Искусственные нейронные сети: основы практического применения

Лекция 5

Крощенко А.А.

Брестский государственный технический университет

06.06.2017

Обсуждение домашнего задания

- Изучить выборку Forest Fires (`https://archive.ics.uci.edu/ml/datasets/Forest+Fires`). Решить регрессионную задачу на прогнозирование площади леса, поврежденного лесными пожарами. Сравнить результаты и обсудить применяемые модели на следующем занятии.
- Решить задачу кластеризации выборки Seeds Data Set с помощью нейронной сети Кохонена. Сравнить полученные результаты с результатами для алгоритма k-means.

Выборка Forest Fires

Forest Fires Data Set – пример очень сложной регрессионной задачи, первая попытка решить которую редко приводит к хорошему результату. Данные в этой выборке упорядочены в соответствии со следующими признаками:

1. X – пространственная координата на карте парка
2. Y – пространственная координата на карте парка
3. month – месяц года: 'jan' to 'dec'
4. day - день недели: 'mon' to 'sun'
5. FFMC – индекс FFMC (from the FWI system): 18.7 to 96.20
6. DMC – индекс DMC (from the FWI system): 1.1 to 291.3
7. DC – индекс DC (from the FWI system): 7.9 to 860.6
8. ISI – индекс ISI (from the FWI system): 0.0 to 56.10
9. temp – температура в градусах Цельсия: 2.2 to 33.30
10. RH – относительная влажность в %: 15.0 to 100
11. wind – скорость ветра в км/ч: 0.40 to 9.40
12. rain – осадки в мм/м2 : 0.0 to 6.4
13. area – выжженная площадь леса (в га) 0.00 to 1090.84

Первая рекомендация размещена прямо на странице датасета, а именно *this output variable is very skewed towards 0.0, thus it may make sense to model with the logarithm transform*.

Forest Fires Data Set

Download: Data Folder, Data Set Description

Abstract: This is a difficult regression task, where the aim is to predict the burned area of forest fires, in the northeast region of Portugal, by using meteorological and other data (see details at: [Web Link]).

Data Set Characteristics:	Multivariate	**Number of Instances:**	517	**Area:**	Physical
Attribute Characteristics:	Real	**Number of Attributes:**	13	**Date Donated**	2008-02-29
Associated Tasks:	Regression	**Missing Values?**	N/A	**Number of Web Hits:**	550330

Source:

Paulo Cortez, pcortez '@' dsi.uminho.pt, Department of Information Systems, University of Minho, Portugal.
Aníbal Morais, araimorais '@' gmail.com, Department of Information Systems, University of Minho, Portugal.

Data Set Information:

In [Cortez and Morais, 2007], the output 'area' was first transformed with a ln(x+1) function. Then, several Data Mining methods were applied. After fitting the models, the outputs were post-processed with the inverse of the ln(x+1) transform. Four different input setups were used. The experiments were conducted using a 10-fold (cross-validation) x 30 runs. Two regression metrics were measured: MAD and RMSE. A Gaussian support vector machine (SVM) fed with only 4 direct weather conditions (temp, RH, wind and rain) obtained the best MAD value: 12.71 +- 0.01 (mean and confidence interval within 95% using a t-student distribution). The best RMSE was attained by the naive mean predictor. An analysis to the regression error curve (REC) shows that the SVM model predicts more examples within a lower admitted error. In effect, the SVM model predicts better small fires, which are the majority.

Таким образом, первое, что можно попробовать, это **взять логарифм от выходной переменной**. И это сразу же улучшает поведение обучающего алгоритма.

Другие полезные рекомендации можно узнать, если обратиться к ссылке на статью авторов датасета: `http://www3.dsi.uminho.pt/pcortez/fires.pdf`. В ней авторы, например, используют ограниченный набор признаков для построения модели (так, в частности, используются только признаки 9-12).

Для оценки качества модели авторами рекомендуется использовать показатели MAD и RMSE:

$$MAD = \frac{1}{N}\sum_{i=1}^{N}|y_i - \tilde{y}_i|$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}$$

Лучший показатель MAD, полученный авторами составил **12.86**. Такой результат получен применением метода SVM.

Отбор признаков (Feature Selection)

Отбор признаков является еще одним важным этапом после предобработки исходных данных, который способен предопределять последующий процесс обучения модели.
К методам отбора признаков (feature selection) относятся:

- Удаление признаков с малой дисперсией
- Проверка статистических гипотез (χ^2)
- Рекурсивное удаление признаков
- Выбор признаков, основывающийся на использовании определенной модели (к этому типу относится использование L1-нормы для удаления неинформативных признаков)

Все эти методы реализованы в scikit-learn.

Примеры: удаление признаков с малой дисперсией

```
from sklearn.feature_selection import VarianceThreshold

X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1,
    1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
print sel.fit_transform(X)
```

```
[[0 1]
 [1 0]
 [0 0]
 [1 1]
 [1 0]
 [1 1]]
```

L1-норма

```
from sklearn.svm import LinearSVC
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel

iris = load_iris()
X, y = iris.data, iris.target

lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X)
```

Небольшие изменения в коде для реализации сети Кохонена (двумерная решетка)

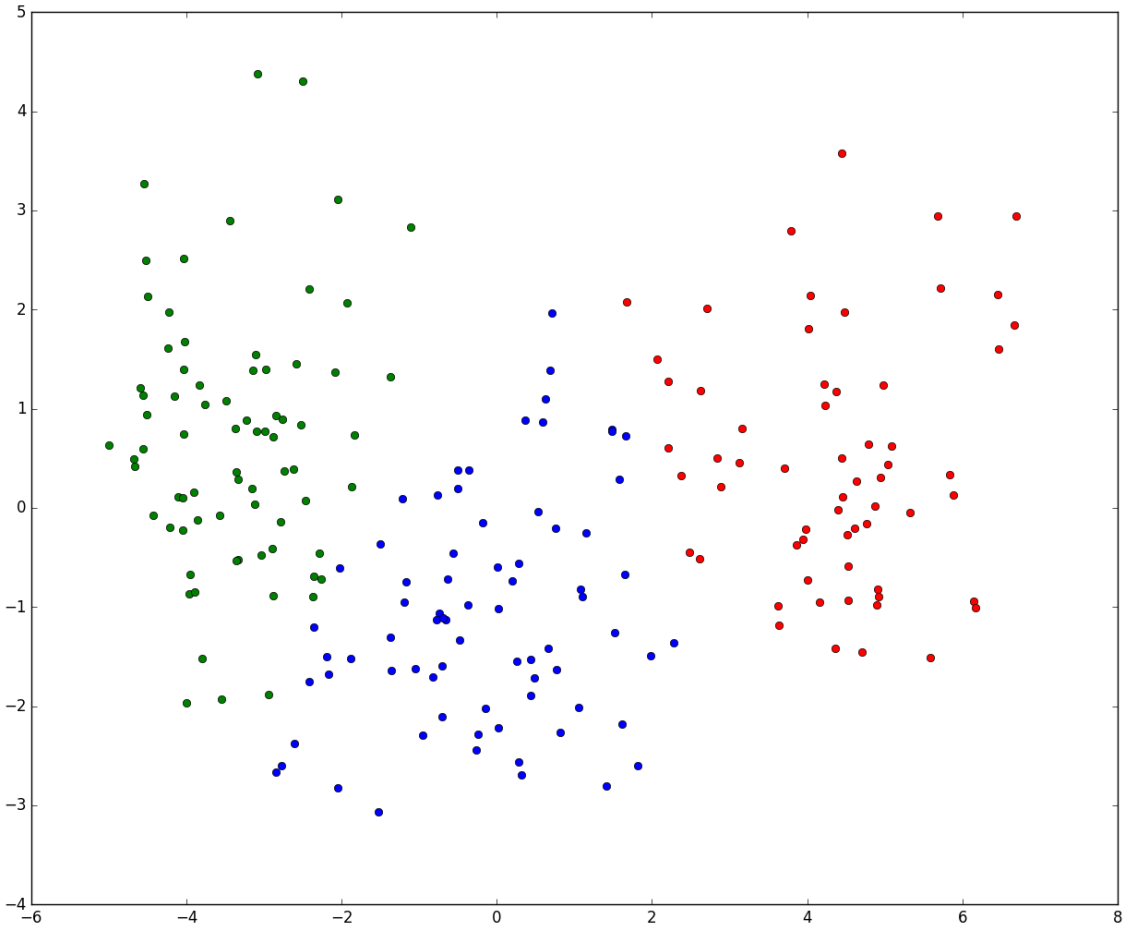
```
class KohonenMap:

  def __init__(self, shape, dimension, rate0, sigma0, tau2):
    self.weights = np.random.random((shape[0],shape[1], dimension))
    self.tau2 = tau2
    self.rate0 = rate0
    self.rate = rate0
    self.sigma0 = sigma0
    self.sigma = sigma0
    self.shape = shape
    self.neurons_count = shape[0] * shape[1]
```

```
def _define_win_neuron(self, sample):
  dist = float('Inf')
  row = col = -1
  for i in range(0, self.shape[0]):
    for j in range(0, self.shape[1]):
      if np.linalg.norm(sample-self.weights[i,j]) < dist:
        dist = np.linalg.norm(sample-self.weights[i, j])
        row = i
        col = j
  return [row, col]

def _top_loc(self, index):
  distance = np.zeros((self.shape[0], self.shape[1]))
  for i in range(0, self.shape[0]):
    for j in range(0, self.shape[1]):
      distance[i, j] = np.linalg.norm(index-np.array([i, j]))**2
  return np.exp(-distance/(2*self.sigma**2))

def _change_sigma(self, n):
  tau1 = 1000.0 / math.log(self.sigma0)
  self.sigma = self.sigma0 * math.exp(-n/tau1)

def _change_rate(self, n):
  self.rate = self.rate0 * math.exp(-n/self.tau2)
```

```
def train(self, data):
  self.core(data, 1000, True)
  self.rate = 0.01
  self.core(data, 25000, False)

def core(self, data, iterationsLimit, changeRate):
  samples_count = len(data)
  iterations = 0
  while iterations < iterationsLimit:
    index = rnd.randint(0, samples_count - 1)
    sample = data[index]
    index = self._define_win_neuron(sample)
    top_loc = self._top_loc(index)
    for i in range(0, self.shape[0]):
      for j in range(0, self.shape[1]):
        self.weights[i, j] += self.rate * top_loc[i, j] * (sample -
            self.weights[i, j])
    iterations += 1
    if changeRate:
      self._change_rate(iterations)
    self._change_sigma(iterations)
```

```
def get_clusters(self, data):
    clustering = []
    for sample in data:
        index = self._define_win_neuron(sample)
        clustering.append(index[0]*self.shape[1] + index[1])
    return clustering
```

Результаты



Сравнение с методом k-means

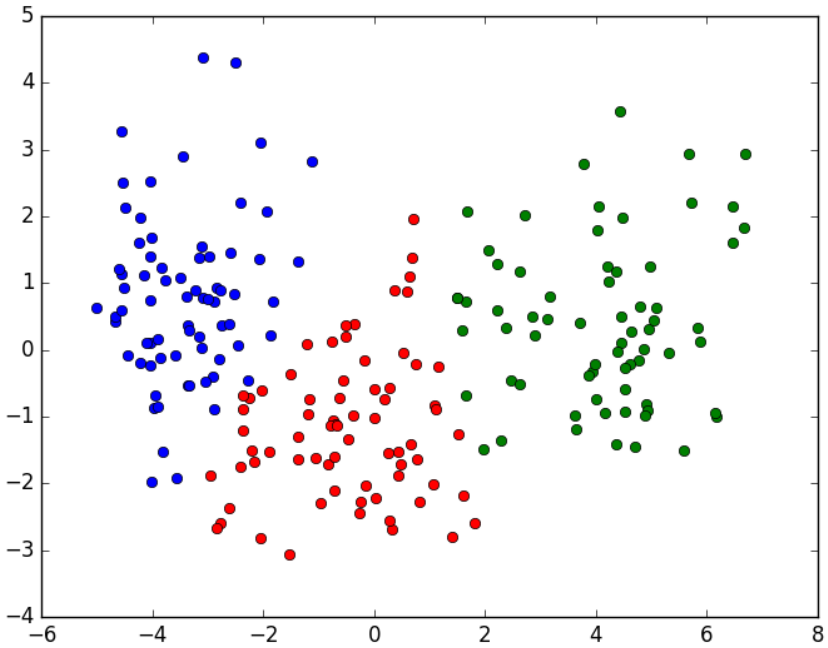


Figure 1: Результат кластеризации (SOM)

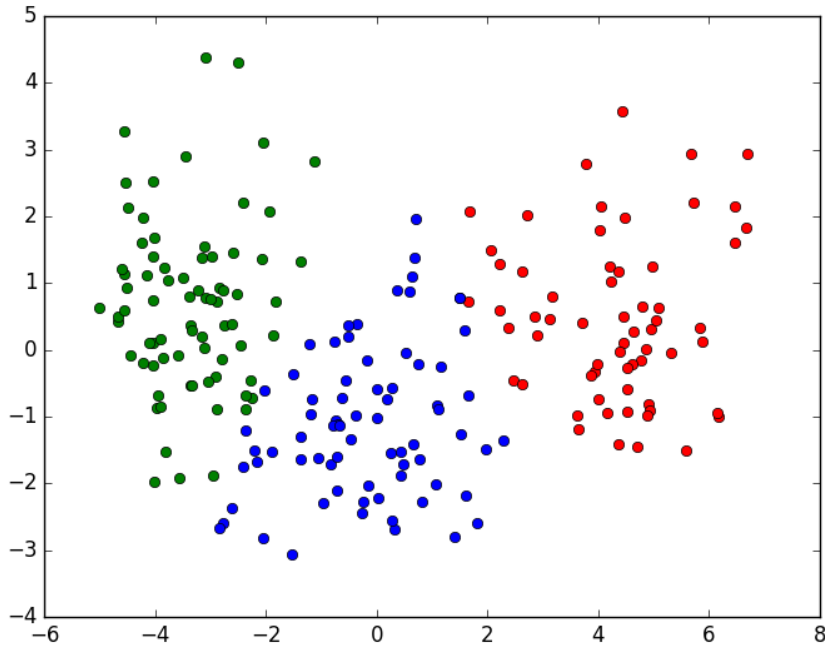
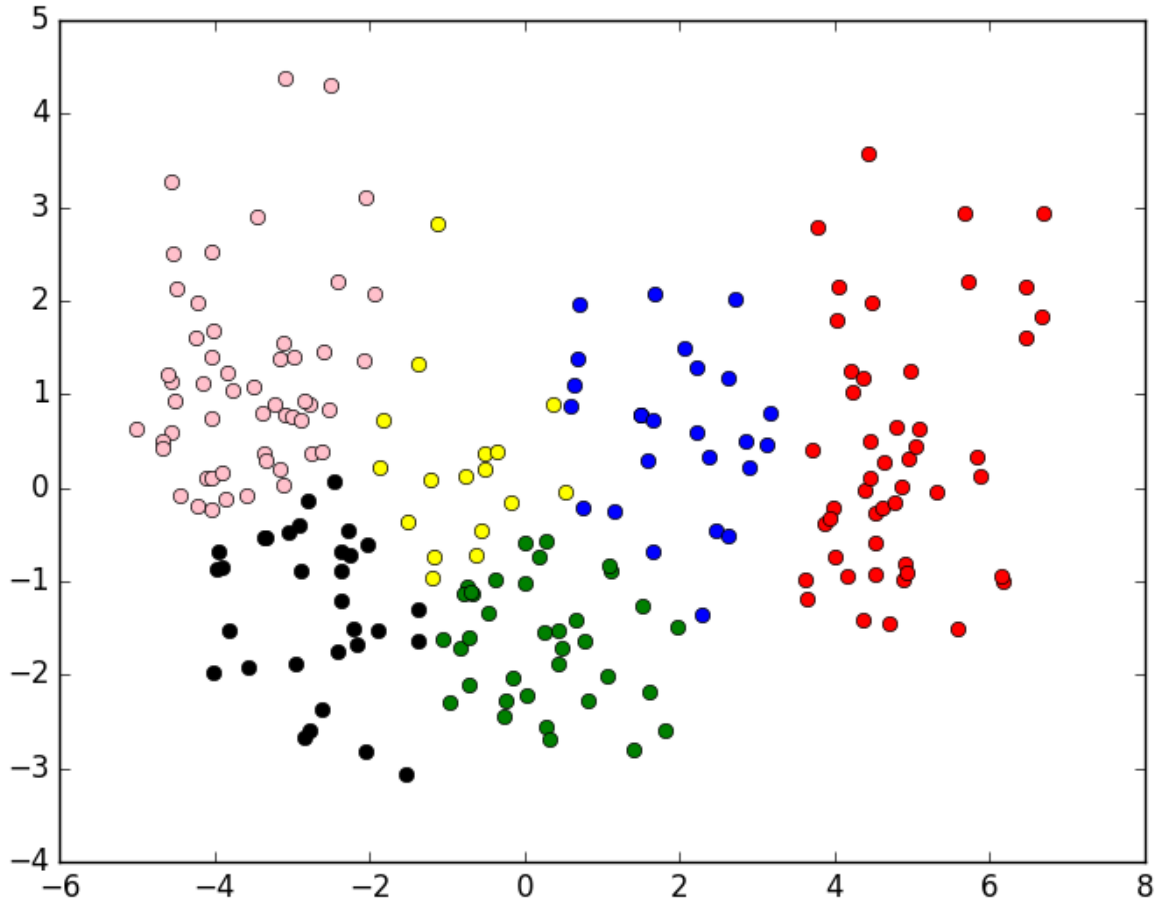
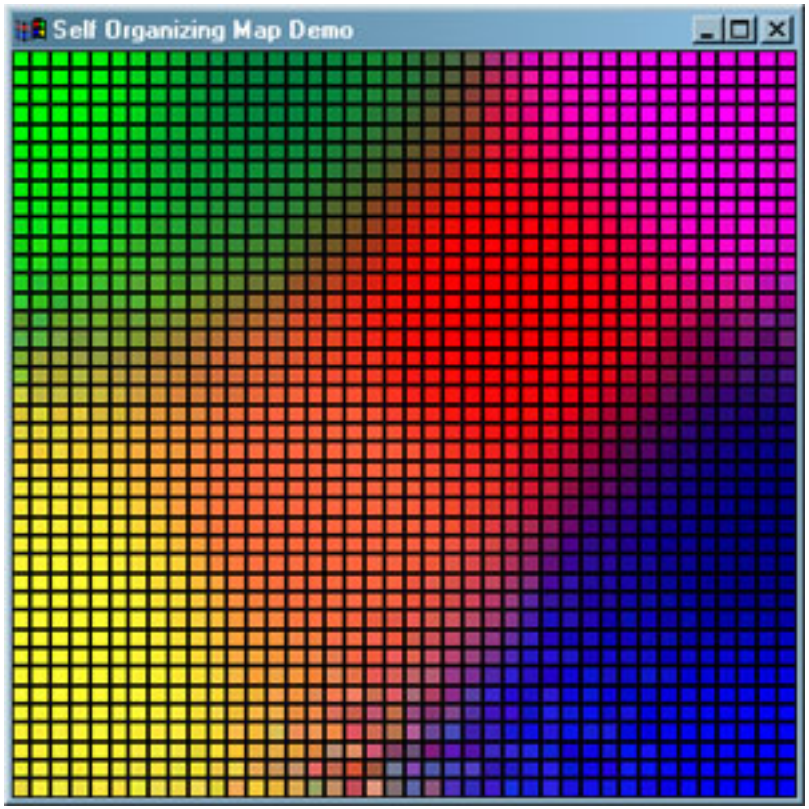


Figure 2: Результат кластеризации (k-means)

Случай k=6



Пример использования сети Кохонена для задачи кластеризации цветов



http://www.ai-junkie.com/ann/som/som1.html

Основные выводы

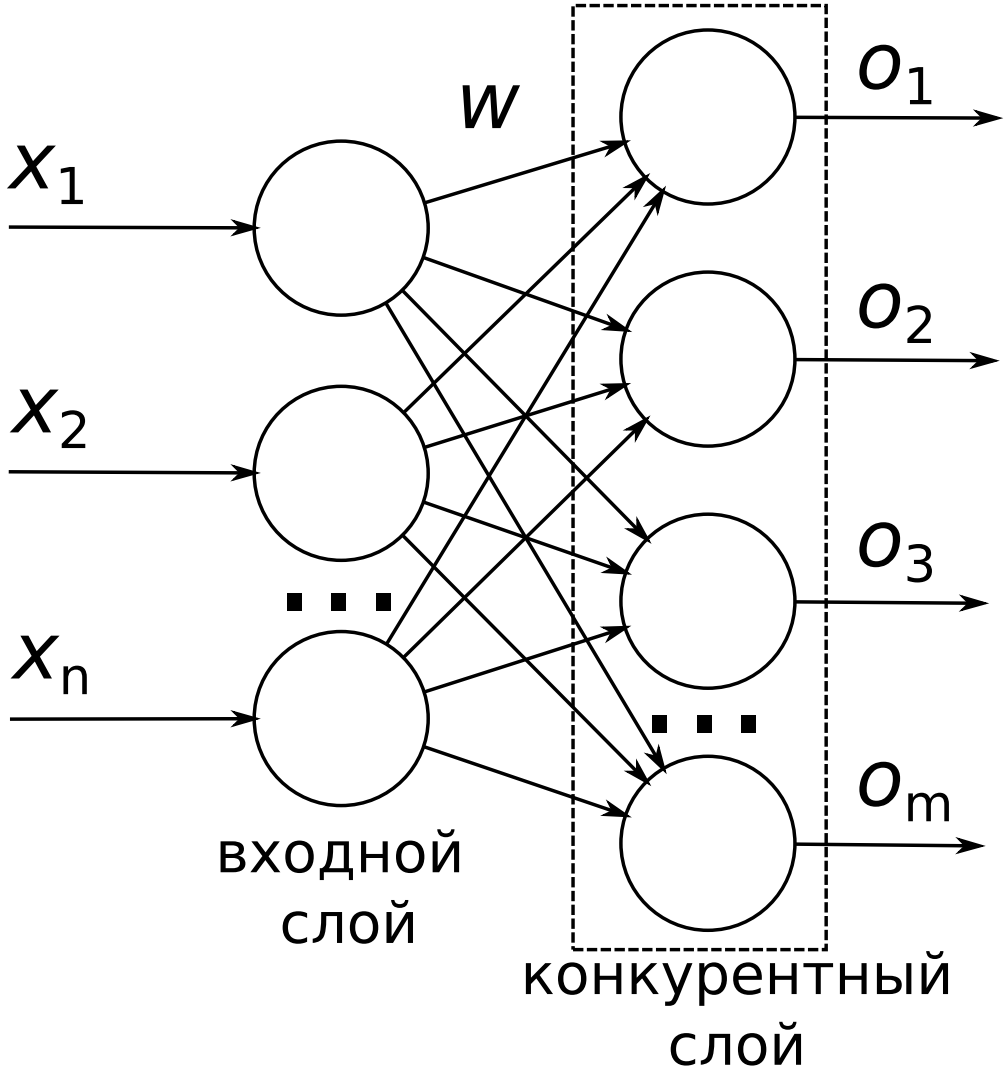
- В сущности методы кластеризации с помощью НС Кохонена и k-средних практически не отличаются по получаемым результатам
- Недостатком метода k-средних является случайный характер определения размещения начальных центроидов. Это влияет на стабильность работы метода
- Нужно понимать, что оба метода используются только для выделения подмножеств данных, близких по какой-либо метрике, но не могут использоваться для решения задач классификации
- Выбор количества кластеров тоже существенно влияет на получаемый результат. К сожалению, оба рассматриваемых подхода не способны автоматически определять это значение

LVQ

Термином **обучающееся векторное квантование (learning vector quantization)** обозначается целый класс алгоритмов (LVQ1, LVQ2, LVQ3 и OLVQ1). В то время, как базовый вариант SOM является обучением без учителя, LVQ описывает **обучение с учителем**. В то же время, в отличие от SOM, в LVQ не задаются окрестности нейрона-«победителя», тем самым нет оснований ожидать **сохранения пространственной упорядоченности кодирующих векторов**.
LVQ ориентирован на решение задачи **статистической классификации или распознавания**, т.е. основное его назначение – **разделить пространство входных данных на классы**.
С этой целью в каждую из областей, соответствующую определенному классу, помещается **подмножество сходным образом помеченных кодирующих векторов**.

Т. Кохонен – Самоорганизующиеся карты, 3-е издание

Общий вид конкурирующей сети (Competitive Network)



Алгоритм обучения LVQ

Вход: X – обучающая выборка,
Результат: Обученный классификатор LVQ
– Определить количество кластеров M
– Задать M центроид $w_c(0), c = 1, ..., M$, $M >= C$, где C – число классов
– Задать скорость обучения α и максимальное число эпох обучения
while *не выполнится условие останова* **do**
  foreach $x_i \in X$ **do**
    Подать x_i на вход нейронной сети
    Выбрать победивший нейрон m
    Обновить вектор весов для нейрона-победителя по формуле:

$$w_m(t+1) = \begin{cases} w_m(t) + \alpha(t)[x_i - w_m(t)], x_i, w_m \in C_n \\ w_m(t) - \alpha(t)[x_i - w_m(t)], x_i \in C_n, w_m \in C_k, C_n \neq C_k, n \neq k \end{cases}$$

  end
  Уменьшить α
end

α можно изменять так: $\alpha = \alpha * k$, где k – некоторый коэффициент

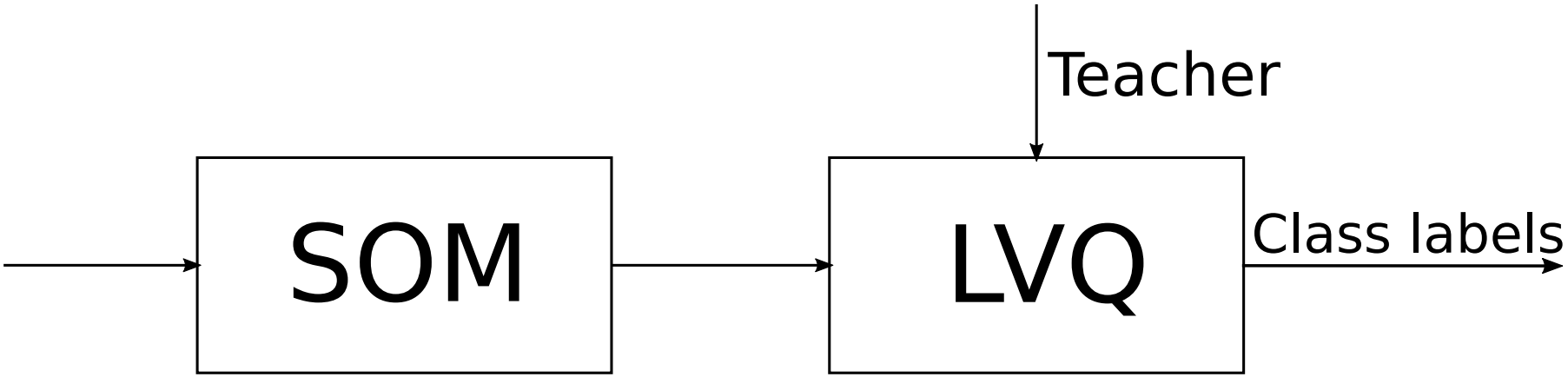
Реализация

```
import numpy as np
import itertools as it

class LVQ:
  def __init__(self, centroids, targets, max_epochs_count, rate):
    self.centroids = centroids
    self.targets = targets
    self.max_epochs_count = max_epochs_count
    self.rate = rate

  def train(self, data, targets):
    epoch = 0
    while epoch < self.max_epochs_count:
      for sample, target in it.izip(data, targets):
        index = self.define_win_neuron(sample)
        self.change_neuron_weights(index, sample, target)
        self.change_rate()
      epoch += 1
```

Реализация

```
def define_win_neuron(self, sample):
  tmp = np.linalg.norm(sample - self.centroids, axis=1)
  return tmp.argmin()

def change_neuron_weights(self, index, sample, target):
  if self.targets[index] == target:
    self.centroids[index]+=self.rate*(sample-self.centroids[index])
  else:
    self.centroids[index]-=self.rate*(sample-self.centroids[index])

def change_rate(self):
  self.rate *= 0.9

def test(self, data):
  output = []
  for sample in data:
    output.append(self.targets[self.define_win_neuron(sample)])
  return output
```

Использование сети Кохонена в составе классифицирующей системы



Пример: выборка Seeds Data Set

Продемонстрируем построение классифицирующей системы на базе SOM и LVQ на примере все той же выборки Seeds Data Set.

```
#clusterization part
map = kohonenmap.KohonenMap((4, 4), 7, 0.1, 2., 1000.)
map.train(data)
centroids = map.get_plain_presentation_of_weights('
    kohonen_map_weights.txt', True)

#save and analysis of weights
#...

#classification part
lvq_net = lvq.LVQ(centroids, [2,2,2,0,2,2,0,1,0,0,1,1,0,0,1,1],
    1000, 0.1)
lvq_net.train(data, labels)

lvq_otput = lvq_net.test(data)
print testing(lvq_otput, labels)

data = PCA.pca_method_sklearn(data)
PCA.visualise_data(data, lvq_otput)
```

Результат

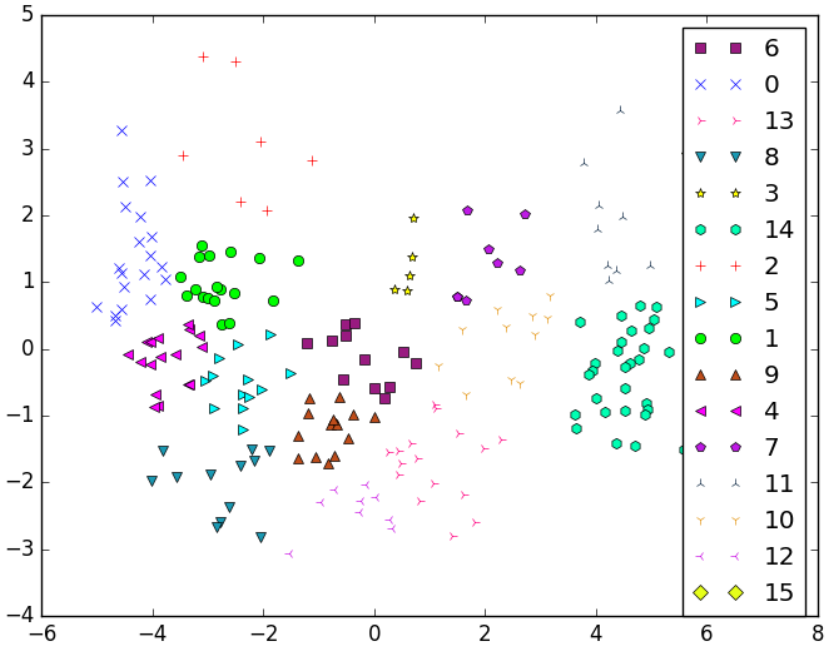


Figure 3: Результат после кластеризации

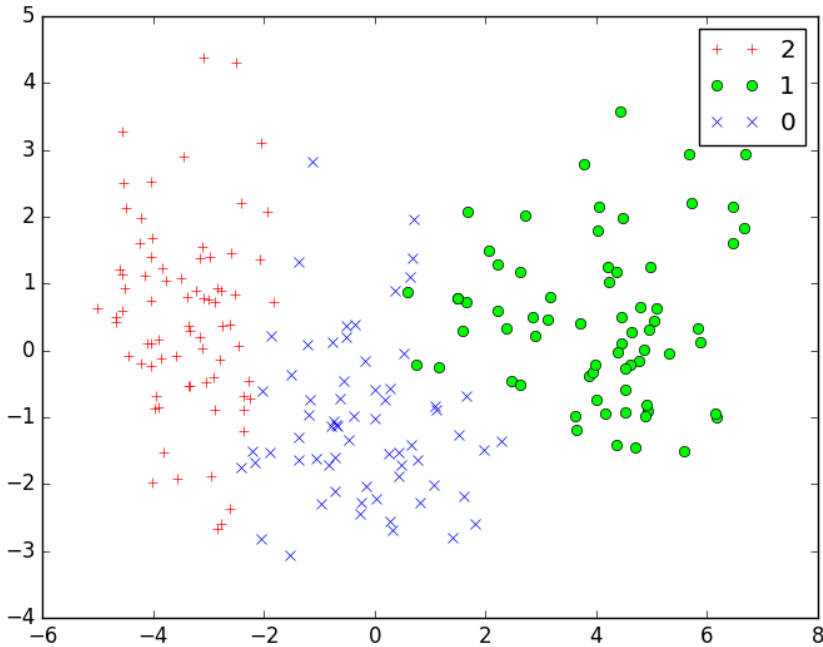
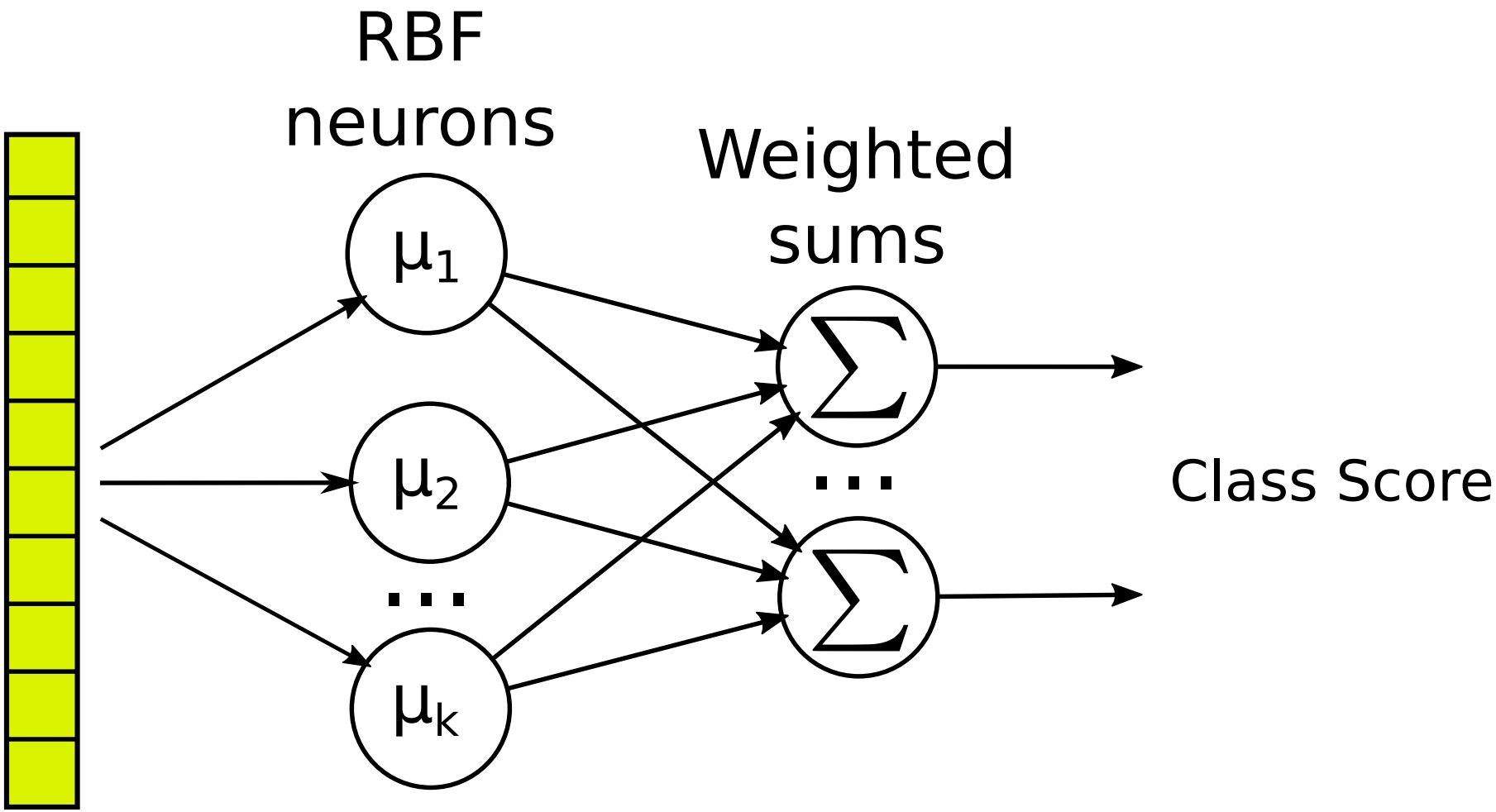


Figure 4: Результат после применения LVQ, точность = 93.3%

Радиально-базисная нейронная сеть (Radial Basis Function Network)

- Представляет собой сеть, выполняющую классификацию входных образов, сравнивая их с хранящимися «прототипами»
- Прототип являются просто одним из образов из тренировочного множества
- Когда необходимо классифицировать новый образ, каждый нейрон вычисляет евклидово расстояние между входом и своим прототипом
- RBF-сеть имеет один дополнительный полносвязный слой, обучаемый методом градиентного спуска

Архитектура РБНС



Обучение радиально-базисной НС

РБНС-сеть может обучаться как **непосредственно** методом обратного распространения ошибки (прототипы конфигурируются в процессе обучения), так и более простым методом, включающим использование **заранее определенных прототипов**. В качестве прототипов может использоваться результат работы алгоритма **k-средних (центроиды)**, которые и зададут значения для прототипов RBF-нейронов.
Для RBF-нейронов выполняется вычисление меры близости по формуле:

$$\phi(x) = e^{-\beta||x-\mu||^2}$$

где μ определяет прототип для заданного нейрона
Параметры β могут быть вычислены по формуле:

$$\beta = \frac{1}{2\sigma^2}$$

$$\sigma = \frac{1}{m}\sum_{i=1}^{m}||x_i - \mu||$$

где m – количество образов, относящихся к кластеру

Алгоритм обучения РБНС

Вход: X – обучающая выборка, G – желаемые отклики
Результат: Обученный радиально-базисный классификатор
– Получить k прототипов каким-либо алгоритмом
– Вычислить параметры σ_k для каждого RBF-нейрона по формуле $\sigma_k = \frac{1}{m}\sum_{i=1}^{m}||x_i - \mu_k||$
– Вычислить параметры β_k для каждого RBF-нейрона по формуле $\beta_k = \frac{1}{2\sigma_k^2}$
– Задать скорость обучения α и максимальное число эпох обучения
– Вычислить выходную активность RBF-слоя
while *не выполнится условие останова* **do**
  foreach $x_i \in X$ **and** $g_j \in G$ **do**
    Подать x_i на вход полносвязного слоя нейронной сети, получить y_j
    Обновить веса и пороги полносвязного слоя:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha(y_j - g_j)x_i$$

$$T_j(t+1) = T_j(t) - \alpha(y_j - g_j)$$

  end
end

Задача прогнозирования: постановка

Задача прогнозирования формулируется следующим образом: Необходимо, зная значения некоторой (вообще говоря, неизвестной) функции $f(t)$ в предшествующие моменты времени $t_{i-n}, t_{i-n+1}, ...t_{i-1}$, определить ее значение в момент времени t_i.

Решение подобных задач с помощью нейронных сетей сводится к обучению модели на сериях значений функции в предшествующие моменты времени. При этом целевым значением выступает значение функции в последующий момент.