Practica 5 Clasificación usando método multiclase

February 10, 2021

0.0.1 Ejercicio 1: Seleccione un algoritmo clasificación de los disponibles en scikit que sea capaz de resolver problemas de más de dos clases y al menos 10 conjuntos de datos de mñas de 2 clases (puede reusar las prácticas anteriores)

[ ]:
```
# -*- coding: utf-8 -*-
from scipy.io import arff
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
from os import listdir
from sklearn.svm import LinearSVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.multiclass import OutputCodeClassifier
import warnings
warnings.filterwarnings('ignore')

lista_datasets=listdir('./Datos/P5/')

def base():

    print('CLASIFICADOR BASE LINEARSVC\n')

    for indice in lista_datasets:
        print('\n\tBase de datos: ' + str(indice))
        dataset = arff.loadarff('./Datos/P5/' + str(indice))
        df = pd.DataFrame(dataset[0])
        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]

        X_train, X_test, Y_train, Y_test = train_test_split(input, output,␣
↪test_size=0.25)
        # llamada y entrenamiento algoritmo SVM

        svc = LinearSVC(random_state=0, tol=1e-5, max_iter=500)

        svc.fit(X_train, Y_train)
```

```
        print('\n\t\t\tPorcentaje de bien clasificados LINEARSVC: '+str(svc.
↪score(X_test, Y_test)))



    print('-------------------------\n')

def OVA_OVO(param):

    print('Aplicando metodo '+param)

    for i in lista_datasets:
        print('\n\tBase de datos: ' + str(i))
        dataset = arff.loadarff('./Datos/P5/' + str(i))
        df = pd.DataFrame(dataset[0])
        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]

        X_train, X_test, Y_train, Y_test = train_test_split(input, output,␣
↪test_size=0.25)

        kernel = ( 1.0 * RBF(1.0) )

        gpc = GaussianProcessClassifier(kernel=kernel, random_state=0,␣
↪multi_class=param)
        gpc.fit(X_train, Y_train)
        print('\n\t\t\tPorcentaje de bien clasificados '+param+':'+str(gpc.
↪score(X_test, Y_test)))


    print('-------------------------')

def ECOC():

    print('Aplicando metodo multiclase ERROR CORRECTING OUTPUT CODES')
    for indice in lista_datasets:

        print('\n\tBase de datos: ' + str(indice))
        dataset = arff.loadarff('./Datos/P5/' + str(indice))
        df = pd.DataFrame(dataset[0])
        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]
        X_train, X_test, Y_train, Y_test = train_test_split(input, output,␣
↪test_size=0.25)

        clf = OutputCodeClassifier(KNeighborsClassifier(n_neighbors=5),␣
↪code_size=2, random_state=0)
```

```
        clf.fit(X_train, Y_train)

        print('\n\t\t\tPorcentaje de bien clasificados ERROR CORRECTING OUTPUT␣
↪CODES: '+str(clf.score(X_test, Y_test)))
    print('--------------------------')
```

0.0.2 Ejercicio2: Aplique el clasificador base a cada uno de los conjuntos y anote los resultados obtenidos.

[7]:
```
base()
```

```
CLASIFICADOR BASE LINEARSVC


        Base de datos: 0diabetes.arff

                        Porcentaje de bien clasificados LINEARSVC: 0.640625

        Base de datos: 1glass.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.3888888888888889

        Base de datos: 2ionosphere.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.8977272727272727

        Base de datos: 3ris.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.9473684210526315

        Base de datos: 4cpu.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.02666666666666667

        Base de datos: 5contactLenses.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.8333333333333334

        Base de datos: 6segment-challenge.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.7866666666666666
```

```
        Base de datos: 7segment-test.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.8275862068965517

        Base de datos: 8weather.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.3076923076923077

        Base de datos: 9iris.arff

                        Porcentaje de bien clasificados LINEARSVC:
0.9736842105263158
-------------------------
```

0.0.3 Ejercicio 3: Aplique los métodos multiclase one vs one (OVO), one vs all (OVA) y error correcting output codes (ECOC) a cada uno de los conjuntos de datos y anote los resultados obtenidos.

[11]:
```
OVA_OVO('one_vs_one')
```

```
Aplicando metodo one_vs_one

        Base de datos: 0diabetes.arff

                        Porcentaje de bien clasificados
one_vs_one:0.7552083333333334

        Base de datos: 1glass.arff

                        Porcentaje de bien clasificados
one_vs_one:0.8148148148148148

        Base de datos: 2ionosphere.arff

                        Porcentaje de bien clasificados
one_vs_one:0.8863636363636364

        Base de datos: 3ris.arff

                        Porcentaje de bien clasificados
one_vs_one:0.9736842105263158

        Base de datos: 4cpu.arff
```

```
                        Porcentaje de bien clasificados
one_vs_one:0.37333333333333335

        Base de datos: 5contactLenses.arff

                        Porcentaje de bien clasificados
one_vs_one:0.16666666666666666

        Base de datos: 6segment-challenge.arff

                        Porcentaje de bien clasificados
one_vs_one:0.9706666666666667

        Base de datos: 7segment-test.arff

                        Porcentaje de bien clasificados
one_vs_one:0.9310344827586207

        Base de datos: 8weather.arff

                        Porcentaje de bien clasificados
one_vs_one:0.9230769230769231

        Base de datos: 9iris.arff

                        Porcentaje de bien clasificados
one_vs_one:0.9736842105263158
--------------------------
```

[13]:
```
OVA_OVO('one_vs_rest')
```

```
Aplicando metodo one_vs_rest

        Base de datos: 0diabetes.arff

                        Porcentaje de bien clasificados
one_vs_rest:0.7708333333333334

        Base de datos: 1glass.arff

                        Porcentaje de bien clasificados
one_vs_rest:0.7592592592592593

        Base de datos: 2ionosphere.arff

                        Porcentaje de bien clasificados
one_vs_rest:0.8522727272727273
```

```
        Base de datos: 3ris.arff

                        Porcentaje de bien clasificados one_vs_rest:1.0

        Base de datos: 4cpu.arff

                        Porcentaje de bien clasificados
one_vs_rest:0.06666666666666667

        Base de datos: 5contactLenses.arff

                        Porcentaje de bien clasificados
one_vs_rest:0.8333333333333334

        Base de datos: 6segment-challenge.arff


      ␣
↪------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call␣
↪last)

        <ipython-input-13-4df446b33f94> in <module>
    ----> 1 OVA_OVO('one_vs_rest')


        <ipython-input-6-0b5ba0ed5832> in OVA_OVO(param)
         54
         55         gpc = GaussianProcessClassifier(kernel=kernel,␣
↪random_state=0, multi_class=param)
    ---> 56         gpc.fit(X_train, Y_train)
         57         print('\n\t\t\tPorcentaje de bien clasificados '+param+':
↪'+str(gpc.score(X_test, Y_test)))
         58


        C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪fit(self, X, y)
        655                                     % self.multi_class)
        656
    --> 657         self.base_estimator_.fit(X, y)
        658
        659         if self.n_classes_ > 2:


        C:\Anaconda3\lib\site-packages\sklearn\multiclass.py in fit(self, X, y)
```

```
       239         # n_jobs > 1 in can results in slower performance due to the␣
↪overhead
       240         # of spawning threads.  See joblib issue #112.
   --> 241         self.estimators_ = Parallel(n_jobs=self.
↪n_jobs)(delayed(_fit_binary)(
       242             self.estimator, X, column, classes=[
       243                 "not %s" % self.label_binarizer_.classes_[i],


       C:\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self,␣
↪iterable)
      1030                 self._iterating = self._original_iterator is not None
      1031
   -> 1032             while self.dispatch_one_batch(iterator):
      1033                 pass
      1034


       C:\Anaconda3\lib\site-packages\joblib\parallel.py in␣
↪dispatch_one_batch(self, iterator)
       845                 return False
       846             else:
   --> 847                 self._dispatch(tasks)
       848                 return True
       849


       C:\Anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self,␣
↪batch)
       763         with self._lock:
       764             job_idx = len(self._jobs)
   --> 765             job = self._backend.apply_async(batch, callback=cb)
       766             # A job can complete so quickly than its callback is
       767             # called before we get here, causing self._jobs to


       C:\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in␣
↪apply_async(self, func, callback)
       206     def apply_async(self, func, callback=None):
       207         """Schedule a func to be run"""
   --> 208         result = ImmediateResult(func)
       209         if callback:
       210             callback(result)


       C:\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in␣
↪__init__(self, batch)
```

```
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):


    C:\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    250         # change the default number of processes to -1
    251         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 252             return [func(*args, **kwargs)
    253                     for func, args, kwargs in self.items]
    254


    C:\Anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    250         # change the default number of processes to -1
    251         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 252             return [func(*args, **kwargs)
    253                     for func, args, kwargs in self.items]
    254


    C:\Anaconda3\lib\site-packages\sklearn\multiclass.py in␣
↪_fit_binary(estimator, X, y, classes)
     79     else:
     80         estimator = clone(estimator)
---> 81         estimator.fit(X, y)
     82     return estimator
     83


    C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪fit(self, X, y)
    210
    211             # First optimize starting from theta specified in kernel
--> 212             optima = [self._constrained_optimization(obj_func,
    213                                                      self.kernel_.
↪theta,
    214                                                      self.kernel_.
↪bounds)]


    C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪_constrained_optimization(self, obj_func, initial_theta, bounds)
    441     def _constrained_optimization(self, obj_func, initial_theta,␣
↪bounds):
```

```
       442         if self.optimizer == "fmin_l_bfgs_b":
   --> 443             opt_res = scipy.optimize.minimize(
       444                 obj_func, initial_theta, method="L-BFGS-B", jac=True,
       445                 bounds=bounds)


       C:\Anaconda3\lib\site-packages\scipy\optimize\_minimize.py in␣
↪minimize(fun, x0, args, method, jac, hess, hessp, bounds, constraints, tol,␣
↪callback, options)
       615                                     **options)
       616     elif meth == 'l-bfgs-b':
   --> 617         return _minimize_lbfgsb(fun, x0, args, jac, bounds,
       618                                 callback=callback, **options)
       619     elif meth == 'tnc':


       C:\Anaconda3\lib\site-packages\scipy\optimize\lbfgsb.py in␣
↪_minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps,␣
↪maxfun, maxiter, iprint, callback, maxls, finite_diff_rel_step,␣
↪**unknown_options)
       358             # until the completion of the current minimization␣
↪iteration.
       359             # Overwrite f and g:
   --> 360             f, g = func_and_grad(x)
       361         elif task_str.startswith(b'NEW_X'):
       362             # new iteration


       C:\Anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.
↪py in fun_and_grad(self, x)
       198         if not np.array_equal(x, self.x):
       199             self._update_x_impl(x)
   --> 200         self._update_fun()
       201         self._update_grad()
       202         return self.f, self.g


       C:\Anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.
↪py in _update_fun(self)
       164     def _update_fun(self):
       165         if not self.f_updated:
   --> 166             self._update_fun_impl()
       167             self.f_updated = True
       168
```

```
      C:\Anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.
↪py in update_fun()
       71
       72         def update_fun():
  ---> 73             self.f = fun_wrapped(self.x)
       74
       75         self._update_fun_impl = update_fun


      C:\Anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.
↪py in fun_wrapped(x)
       68         def fun_wrapped(x):
       69             self.nfev += 1
  ---> 70             return fun(x, *args)
       71
       72         def update_fun():


      C:\Anaconda3\lib\site-packages\scipy\optimize\optimize.py in␣
↪__call__(self, x, *args)
       72     def __call__(self, x, *args):
       73         """ returns the the function value """
  ---> 74         self._compute_if_needed(x, *args)
       75         return self._value
       76


      C:\Anaconda3\lib\site-packages\scipy\optimize\optimize.py in␣
↪_compute_if_needed(self, x, *args)
       66         if not np.all(x == self.x) or self._value is None or self.
↪jac is None:
       67             self.x = np.asarray(x).copy()
  ---> 68             fg = self.fun(x, *args)
       69             self.jac = fg[1]
       70             self._value = fg[0]


      C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪obj_func(theta, eval_gradient)
      202             def obj_func(theta, eval_gradient=True):
      203                 if eval_gradient:
  --> 204                     lml, grad = self.log_marginal_likelihood(
      205                         theta, eval_gradient=True,␣
↪clone_kernel=False)
      206                     return -lml, -grad
```

```
       C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪log_marginal_likelihood(self, theta, eval_gradient, clone_kernel)
       360         # which can be reused for computing Z's gradient
       361         Z, (pi, W_sr, L, b, a) = \
   --> 362             self._posterior_mode(K, return_temporaries=True)
       363
       364         if not eval_gradient:


       C:\Anaconda3\lib\site-packages\sklearn\gaussian_process\_gpc.py in␣
↪_posterior_mode(self, K, return_temporaries)
       412             W_sr = np.sqrt(W)
       413             W_sr_K = W_sr[:, np.newaxis] * K
   --> 414             B = np.eye(W.shape[0]) + W_sr_K * W_sr
       415             L = cholesky(B, lower=True)
       416             # Line 6


       KeyboardInterrupt:
```

[14]:
```
ECOC()
```

```
Aplicando metodo multiclase ERROR CORRECTING OUTPUT CODES

       Base de datos: 0diabetes.arff

                      Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.7135416666666666

       Base de datos: 1glass.arff

                      Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.7222222222222222

       Base de datos: 2ionosphere.arff

                      Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.8295454545454546

       Base de datos: 3ris.arff

                      Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.9736842105263158

       Base de datos: 4cpu.arff

                      Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
```

```
CODES: 0.29333333333333333

        Base de datos: 5contactLenses.arff

                        Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.8333333333333334

        Base de datos: 6segment-challenge.arff

                        Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.912

        Base de datos: 7segment-test.arff

                        Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 0.9014778325123153

        Base de datos: 8weather.arff

                        Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 1.0

        Base de datos: 9iris.arff

                        Porcentaje de bien clasificados ERROR CORRECTING OUTPUT
CODES: 1.0
--------------------------
```

One-vs-rest: Se ajusta cada clasificador de proceso gaussiano para cada clase. Dicha clase será separada del resto.

One-vs-one: Se ajusta cada clasificador de proceso gaussiano para cada par de clases. Estas dos clases se separarán del resto. Este método no admitirá la predicción de estimaciones de probabilidad.

Error Correcting Output Codes (ECOC): Con este método se representa cada clase con un código binaro en una matriz de 0 y 1. Se ajustará un clasificador binario por bit en el 'libro de códigos' y en el momento de la predicción, los clasificadores proyectarán nuevos puntos en el espacio de clase eligidiendo la clase más cercana a los puntos.

0.0.4 Ejercicio 5: Compare si hay diferencias significativas entre ellos usando el test de Iman-Davenport. Si es así, aplique el procedimiento de wilcoxon para comparar cada método multiclase con el clasificador base y los diferentes métodos entre ellos.

[ ]:
```
Se obtiene un nan en el estadistico.
```

0.0.5 Ejercicio 7: Enuncue las conclusiones del estudio

Al realizar las pruebas el tiempo de procesamiento y espera se ha disparado y tambien se hace notar que la puntuación final no varia en gran medida al respecto de las practicas anteriores

[ ]: