机器学习入门之深度学习基础阶段

2018-10-18 来自

摘要：本文主要向大家介绍了机器学习入门之深度学习基础阶段，通过具体的内容向大家展现，希望对大家学习机器学习入门有所帮助。

本文主要向大家介绍了机器学习入门之深度学习基础阶段，通过具体的内容向大家展现，希望对大家学习机器学习入门有所帮助。

1.1 课程介绍 & 机器学习介绍
1. 课程介绍

2. 机器学习 （Machine Learning, ML）

2.1 概念：多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。

2.2 学科定位：人工智能(Artificial Intelligence, AI）的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。

2.3 定义：探究和开发一系列算法来如何使计算机不需要通过外部明显的指示，而可以自己通过数据来学习，建模，并且利用建好的模型和新的输入来进行预测的学科。

Arthur Samuel (1959): 一门不需要通过外部程序指示而让计算机有能力自我学习的学科

Langley（1996）：“机器学习是一门人工智能的科学，该领域的主要研究对象是人工智能，特别是如何在经验学习中改善具体算法的性能”

Tom Michell (1997): “机器学习是对能通过经验自动改进的计算机算法的研究”

2.4： 学习：针对经验E (experience) 和一系列的任务 T (tasks) 和一定表现的衡量 P，如果随之经验E的积累，针对定义好的任务T可以提[illegible]

语音识别
自动驾驶
语言翻译
计算机视觉
推荐系统
无人机
识别垃圾邮件

4. Demo：
人脸识别
无人驾驶汽车
电商推荐系统

5. 置业市场需求：Linkedln所有职业技能需求量第一：机器学习，数据挖掘和统计分析人才

1.2 深度学习(Deep Learning)介绍

1. 机器学习更多应用举例： 人脸识别

2. 机器学习就业需求：
Linkedln所有职业技能需求量第一：机器学习，数据挖掘和统计分析人才
http://blog.linkedin.com/2014/12/17/the-25-hottest-skills-that-got-people-hired-in-2014/

3. 深度学习(Deep Learning)

3.1 什么是深度学习?

3.2 深度学习什么时间段发展起来的?

其概念由著名科学家Geoffrey Hinton等人在2006年和2007年在《Sciences》等上发表的文章被提出和兴起。

3.3 学习能用来干什么？为什么近年来引起如此广泛的关注?

深度学习，作为机器学习中延伸出来的一个领域，被应用在图像处理与计算机视觉，自然语言处理以及语音识别等领域。自2006年至今，学术界和工业界合作在深度学习方面的研究与应用在以上领域取得了突破性的进展。以ImageNet为数据库的经典图像中的物体识别竞赛为例，击败了所有传统算法，取得了前所未有的精确度。

3.4 深度学习目前有哪些代表性的学术机构和公司走在前沿？人才需要如何?

学校以多伦多大学，纽约大学，斯坦福大学为代表，工业界以Google, Facebook, 和百度为代表走在深度学习研究与应用的前沿。Google挖走了Hinton，Facebook挖走了LeCun，百度硅谷的实验室挖走了Andrew Ng，Google去年4月份以超过5亿美金收购了专门研究深度学习的初创公司DeepMind, 深度学习方因技术的发展与人才的稀有造成的人才抢夺战达到了前所未有激烈的程度。诸多的大大小小(如阿里巴巴，雅虎）等公司也都在跟进，开始涉足深度学习领域，深度学习人才需求量会持续快速增长。

3.5深度学习如今和未来将对我们生活造成怎样的影响?

4. 深度学习的应用展示：

4.1 无人驾驶汽车中的路标识别
4.2 Google Now中的语音识别
4.3 百度识图
4.4 针对图片，自动生成文字的描述：

“A person riding a motorcycle on a dirt road,”

“A group of young people playing Frisbee,”

“A herd of elephants walking across a dry grass field.”

2 基本概念 (Basic Concepts)

1. 基本概念：训练集，测试集，特征值，监督学习，非监督学习，半监督学习，分类，回归

2. 概念学习：人类学习概念：鸟，车，计算机

定义：概念学习是指从有关某个布尔函数的输入输出训练样例中推断出该布尔函数

3. 例子：学习“享受运动" 这一概念：

预报：一样，变化

享受运动：是，否

概念定义在实例(instance)集合之上，这个集合表示为X。（X：所有可能的日子，每个日子的值由 天气，温度，湿度，风力，水温，预报 6个属性表示。

待学习的概念或目标函数成为目标概念（target concept), 记做c。

c(x) = 1, 当享受运动时， c(x) = 0 当不享受运动时，c(x)也可叫做y

x: 每一个实例

X: 样例, 所有实例的集合

学习目标：f: X –> Y

4. 训练集(training set/data)/训练样例（training examples): 用来进行训练，也就是产生模型或者算法的数据集

测试集(testing set/data)/测试样例 (testing examples)：用来专门进行测试已经学习好的模型或者算法的数据集

特征向量(features/feature vector)：属性的集合，通常用一个向量来表示，附属于一个实例

标记(label): c(x), 实例类别的标记

正例(positive example)

反例(negative example)

5. 例子：研究美国硅谷房价

6. 分类 (classification): 目标标记为类别型数据(category)

回归(regression): 目标标记为连续性数值 (continuous numeric value)

7. 例子：研究肿瘤良性，恶性于尺寸，颜色的关系

特征值：肿瘤尺寸，颜色

标记：良性/恶性

8. 机器学习步骤框架

8.1 把数据拆分为训练集和测试集
8.2 用训练集和训练集的特征向量来训练算法
8.2 用学习来的算法运用在测试集上来评估算法 （可能要设计到调整参数（parameter tuning), 用验证集（validation set）

100 天： 训练集
10天：测试集 （不知道是否 ”享受运动“， 知道6个属性，来预测每一天是否享受运动）

3.1 决策树(decision tree)算法

0. 机器学习中分类和预测算法的评估：

准确率
速度
强壮行
可规模性
可解释性

1. 什么是决策树/判定树（decision tree)?

2. 机器学习中分类方法中的一个重要算法

3. 构造决策树的基本算法 分支 根结点
结点
树叶

3.1 熵（entropy）概念：

信息和抽象，如何度量?
1948年，香农提出了 ”信息熵(entropy)“的概念
一条信息的信息量大小和它的不确定性有直接的关系，要搞清楚一件非常非常不确定的事情，或者是我们一无所知的事情，需要了解大量信息==>信息量的度量就等于不确定性的多少

例子：猜世界杯冠军，假如一无所知，猜多少次?
每个队夺冠的几率不是相等的

比特(bit)来衡量信息的多少

变量的不确定性越大，熵也就越大

选择属性判断结点

信息获取量(Information Gain)：Gain(A) = Info(D) – Infor_A(D)
通过A来作为节点分类获取了多少信息

类似，Gain(income) = 0.029, Gain(student) = 0.151, Gain(credit_rating)=0.048

所以，选择age作为第一个根节点

重复。。。

算法：

树以代表训练样本的单个结点开始（步骤1）。
如果样本都在同一个类，则该结点成为树叶，并用该类标号（步骤2 和3）。
否则，算法使用称为信息增益的基于熵的度量作为启发信息，选择能够最好地将样本分类的属性（步骤6）。该属性成为该结点的“测试”或“判定”

递归划分步骤仅当下列条件之一成立停止：
(a) 给定结点的所有样本属于同一类（步骤2 和3）。
(b) 没有剩余属性可以用来进一步划分样本（步骤4）。在此情况下，使用多数表决（步骤5）。
这涉及将给定的结点转换成树叶，并用样本中的多数所在的类标记它。替换地，可以存放结
点样本的类分布。
(c) 分枝
test_attribute = a i 没有样本（步骤11）。在这种情况下，以 samples 中的多数类
创建一个树叶（步骤12）

3.1 其他算法：
C4.5：Quinlan
Classification and Regression Trees (CART): (L. Breiman, J. Friedman, R. Olshen, C. Stone)
共同点：都是贪心算法，自上而下(Top-down approach)
区别：属性选择度量方法不同： C4.5 （gain ratio), CART(gini index), ID3 (Information Gain)

3.2 如何处理连续性变量的属性?

4. 树剪枝叶 （避免overfitting）
4.1 先剪枝
4.2 后剪枝

5. 决策树的优点：
直观，便于理解，小规模数据集有效

3.2 决策树(decision tree)应用
1. Python

2. Python机器学习的库：scikit-learn

2.1： 特性：

简单高效的数据挖掘和机器学习分析
对所有用户开放，根据不同需求高度可重用性
基于Numpy, SciPy和matplotlib
开源，商用级别：获得 BSD许可

2.2 覆盖问题领域：
分类（classification), 回归（regression), 聚类（clustering), 降维(dimensionality reduction)
模型选择(model selection), 预处理(preprocessing)

3. 使用用scikit-learn
安装scikit-learn: pip, easy_install, windows installer
安装必要package：numpy， SciPy和matplotlib， 可使用Anaconda (包含numpy, scipy等科学计算常用package)
安装注意问题：Python解释器版本（2.7 or 3.4？）, 32-bit or 64-bit系统

4. 例子：

安装 Graphviz： http://www.graphviz.org/
配置环境变量

转化dot文件至pdf可视化决策树：dot -Tpdf iris.dot -o outpu.pdf

1. Python

2. Python机器学习的库：scikit-learn

2.1： 特性：

简单高效的数据挖掘和机器学习分析
对所有用户开放，根据不同需求高度可重用性
基于Numpy, SciPy和matplotlib
开源，商用级别：获得 BSD许可

2.2 覆盖问题领域：
分类（classification), 回归（regression), 聚类（clustering), 降维(dimensionality reduction)
模型选择(model selection), 预处理(preprocessing)

3. 使用用scikit-learn

4. 例子：

文档： http://scikit-learn.org/stable/modules/tree.html

解释Python代码

安装 Graphviz： http://www.graphviz.org/
配置环境变量

转化dot文件至pdf可视化决策树：dot -Tpdf iris.dot -o outpu.pdf

4.1 最邻近规则分类(K-Nearest Neighbor)KNN算法

1. 综述
   1.1 Cover和Hart在1968年提出了最初的邻近算法
   1.2 分类(classification)算法
   1.3 输入基于实例的学习(instance-based learning), 懒惰学习(lazy learning)

2. 例子：

3. 算法详述

3.1 步骤：
为了判断未知实例的类别，以所有已知类别的实例作为参照
选择参数K
计算未知实例与所有已知实例的距离
选择最近K个已知实例
根据少数服从多数的投票法则(majority-voting)，让未知实例归类为K个最邻近样本中最多数的类别

3.2 细节:
关于K
关于距离的衡量方法:
3.2.1 Euclidean Distance 定义

其他距离衡量：余弦值（cos），相关度 （correlation），曼哈顿距离 （Manhattan distance）

3.3 举例

4. 算法优缺点：

4.1 算法优点

简单

易于理解

容易实现

通过对K的选择可具备丢噪音数据的健壮性

4.2 算法缺点

需要大量空间储存所有已知实例

算法复杂度高（需要比较所有已知实例与要分类的实例）

当其样本分布不平衡时，比如其中一类样本过大（实例数量过多）占主导的时候，新的未知实例容易被归类为这个主导样本，因为这类样本实例的数量过大，但这个新的未知实例实际并木接近目标样本

5. 改进版本

考虑距离，根据距离加上权重

比如: 1/d (d: 距离)

4.2 最邻近规则分类(K-Nearest Neighbor)KNN算法应用

1 数据集介绍：

虹膜

150个实例

Iris setosa, Iris versicolor, Iris virginica.

2. 利用Python的机器学习库sklearn: SkLearnExample.py

```
from sklearn import neighbors
from sklearn import datasets

knn = neighbors.KNeighborsClassifier()


iris = datasets.load_iris()


print iris

knn.fit(iris.data, iris.target)

predictedLabel = knn.predict([[0.1, 0.2, 0.3, 0.4]])

print predictedLabel
```

```
# Example of kNN implemented from Scratch in Python

import csv
import random
import math
import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
```

```
        for x in range(k):
            neighbors.append(distances[x][0])
        return neighbors

    def getResponse(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
            response = neighbors[x][-1]
            if response in classVotes:
                classVotes[response] += 1
            else:
                classVotes[response] = 1
        sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]

    def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
            if testSet[x][-1] == predictions[x]:
                correct += 1
        return (correct/float(len(testSet))) * 100.0

    def main():
        # prepare data
        trainingSet=[]
        testSet=[]
        split = 0.67
        loadDataset(r'D:\MaiziEdu\DeepLearningBasics_MachineLearning\Datasets\iris.data.txt', split, trainingSet, testSet)
        print 'Train set: ' + repr(len(trainingSet))
```

```
            neighbors = getNeighbors(trainingSet, testSet[x], k)
            result = getResponse(neighbors)
            predictions.append(result)
            print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: ' + repr(accuracy) + '%')

main()
```

5.1 支持向量机(SVM)算法（上）

1. 背景：

    1.1 最早是由 Vladimir N. Vapnik 和 Alexey Ya. Chervonenkis 在1963年提出
    1.2 目前的版本(soft margin)是由Corinna Cortes 和 Vapnik在1993年提出，并在1995年发表
    1.3 深度学习（2012）出现之前，SVM被认为机器学习中近十几年来最成功，表现最好的算法

2. 机器学习的一般框架：
    训练集 => 提取特征向量 => 结合一定的算法（分类器：比如决策树，KNN）=>得到结果

3. 介绍：

    3.1 例子：

        两类？哪条线最好?

总共可以有多少个可能的超平面？无数条

如何选取使边际(margin)最大的超平面 (Max Margin Hyperplane)?

超平面到一侧最近点的距离等于到另一侧最近点的距离，两侧的两个超平面平行

3. 线性可区分(linear separable) 和 线性不可区分 (linear inseparable)

4. 定义与公式建立

超平面可以定义为：

W: weight vectot, , n 是特征值的个数
X: 训练实例
b: bias

4.1 假设2维特征向量：X = (x1, X2)
把 b 想象为额外的 wight
超平面方程变为：

所有超平面右上方的点满足：

综合以上两式，得到： (1)

所有坐落在边际的两边的的超平面上的被称作”支持向量(support vectors)"

分界的超平面和H1或H2上任意一点的距离为 (i.e.: 其中||W||是向量的范数(norm))

所以，最大边际距离为：

5. 求解

5.1 SVM如何找出最大边际的超平面呢(MMH)?

利用一些数学推倒，以上公式 (1) 可变为有限制的凸优化问题(convex quadratic optimization)
利用 Karush–Kuhn–Tucker (KKT)条件和拉格朗日公式，可以推出MMH可以被表示为以下“决定边界 (decision boundary)”

其中，

是支持向量点 (support vector)的类别标记（class label)

是要测试的实例

和 都是单一数值型参数，由以上提到的最有算法得出

6. 例子：

5.1 支持向量机(SVM)算法（上）应用

1 sklearn简单例子

```
from sklearn import svm

X = [[2, 0], [1, 1], [2,3]]
y = [0, 0, 1]
clf = svm.SVC(kernel = 'linear')
clf.fit(X, y)

print clf

# get support vectors
print clf.support_vectors_

# get indices of support vectors
print clf.support_
```

2 sklearn画出决定界限

```
print(__doc__)

import numpy as np
import pylab as pl
from sklearn import svm

# we create 40 separable points
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) – [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# fit the model
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)

# get the separating hyperplane
w = clf.coef_[0]
a = –w[0] / w[1]
xx = np.linspace(–5, 5)
yy = a * xx – (clf.intercept_[0]) / w[1]

# plot the parallels to the separating hyperplane that pass through the
# support vectors
b = clf.support_vectors_[0]
yy_down = a * xx + (b[1] – a * b[0])
b = clf.support_vectors_[–1]
```

```
# print " xx: ", xx
# print " yy: ", yy
print "support_vectors_: ", clf.support_vectors_
print "clf.coef_: ", clf.coef_

# In scikit-
learn coef_ attribute holds the vectors of the separating hyperplanes for linear models. It has shape (n_classes, n_features) if n_classes > 1 (
multi-class one-vs-all) and (1, n_features) for binary classification.
#
# In this toy binary classification example, n_features == 2, hence w = coef_[0] is the vector orthogonal to the hyperplane (the hyperplane is
fully defined by it + the intercept).
#
# To plot this hyperplane in the 2D case (any hyperplane of a 2D plane is a 1D line), we want to find a f as in y = f(x) = a.x + b. In this case a
 is the slope of the line and can be computed by a = -w[0] / w[1].




# plot the line, the points, and the nearest vectors to the plane
pl.plot(xx, yy, 'k-')
pl.plot(xx, yy_down, 'k--')
pl.plot(xx, yy_up, 'k--')

pl.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
           s=80, facecolors='none')
pl.scatter(X[:, 0], X[:, 1], c=Y, cmap=pl.cm.Paired)

pl.axis('tight')
pl.show()
```

1.1 训练好的模型的算法复杂度是由支持向量的个数决定的，而不是由数据的维度决定的。所以SVM不太容易产生overfitting

1.2 SVM训练出来的模型完全依赖于支持向量(Support Vectors), 即使训练集里面所有非支持向量的点都被去除，重复训练过程，结果仍然会得到完全一样的模型。

1.3 一个SVM如果训练得出的支持向量个数比较小，SVM训练出的模型比较容易被泛化。

2. 线性不可分的情况 （linearly inseparable case）

2.1 数据集在空间中对应的向量不可被一个超平面区分开

2.2 两个步骤来解决：

2.2.1 利用一个非线性的映射把原数据集中的向量点转化到一个更高维度的空间中

2.2.2 在这个高维度的空间中找一个线性的超平面来根据线性可分的情况处理

2.2.3 视觉化演示 https://www.youtube.com/watch?v=3liCbRZPrZA

2.3 如何利用非线性映射把原始数据转化到高维中?

2.3.1 例子：

3维输入向量：

转化到6维空间 Z 中去：

2.3.2 思考问题：

2.3.2.1: 如何选择合理的非线性转化把数据转到高纬度中?

2.3.2.2： 如何解决计算内积时算法复杂度非常高的问题?

2.3.3 使用核方法（kernel trick）

3. 核方法（kernel trick）

3.1 动机

在线性SVM中转化为最优化问题时求解的公式计算都是以内积(dot product)的形式出现的，其中 是把训练集中的向量点转化到高维的非线性映射函数，因为内积的算法复杂度非常大，所以我们利用核函数来取代计算非线性映射函数的内积

3.1 以下核函数和非线性映射函数的内积等同

3.2 常用的核函数(kernel functions)

h度多项式核函数(polynomial kernel of degree h)：

高斯径向基核函数(Gaussian radial basis function kernel):

S型核函数(Sigmoid function kernel):

如何选择使用哪个kernel?

根据先验知识，比如图像分类，通常使用RBF，文字不使用RBF

尝试不同的kernel，根据结果准确度而定

3.3 核函数举例:

假设定义两个向量： x = (x1, x2, x3); y = (y1, y2, y3)

f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9) f(y) = (16, 20, 24, 20, 25, 36, 24, 30, 36) <f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

K(x, y) = (4 + 10 + 18) ^2 = 32^2 = 1024

同样的结果，使用kernel方法计算容易很多

4. SVM扩展可解决多个类别分类问题
对于每个类，有一个当前类和其他类的二类分类器（one-vs-rest）

5.3 支持向量机(SVM)算法（下）应用
利用SVM进行人脸识别实例：

```
from __future__ import print_function

from time import time
import logging
import matplotlib.pyplot as plt

from sklearn.cross_validation import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import RandomizedPCA
from sklearn.svm import SVC
```

```
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')


###############################################################################
# Download the data, if not already on disk and load it as numpy arrays

lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape

# for machine learning we use the 2 data directly (as relative pixel
# positions info is ignored by this model)
X = lfw_people.data
n_features = X.shape[1]

# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)


###############################################################################
# Split into a training set and a test set using a stratified k fold
```

```
###############################################################################
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150

print("Extracting the top %d eigenfaces from %d faces"
      % (n_components, X_train.shape[0]))
t0 = time()
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))


###############################################################################
# Train a SVM classification model

print("Fitting the classifier to the training set")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='auto'), param_grid)
```

```
###############################################################################
# Quantitative evaluation of the model quality on the test set

print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=target_names))
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))


###############################################################################
# Qualitative evaluation of the predictions using matplotlib

def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())


# plot the result of the prediction on a portion of the test set
```

```
prediction_titles = [title(y_pred, y_test, target_names, i)
                     for i in range(y_pred.shape[0])]

plot_gallery(X_test, prediction_titles, h, w)

# plot the gallery of the most significative eigenfaces

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()
```

6.1 神经网络算法(Nerual Networks)（上）

1. 背景:
   1.1 以人脑中的神经网络为启发，历史上出现过很多不同版本
   1.2 最著名的算法是1980年的 backpropagation

2. 多层向前神经网络(Multilayer Feed-Forward Neural Network)
   2.1 Backpropagation被使用在多层向前神经网络上
   2.2 多层向前神经网络由以下部分组成:
      输入层(input layer), 隐藏层 (hidden layers), 输入层 (output layers)

   2.3 每层由单元(units)组成
   2.4 输入层(input layer)是由训练集的实例特征向量传入

2.9 作为多层向前神经网络，理论上，如果有足够多的隐藏层(hidden layers) 和足够大的训练集，可以模拟出任何方程

3. 设计神经网络结构
   3.1 使用神经网络训练数据之前，必须确定神经网络的层数，以及每层单元的个数
   3.2 特征向量在被传入输入层时通常被先标准化(normalize）到0和1之间 （为了加速学习过程）
   3.3 离散型变量可以被编码成每一个输入单元对应一个特征值可能赋的值
   比如：特征值A可能取三个值（a0, a1, a2), 可以使用3个输入单元来代表A。
   如果A=a0, 那么代表a0的单元值就取1, 其他取0；
   如果A=a1, 那么代表a1de单元值就取1，其他取0，以此类推

   3.4 神经网络即可以用来做分类(classification）问题，也可以解决回归(regression)问题
   3.4.1 对于分类问题，如果是2类，可以用一个输出单元表示（0和1分别代表2类）
   如果多余2类，每一个类别用一个输出单元表示
   所以输入层的单元数量通常等于类别的数量

   3.4.2 没有明确的规则来设计最好有多少个隐藏层
   3.4.2.1 根据实验测试和误差，以及准确度来实验并改进

4. 交叉验证方法(Cross-Validation)

   K-fold cross validation

5. Backpropagation算法
   5.1 通过迭代性的来处理训练集中的实例
   5.2 对比经过神经网络后输入层预测值(predicted value)与真实值(target value)之间
   5.3 反方向（从输出层=>隐藏层=>输入层）来以最小化误差(error)来更新每个连接的权重(weight)
   5.4 算法详细介绍

一个偏向

5.4.2 对于每一个训练实例X，执行以下步骤：

5.4.2.1： 由输入层向前传送

5.4.2.2 根据误差(error)反向传送

对于输出层：

对于隐藏层：

权重更新：

偏向更新

5.4.3 终止条件

5.4.3.1 权重的更新低于某个阈值

5.4.3.2 预测的错误率低于某个阈值

5.4.3.3 达到预设一定的循环次数

6. Backpropagation 算法举例

对于输出层：

对于隐藏层：

权重更新：

偏向更新

sigmoid函数(S 曲线)用来作为activation function:

1.1 双曲函数(tanh)

1.2 逻辑函数(logistic function)

2. 实现一个简单的神经网络算法

```
import numpy as np

def tanh(x):
    return np.tanh(x)

def tanh_deriv(x):
    return 1.0 – np.tanh(x)*np.tanh(x)

def logistic(x):
    return 1/(1 + np.exp(–x))

def logistic_derivative(x):
    return logistic(x)*(1–logistic(x))



class NeuralNetwork:
    def __init__(self, layers, activation='tanh'):
        """
```

```
        if activation == 'logistic':
            self.activation = logistic
            self.activation_deriv = logistic_derivative
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_deriv = tanh_deriv

        self.weights = []
        for i in range(1, len(layers) - 1):
            self.weights.append((2*np.random.random((layers[i - 1] + 1, layers[i] + 1))-1)*0.25)
            self.weights.append((2*np.random.random((layers[i] + 1, layers[i + 1]))-1)*0.25)


    def fit(self, X, y, learning_rate=0.2, epochs=10000):
        X = np.atleast_2d(X)
        temp = np.ones([X.shape[0], X.shape[1]+1])
        temp[:, 0:-1] = X  # adding the bias unit to the input layer
        X = temp
        y = np.array(y)

        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]

            for l in range(len(self.weights)):  #going forward network, for each layer
                a.append(self.activation(np.dot(a[l], self.weights[l])))  #Computer the node value for each layer (O_i) using activation functi
            error = y[i] - a[-1]  #Computer the error at the top layer
            deltas = [error * self.activation_deriv(a[-1])] #For output layer, Err calculation (delta is updated error)

            #
```

```
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)


    def predict(self, x):
        x = np.array(x)
        temp = np.ones(x.shape[0]+1)
        temp[0:-1] = x
        a = temp
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

6.3 神经网络算法(Nerual Networks)应用(下)

1. 简单非线性关系数据集测试(XOR):

X:	Y
0 0	0
0 1	1
1 0	1
1 1	0

Code:

```
from NeuralNetwork import NeuralNetwork
import numpy as np
```

```
for i in [[0, 0], [0, 1], [1, 0], [1,1]]:
    print(i, nn.predict(i))
```

2. 手写数字识别：

每个图片8x8
识别数字：0,1,2,3,4,5,6,7,8,9

Code:

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import LabelBinarizer
from NeuralNetwork import NeuralNetwork
from sklearn.cross_validation import train_test_split

digits = load_digits()
X = digits.data
y = digits.target
X -= X.min() # normalize the values to bring them into the range 0-1
X /= X.max()

nn = NeuralNetwork([64,100,10],'logistic')
X_train, X_test, y_train, y_test = train_test_split(X, y)
labels_train = LabelBinarizer().fit_transform(y_train)
labels_test = LabelBinarizer().fit_transform(y_test)
```

```
    predictions.append(np.argmax(o))
print confusion_matrix(y_test,predictions)
print classification_report(y_test,predictions)
```

7.1 简单线性回归 (Simple Linear Regression)上

0. 前提介绍：

为什么需要统计量?

统计量：描述数据特征

0.1 集中趋势衡量

0.1.1均值（平均数，平均值）（mean）

{6, 2, 9, 1, 2}
(6 + 2 + 9 + 1 + 2) / 5 = 20 / 5 = 4
0.1.2中位数 （median）：将数据中的各个数值按照大小顺序排列，居于中间位置的变量
本文由职坐标整理并发布，希望对同学们有所帮助。了解更多详情请关注职坐标人工智能机器学习频道!

喜欢
0

不喜欢
0

看完这篇文章有何感觉?
已经有0人表态，0%的人喜欢

大家都在看

机器学习入门书籍 | 机器学习入门 视频 | 嵌入式系统

机器学习入门之深度学习基础阶段

物联网 09/29 188

机器学习入门之深度学习基础阶段

AI技术研究 09/26 156

机器学习入门之深度学习基础阶段

物联网 09/21 1

说说你的看法....

0

AI技术研究 10/12 104

机器学习入门之深度学习基础阶段

智能家居 09/25 100

热门评论（0）

暂无评论

说说你的看法....

0