WisardClassifier Documentation

Release beta

Maurizio Giordano

Mar 06, 2018

CONTENTS:

class wis.**WisardClassifier**(*n_bits=8*, *n_tics=256*, *mapping='random'*, *debug=False*, *bleaching=True*, *default_bleaching=1*, *confidence_bleaching=0.01*, *n_jobs=1*, *random_state=0*)

Wisard Classifier.

This model uses the WiSARD weightless neural network. WiSARD stands for "Wilkie, Stonham, Aleksander Recognition Device". It is a weightless neural network model to recognize binary patterns. For a introduction to WiSARD, please read a brief introduction to weightless neural network (https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2009-6.pdf)

n_bits [int, optional, default 8] number of bits used in n-tuple extraction from input (network resolution), should be in [1, 32]

n_tics [int, optional, default 256] datum scaling factor (e.g. max discretization value) high values slow down system perfromance

mapping [{'linear', 'random'}, optional, default 'random'] input to neurons mapping

bleaching [bool, optional, default True] enable bleaching algorithm to solve classification ties

default_bleaching [integer, optional, default 1] bleaching variable step

confidence_bleaching [floar, optional, default 0.01] bleaching confidence tie paramater, should be in range]1, 0]

n_jobs [integer, optional (default=1)] The number of jobs to run in parallel for both *fit* and *predict*. If -1, then the number of jobs is set to the number of cores. random_state : int, or 0, optional, default None If int, random_state is the seed used by the random number generator; If 0, the random number generator is the RandomState instance used

by *np.random*. debug : bool, optional, default True enable debugging

wiznet_ [dictionary] The set of WiSARD discriminators (one for each class)

nrams_ [int] The number of RAMs in each discriminato

nclasses_ [int] The number of classes

nfeatures_ [int] The number of features (variable) in the datum

ranges_ [array of shape = [**nfeatures_**]] The range of features (variables) in the datum

offsets_ [array of shape = [**nfeatures_**]] The offsets of features (variables) in the datum

classes_ [array of shape = [**nclasses_**]] The set of classes

npixels_ [int] The number of pixels in input binarized

progress_ [float] Progress bar monitoring step, default 0.0

starttm_ [int] Progress bar monitoring time starter

Here you find a simple example of using WisardClassifier in Python.

```
>>> from wis import WisardClassifier
>>> from sklearn.datasets import make_classification
>>>
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = WisardClassifier(n_bits=4, n_tics=128, debug=True, random_state=0)
>>> clf.fit(X, y)
train |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX| 100 %  00:00:00 00:00:00
WisardClassifier(bleaching=True, confidence_bleaching=0.01, debug=True,
default_bleaching=1, mapping='random', n_bits=4, n_jobs=1,
n_tics=128, random_state=0)
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

The default values for the parameters controlling the number of bits (‘’n_bits’‘) and the datum scaling range (‘’n_btics’‘) are set in order to have an averaged high accuracy on several classification problems. By using parallel computation you only affect classification stage. Model fitting does not exploit multcore yet. To obtain a deterministic behaviour during fitting, `random_state` has to be fixed. For more information, please read .. [1]

fit (*X*, *y*)

Fit the WiSARD model to data matrix X and target(s) y.

X : array-like or sparse matrix, shape (n_samples, n_features) The input data. y : array-like, shape (n_samples,) or (n_samples, n_outputs) The target values (class labels in classification, real numbers in regression).

self : returns a trained WiSARD model.

get_params (*deep=True*)

Get parameters for this estimator.

deep [boolean, optional] If True, will return the parameters for this estimator and contained subobjects that are estimators.

params [mapping of string to any] Parameter names mapped to their values.

predict (*X*)

Predict using the WiSARD model.

X [{array-like, sparse matrix}, shape (n_samples, n_features)] The input data.

y [array-like, shape (n_samples, n_outputs)] The predicted values.

set_params (***parameters*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it’s possible to update each component of a nested object.

self : returns the WiSARD model.

CHAPTER

ONE

WISARDCLASSIFIER'S USAGE IN SCIKIT LEARN

The following is an exampleof usage of WisardClassifier in the Scikit Learn library.

```
# import sklearn and scipy stuff
from sklearn.datasets import load_svmlight_file
from sklearn import cross_validation
import scipy.sparse as sps
from scipy.io import arff
# import wisard classifier library
from wis import WisardClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from utilities import *
import time

# (Try) import matplot for graphics
try:
    import matplotlib.pyplot as plt
    matplotfound = True
except ImportError:
    matplotfound = False
    pass

B_enabled = True
# IRIS (arff) - load datasets
data, meta = arff.loadarff(open("datasets/iris.arff", "r"))
y_train = np.array(data['class'])
X_train = np.array([list(x) for x in data[meta._attrnames[0:-1]]])
X_train = X_train.toarray() if sps.issparse(X_train) else X_train  # avoid sparse data
class_names = np.unique(y_train)
# IRIS (arff) - cross validation example
clf = WisardClassifier(n_bits=16,bleaching=B_enabled,n_tics=256,mapping='linear',
↪debug=True,default_bleaching=3)
kf = cross_validation.LeaveOneOut(len(class_names))
predicted = cross_validation.cross_val_score(clf, X_train, y_train, cv=kf, n_jobs=1)
print("Accuracy Avg: %.2f" % predicted.mean())

# IRIS (libsvm) - load datasets
X_train, y_train = load_svmlight_file(open("datasets/iris.libsvm", "r"))
class_names = np.unique(y_train)
X_train = X_train.toarray() if sps.issparse(X_train) else X_train  # avoid sparse data
# IRIS - cross validation example (with fixed seed)
clf = WisardClassifier(n_bits=16,n_tics=1024,debug=True,bleaching=B_enabled,random_
↪state=848484848)
kf = cross_validation.StratifiedKFold(y_train, 10)
predicted = cross_validation.cross_val_score(clf, X_train, y_train, cv=kf, n_jobs=1,␣
↪verbose=0)
```

```
print("Accuracy Avg: %.2f" % predicted.mean())

# DNA (libsvm) - load datasets
X_train, y_train = load_svmlight_file(open("datasets/dna.tr", "r"))
X_train = X_train.toarray() if sps.issparse(X_train) else X_train  # avoid sparse data
class_names = np.unique(y_train)
X_test, y_test = load_svmlight_file(open("datasets/dna.t", "r"))
X_test = X_test.toarray() if sps.issparse(X_test) else X_test  # avoid sparse data

# DNA (arff) - testing example
clf = WisardClassifier(n_bits=16,n_tics=512,debug=True,bleaching=B_enabled,random_
↪state=848484848,n_jobs=-1)
y_pred = clf.fit(X_train, y_train)
tm = time.time()
y_pred = clf.predict(X_test)
print("Time: %d"%(time.time()-tm))
predicted = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy: %.2f" % predicted)

# DNA - plot (print) confusion matrix
if matplotfound:
    plt.figure()
    plot_confusion_matrix(cm, classes=class_names,title='Confusion matrix')
    plt.show()
else:
    print_confmatrix(cm)
```

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search