Source code for metric_learn.nca

```
"""
Neighborhood Components Analysis (NCA)
Ported to Python from https://github.com/vomjom/nca
"""

from __future__ import absolute_import
import numpy as np
from six.moves import xrange

from .base_metric import BaseMetricLearner


class NCA(BaseMetricLearner):                                                    [docs]
  def __init__(self, max_iter=100, learning_rate=0.01):
    self.params = {
      'max_iter': max_iter,
      'learning_rate': learning_rate,
    }
    self.A = None

  def transformer(self):                                                         [docs]
    return self.A


  def fit(self, X, labels):                                                      [docs]
    """
    X: data matrix, (n x d)
    labels: scalar labels, (n)
    """
    n, d = X.shape
    # Initialize A to a scaling matrix
    A = np.zeros((d, d))
    np.fill_diagonal(A, 1./(X.max(axis=0)-X.min(axis=0)))

    # Run NCA
    dX = X[:,None] - X[None]  # shape (n, n, d)
    tmp = np.einsum('...i,...j->...ij', dX, dX)  # shape (n, n, d, d)
    masks = labels[:,None] == labels[None]
    learning_rate = self.params['learning_rate']
    for it in xrange(self.params['max_iter']):
      for i, label in enumerate(labels):
        mask = masks[i]
        Ax = A.dot(X.T).T  # shape (n, d)

        softmax = np.exp(-((Ax[i] - Ax)**2).sum(axis=1))  # shape (n)
        softmax[i] = 0
        softmax /= softmax.sum()

        t = softmax[:, None, None] * tmp[i]  # shape (n, d, d)
        d = softmax[mask].sum() * t.sum(axis=0) - t[mask].sum(axis=0)
        A += learning_rate * A.dot(d)

    self.X = X
    self.A = A
    return self
```