ds_6_5000_feat_contrib']=X
```

```
dy_AllDatasets['dummy_ds_6_5000_feat_contrib']=y
```

In [79]:
```
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 [80]:
```
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 [81]:
```
print ("Comparativo de Redes Neurais Rasas com multiplos datasets")

# Classificadores de interesse com respectivos hyper-parametros
clfs = [
    ('PerceptronPPCOMP',PerceptronPPCOMPClassifier()),
    ('PerceptronSciKit',Perceptron(tol=1e-3, random_state=0)),
    ('LinearSVM',SVC(kernel="linear", C=0.025)),
    ('SGD_LossHinge',SGDClassifier(loss='hinge',max_iter=1000, tol=1e-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)
```

```
Comparativo de Redes Neurais Rasas com multiplos datasets

========================================
breast_cancer
----------------------------------------
Mean: 0.8719969 Std: 0.0425969(+/-) Best: 0.9292035 Time: 0.90(s) [PerceptronPP
COMP]
Mean: 0.8025702 Std: 0.1697021(+/-) Best: 0.9217391 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.9491343 Std: 0.0267773(+/-) Best: 0.9911504 Time: 0.17(s) [LinearSVM]
Mean: 0.9068103 Std: 0.0300324(+/-) Best: 0.9292035 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.8290881 Std: 0.1041403(+/-) Best: 0.9217391 Time: 0.24(s) [MLP]
Best Estimator:  LinearSVM
[[31  3]
 [ 4 76]]

========================================
dummy_ds_1
----------------------------------------
Mean: 0.9929648 Std: 0.0098472(+/-) Best: 1.0000000 Time: 1.39(s) [PerceptronPP
COMP]
Mean: 0.9879698 Std: 0.0081538(+/-) Best: 1.0000000 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.9909748 Std: 0.0092091(+/-) Best: 1.0000000 Time: 0.01(s) [LinearSVM]
Mean: 0.9959899 Std: 0.0058586(+/-) Best: 1.0000000 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.9799647 Std: 0.0170605(+/-) Best: 1.0000000 Time: 0.26(s) [MLP]
Best Estimator:  SGD_LossHinge
```

```
[[102   0]
 [  1  97]]

========================================
dummy_ds_2
----------------------------------------
Mean: 0.9700000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.17(s) [PerceptronPP
COMP]
Mean: 0.9800000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.00(s) [PerceptronSc
iKit]
Mean: 0.9800000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.00(s) [LinearSVM]
Mean: 0.9800000 Std: 0.0400000(+/-) Best: 1.0000000 Time: 0.00(s) [SGD_LossHing
e]
Mean: 0.7900000 Std: 0.2437212(+/-) Best: 1.0000000 Time: 0.07(s) [MLP]
Best Estimator:  PerceptronSciKit
[[10  0]
 [ 0 10]]

========================================
dummy_ds_3
----------------------------------------
Mean: 0.5400000 Std: 0.1019804(+/-) Best: 0.7000000 Time: 0.22(s) [PerceptronPP
COMP]
Mean: 0.5200000 Std: 0.0812404(+/-) Best: 0.6000000 Time: 0.00(s) [PerceptronSc
iKit]
Mean: 0.4300000 Std: 0.0871780(+/-) Best: 0.5500000 Time: 0.01(s) [LinearSVM]
Mean: 0.3800000 Std: 0.0748331(+/-) Best: 0.4500000 Time: 0.00(s) [SGD_LossHing
e]
Mean: 0.4700000 Std: 0.0927362(+/-) Best: 0.5500000 Time: 0.06(s) [MLP]
Best Estimator:  PerceptronPPCOMP
[[ 1 10]
 [ 1  8]]

========================================
dummy_ds_4_10000_10_noise
----------------------------------------
Mean: 0.7925000 Std: 0.0288184(+/-) Best: 0.8300000 Time: 17.09(s) [PerceptronP
PCOMP]
Mean: 0.7817000 Std: 0.0326046(+/-) Best: 0.8280000 Time: 0.03(s) [PerceptronSc
iKit]
Mean: 0.8917000 Std: 0.0051049(+/-) Best: 0.8995000 Time: 3.53(s) [LinearSVM]
Mean: 0.8840000 Std: 0.0027203(+/-) Best: 0.8885000 Time: 0.18(s) [SGD_LossHing
e]
Mean: 0.8883000 Std: 0.0031401(+/-) Best: 0.8935000 Time: 2.02(s) [MLP]
Best Estimator:  LinearSVM
[[925 104]
 [116 855]]

========================================
dummy_ds_5_10000_hard_sep
----------------------------------------
Mean: 0.5156000 Std: 0.0173764(+/-) Best: 0.5400000 Time: 21.60(s) [PerceptronP
PCOMP]
Mean: 0.5084000 Std: 0.0030887(+/-) Best: 0.5135000 Time: 0.04(s) [PerceptronSc
iKit]
Mean: 0.5603000 Std: 0.0074606(+/-) Best: 0.5720000 Time: 9.11(s) [LinearSVM]
Mean: 0.5246000 Std: 0.0060778(+/-) Best: 0.5330000 Time: 0.20(s) [SGD_LossHing
```

```
e]
Mean: 0.6095000 Std: 0.0046260(+/-) Best: 0.6155000 Time: 4.50(s) [MLP]
Best Estimator:  MLP
[[540 453]
 [354 653]]

========================================
dummy_ds_6_5000_feat_contrib
----------------------------------------
Mean: 0.9123974 Std: 0.0127032(+/-) Best: 0.9310689 Time: 8.19(s) [PerceptronPP
COMP]
Mean: 0.8923936 Std: 0.0120148(+/-) Best: 0.9110889 Time: 0.02(s) [PerceptronSc
iKit]
Mean: 0.9369968 Std: 0.0087616(+/-) Best: 0.9480519 Time: 0.50(s) [LinearSVM]
Mean: 0.9275944 Std: 0.0146409(+/-) Best: 0.9440000 Time: 0.08(s) [SGD_LossHing
e]
Mean: 0.9455982 Std: 0.0048626(+/-) Best: 0.9520000 Time: 1.66(s) [MLP]
Best Estimator:  MLP
[[520   7]
 [ 55 418]]
```

In [82]:
```
### Ativando StandardScaler ###
scaler = True
#################################

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)
```

```
========================================
breast_cancer
----------------------------------------
Mean: 0.9701578 Std: 0.0068991(+/-) Best: 0.9823009 Time: 0.88(s) [PerceptronPP
COMP]
Mean: 0.9666795 Std: 0.0148693(+/-) Best: 0.9823009 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.9718969 Std: 0.0065201(+/-) Best: 0.9823009 Time: 0.02(s) [LinearSVM]
Mean: 0.9718969 Std: 0.0128707(+/-) Best: 0.9826087 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.9279877 Std: 0.0231880(+/-) Best: 0.9734513 Time: 0.19(s) [MLP]
Best Estimator:  LinearSVM
[[42  5]
 [ 0 67]]

========================================
dummy_ds_1
----------------------------------------
Mean: 0.9899797 Std: 0.0095189(+/-) Best: 1.0000000 Time: 1.50(s) [PerceptronPP
COMP]
Mean: 0.9899797 Std: 0.0071138(+/-) Best: 1.0000000 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.9909748 Std: 0.0092091(+/-) Best: 1.0000000 Time: 0.02(s) [LinearSVM]
Mean: 0.9949749 Std: 0.0077849(+/-) Best: 1.0000000 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.9639341 Std: 0.0248987(+/-) Best: 0.9950000 Time: 0.22(s) [MLP]
Best Estimator:  SGD_LossHinge
[[ 93   0]
 [  0 107]]

========================================
dummy_ds_2
----------------------------------------
Mean: 0.9800000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.17(s) [PerceptronPP
COMP]
Mean: 0.9800000 Std: 0.0400000(+/-) Best: 1.0000000 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.9800000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.01(s) [LinearSVM]
Mean: 0.9800000 Std: 0.0244949(+/-) Best: 1.0000000 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.8800000 Std: 0.0927362(+/-) Best: 1.0000000 Time: 0.08(s) [MLP]
Best Estimator:  PerceptronPPCOMP
[[ 9  0]
 [ 1 10]]
```

```
========================================
dummy_ds_3
----------------------------------------
Mean: 0.4900000 Std: 0.1019804(+/-) Best: 0.6000000 Time: 0.22(s) [PerceptronPP
COMP]
Mean: 0.4600000 Std: 0.0734847(+/-) Best: 0.5500000 Time: 0.01(s) [PerceptronSc
iKit]
Mean: 0.4300000 Std: 0.1166190(+/-) Best: 0.6000000 Time: 0.01(s) [LinearSVM]
Mean: 0.5500000 Std: 0.0894427(+/-) Best: 0.7000000 Time: 0.01(s) [SGD_LossHing
e]
Mean: 0.5100000 Std: 0.0734847(+/-) Best: 0.6500000 Time: 0.05(s) [MLP]
Best Estimator:  SGD_LossHinge
[[5 6]
 [3 6]]

========================================
dummy_ds_4_10000_10_noise
----------------------------------------
Mean: 0.7917000 Std: 0.0234128(+/-) Best: 0.8235000 Time: 16.83(s) [PerceptronP
PCOMP]
Mean: 0.7792000 Std: 0.0399757(+/-) Best: 0.8315000 Time: 0.07(s) [PerceptronSc
iKit]
Mean: 0.8913000 Std: 0.0052115(+/-) Best: 0.8990000 Time: 3.88(s) [LinearSVM]
Mean: 0.8819000 Std: 0.0013928(+/-) Best: 0.8835000 Time: 0.21(s) [SGD_LossHing
e]
Mean: 0.8893000 Std: 0.0024207(+/-) Best: 0.8940000 Time: 2.18(s) [MLP]
Best Estimator:  LinearSVM
[[899 103]
 [122 876]]

========================================
dummy_ds_5_10000_hard_sep
----------------------------------------
Mean: 0.5203000 Std: 0.0133214(+/-) Best: 0.5365000 Time: 21.95(s) [PerceptronP
PCOMP]
Mean: 0.5066000 Std: 0.0221120(+/-) Best: 0.5350000 Time: 0.08(s) [PerceptronSc
iKit]
Mean: 0.5632000 Std: 0.0084356(+/-) Best: 0.5765000 Time: 9.56(s) [LinearSVM]
Mean: 0.5200000 Std: 0.0137186(+/-) Best: 0.5395000 Time: 0.24(s) [SGD_LossHing
e]
Mean: 0.6138000 Std: 0.0130752(+/-) Best: 0.6350000 Time: 3.33(s) [MLP]
Best Estimator:  MLP
[[491 530]
 [287 692]]

========================================
dummy_ds_6_5000_feat_contrib
----------------------------------------
Mean: 0.9065958 Std: 0.0191927(+/-) Best: 0.9330669 Time: 8.17(s) [PerceptronPP
COMP]
Mean: 0.8513896 Std: 0.0731813(+/-) Best: 0.9310689 Time: 0.03(s) [PerceptronSc
iKit]
Mean: 0.9359968 Std: 0.0086710(+/-) Best: 0.9470000 Time: 0.52(s) [LinearSVM]
Mean: 0.9287934 Std: 0.0140301(+/-) Best: 0.9470000 Time: 0.10(s) [SGD_LossHing
e]
Mean: 0.9405946 Std: 0.0134164(+/-) Best: 0.9580420 Time: 1.62(s) [MLP]
```

```
Best Estimator:  MLP
[[482   9]
 [ 37 472]]
```

Observaçoes sobre o experimento:

- Para alguns datasets (Ex: Breast Cancer) a normalização dos dados trouxe uma melhoria significativa.
- Notória a vantagem da MLP (1 hidden layer) contra os classificadores lineares em um cenário de difício separação (Dummy Dataset 5)

SGD (Hinge Loss) x Linear SVM

Pelo meu entendimento o SGDClassifer é um otimizador para classificadores lineares utilizando o SGD. Por padrão ele otimiza uma SVM Linear com a função de custo Hinge. Com o uso de uma função de custo do tipo log, por exemplo, otimizaria uma regressão logística. Outro ponto é que o mesmo faz uso de mini-batches.

> This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning via the partial_fit method...The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).

Interessante observar que no experimento o SGD não obteve sucesso na otimização em muitos casos com os parametros escolhidos. O tunning destes parâmetros não foi objeto de análise pelo menos nesta primeira experimentação.

In [ ]: