模式识别与机器学习课程作业

姓名：尹志雨　学号：**2017Z8009061116**

1. 任务介绍

1.1 模式分类

模式识别又常称作模式分类，从处理问题的性质和解决问题的方法等角度，模式识别分为有监督的分类（Supervised Classification）和无监督的分类(Unsupervised Classification)两种。二者的主要差别在于，各实验样本所属的类别是否预先已知。一般说来，有监督的分类往往需要提供大量已知类别的样本，但在实际问题中，这是存在一定困难的，因此研究无监督的分类就变得十分有必要了。

1.2 评价指标

分类器常见的评价指标主要有：Accuracy，Precision ，Recall ，F1 score ，ROC 曲线，AUC曲线，PR 曲线等等。本文主要使用Accuracy评价指标，准确率是最常用的分类性能指标。

$\boldsymbol{Accuracy} = (\boldsymbol{TP} + \boldsymbol{TN})/(\boldsymbol{TP} + \boldsymbol{FN} + \boldsymbol{FP} + \boldsymbol{TN})$，即正确预测的正反例数 /总数。精确率

（Precision）容易和准确率被混为一谈。其实，精确率只是针对预测正确的正样本而不是所有预测正确的样本。表现为预测出是正的里面有多少真正是正的。可理解为查准率。

$\boldsymbol{Precision} = \boldsymbol{TP}/(\boldsymbol{TP} + \boldsymbol{FP})$，即正确预测的正例数 /预测正例总数。

2. 数据集介绍

Iris数据集是常用的分类实验数据集，它首次出现在著名的英国统计学家和生物学家Ronald Fisher 1936年的论文《The use of multiple measurements in taxonomic problems》中，由Fisher, 1936收集整理。Iris也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含150个数据集，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。
iris以鸢尾花的特征作为数据来源，常用在分类操作中。该数据集由3种不同类型的鸢尾花的50个样本数据构成。其中的一个种类与另外两个种类是线性可分离的，后两个种类是非线性可分离的。该数据集包含了5个属性：

- Sepal.Length（花萼长度），单位是cm;
- Sepal.Width（花萼宽度），单位是cm;
- Petal.Length（花瓣长度），单位是cm;
- Petal.Width（花瓣宽度），单位是cm;
- 种类：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾），以及Iris Virginica（维吉尼亚鸢尾）。

3. 算法一——KNN

KNN即K最近邻，是最简单的机器学习算法之一。已知训练样本必须先标识，然后进行训练得到未知样本分类。对于未知样本，按照某种计算距离找出它在训练集中的k个最近邻（监督学习：k个最邻近样本的分类已知），如果k个近邻中多数样本属于哪个类别，就将它判决为那一个类别。
由于采用k投票机制，所以能够减小噪声的影响。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

3.1 算法流程：

①**准**备数据，对数据进行预处理；
②**选**用合适的数据结构存储训练数据和测试元组；
③**设**定参数，如k；
④**维**护一个大小为k的的按距离由大到小的优先级队列，用于存储最近邻训练元组。随机从训练元组中选取k个元组作为初始的最近邻元组，分别计算测试元组到这k个元组的距离，将训练元组标号和距离存入优先级队列；
⑤**遍**历训练元组集，计算当前训练元组与测试元组的距离，将所得距离L 与优先级队列中的最大距离Lmax；
⑥ 进行比较。若L>=Lmax，则舍弃该元组，遍历下一个元组。若L < Lmax，删除优先级队列中最大距离的元组，将当前训练元组存入优先级队列；
⑦**遍**历完毕，计算优先级队列中k 个元组的多数类，并将其作为测试元组的类别；
⑧**测**试元组集测试完毕后计算误差率，继续设定不同的k值重新进行训练，最后取误差率最小的k 值。

3.2 算法实现

```
#coding: UTF-8
import csv
import random
import math
import operator
from sklearn import neighbors
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[y])
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(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x],dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    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():
    trainingSet = []
    testSet = []
    split = 0.67
    loadDataset(r"Iris\iris.csv", split, trainingSet, testSet)
    print "Train set :" + repr(len(trainingSet))
    print "Test set :" + repr(len(testSet))
    predictions = []
    k = 3
    for x in range(len(testSet)):
        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) + "%"
if __name__ =="__main__":
    main()
```

3.3 分类结果

```
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
test flower = 'Iris-setosa'  groundtruth = 'Iris-setosa'
Accuracy:100.0%
```

4. 算法二——SVM

4.1 算法流程

SVM算法就是找一个超平面，对于已经被标记的训练样本，SVM训练得到一个超平面，使得两个类别训练集中距离超平面最近的样本之间的垂直距离要最大（也就是如下图所示的两个虚线距离最大）。如下图所示，就是将两种不同类别的样本特征分隔开，且分割面要在最优分隔位置。

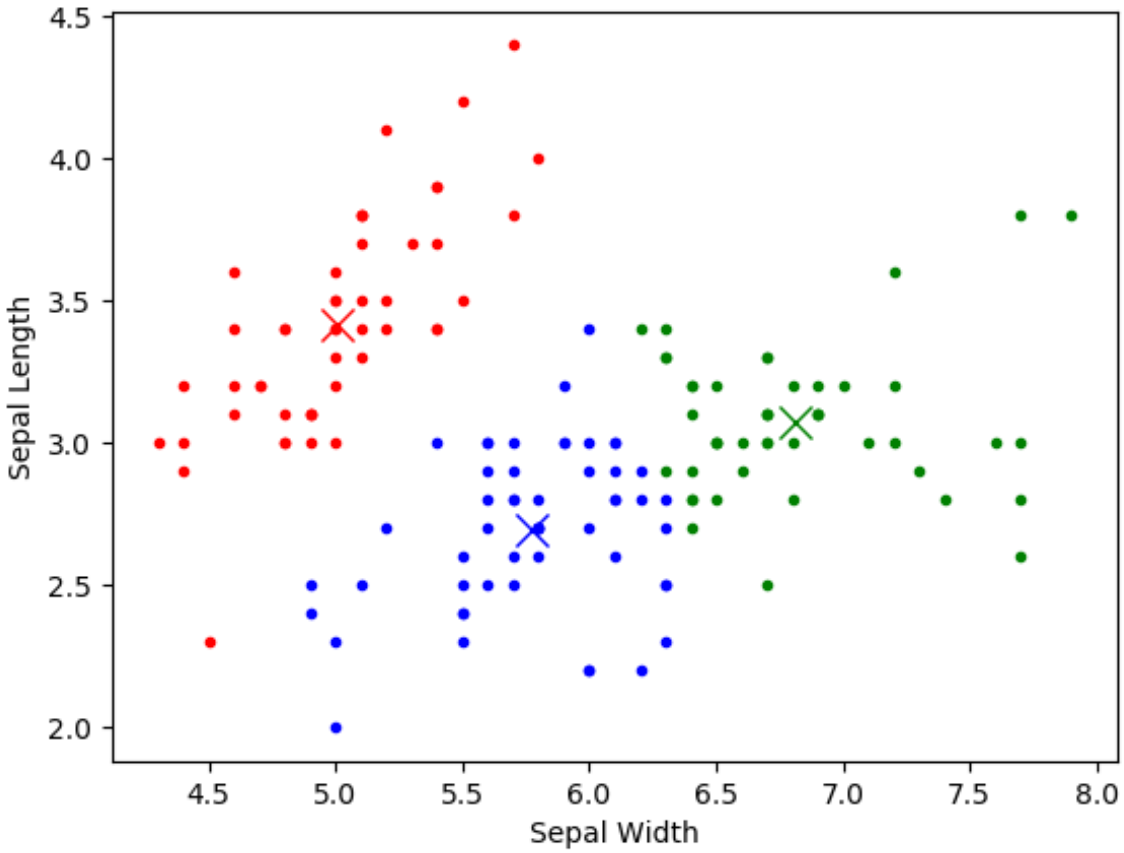
4.2 算法实现

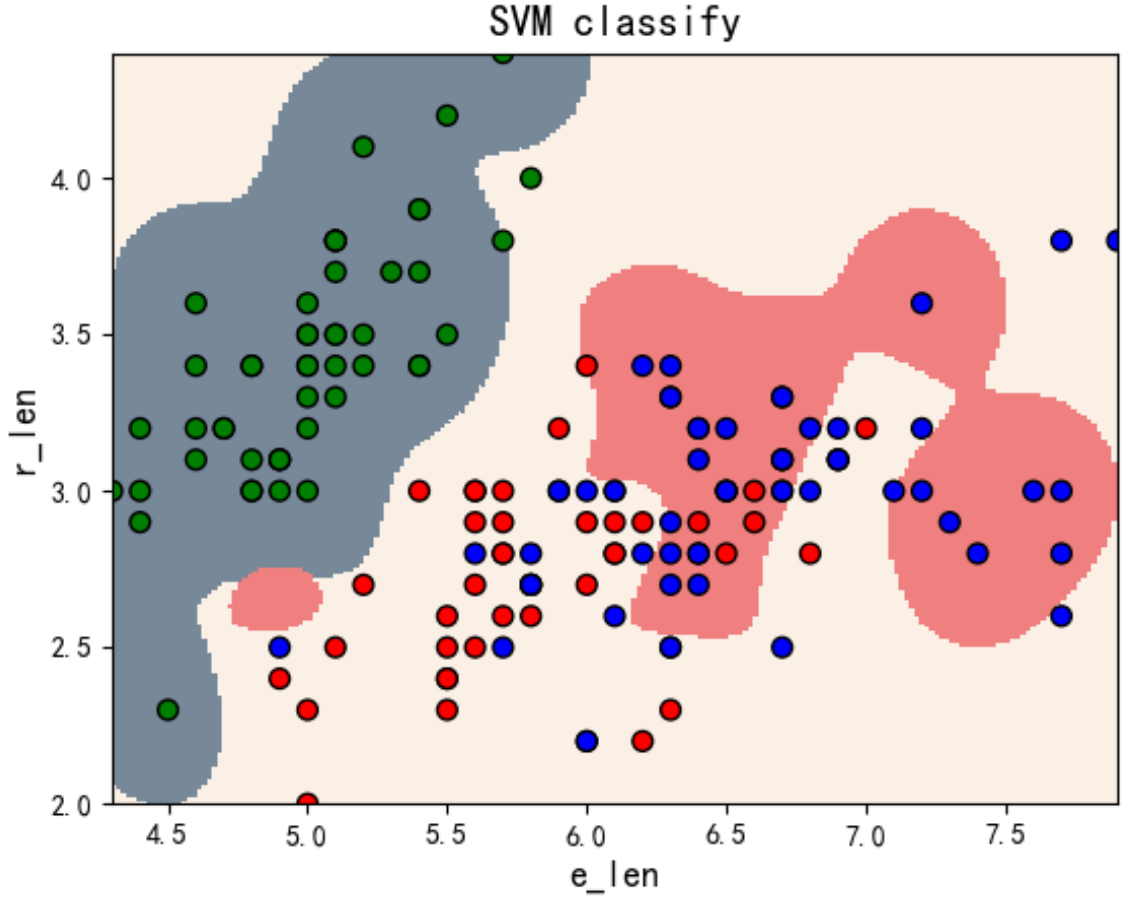
```
#coding:utf-8
import numpy as np
from sklearn import svm
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
def iris_type(s):
    it = {b'Iris-setosa':0,b'Iris-versicolor':1, b'Iris-virginica':2}   #Python2和3有点区
别
    return it[s]
#print(iris_type('Iris-setosa'))
path = 'Iris\UCI\iris.data'
data = np.loadtxt(path, dtype = float, delimiter = ',', converters = {4:iris_type})
#data
x, y = np.split(data, (4,), axis = 1)
x = x[:, :2]
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 1, test_size =
0.39)
clf = svm.SVC(C = 0.8, kernel = 'rbf', gamma = 20)
clf.fit(x_train, y_train.ravel())
print (clf.score(x_train, y_train))
y_hat = clf.predict(x_train)
#show_accuracy(y_hat, y_train, 'trainset')
print (clf.score(x_test, y_test))
y_hat = clf.predict(x_test)
#show_accuracy(y_hat, y_test, 'testset')
#print ('decision_function:\n', clf.decision_function(x_train))
print ('\npredict:\n', clf.predict(x_train))
x1_min, x1_max = x[:, 0].min(), x[:, 0].max()
x2_min, x2_max = x[:, 1].min(), x[:, 1].max()
x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j]
grid_test = np.stack((x1.flat, x2.flat), axis=1)
grid_hat = clf.predict(grid_test)
grid_hat = grid_hat.reshape(x1.shape)

mpl.rcParams['font.sans-serif'] = [u'SimHei']
mpl.rcParams['axes.unicode_minus'] = False

cm_light = mpl.colors.ListedColormap(['#778899', '#FAF0E6', '#F08080'])
cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])
plt.pcolormesh(x1, x2, grid_hat, cmap=cm_light)
plt.scatter(x[:, 0], x[:, 1], c=np.squeeze(y), edgecolors='k', s=50, cmap=cm_dark)
plt.scatter(x_test[:, 0], x_test[:, 1], s=120, facecolors='none', zorder=10)
plt.xlabel(u'e_len', fontsize=13)
plt.ylabel(u'r_len', fontsize=13)
plt.xlim(x1_min, x1_max)
plt.ylim(x2_min, x2_max)
plt.title(u'SVM classify', fontsize=15)
# plt.grid()
plt.show()
```

4.3 分类结果


Sepal Length
Sepal Width
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


SVM classify
r_len
e_len
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

5. 算法四——K-means

5.1 算法流程：

设定参数k，然后将事先输入的n个数据对象划分为k个聚类，使得所获得的每个聚类满足内部对象相似度较高，而不同聚类中的对象相似度较小。注：这个是不需要事先标记的，所以是非监督学习。
①适当选择k个类的初始中心；
②在第m次迭代中，对任意一个样本，求其到k个类中心的距离，将该样本归到距离最短的那个类；
③利用均值方法更新该类的中心；
④对于所有的C个聚类中心，如果利用（2）（3）的迭代法更新后，值保持不变，则迭代结束；否则继续迭代。

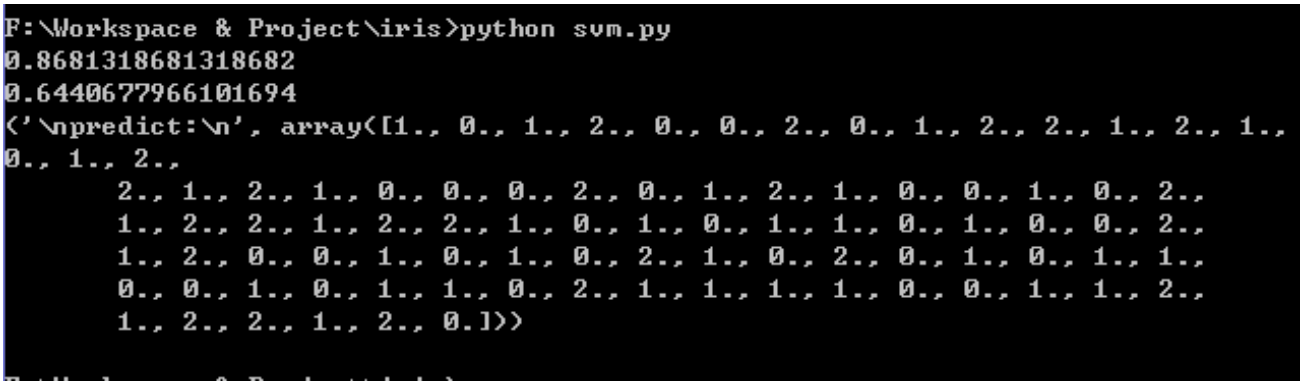
5.2 算法实现

```
import pandas as pd
import matplotlib.pyplot as plt
import random
import math

class main:
    def __init__(self):
        self.epochs = 10
        self.clusters = 3
        self.indexes = {'Iris-setosa':0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
        self.colors = {0: 'ro', 1: 'bo', 2: 'go'}
        self.labels = {0: 'Iris-setosa', 1: 'Iris-versicolor', 2: 'Iris-virginica'}
        self.centroid_colors = {0: 'rx', 1: 'bx', 2: 'gx'}
        data = self.load_data()
        data = self.split_data(data)
        centroids = self.init_centroids(data)
        centroids = self.train(data, centroids)
        centroids = self.rearrange_centroids(data, centroids)
        self.show_clusters(data, centroids)
    def train(self, data, centroids):
        print 'Epoch %s had error of %.3f.' % (0, self.cost(data, centroids))
        for epoch in range(self.epochs):
            klasses = [[] for x in range(self.clusters)]
            for tupl in data:
                klasses[self.klass(tupl, centroids)].append(tupl)
            new_centroids = []
            for klass in klasses:
                centroid_newX, centroid_newY = 0, 0
                for tupl in klass:
                    centroid_newX += tupl[0]
                    centroid_newY += tupl[1]
                new_centroids.append([centroid_newX / len(klass), centroid_newY /
len(klass)])
            centroids = new_centroids
            print 'Epoch %s had error of %.3f.' % (epoch, self.cost(data, centroids))
        return centroids
    def klass(self, tupl, centroids):
        distances = [self.distance(tupl, centroid) for centroid in centroids]
        return distances.index(min(distances))
    def cost(self, data, centroids):
        return sum([min([self.distance(tupl, centroid)**2 for centroid in centroids]) for
tupl in data])
    def distance(self, tupl, centroid):
        return math.sqrt((tupl[0]-centroid[0])**2 + (tupl[1]-centroid[1])**2)
    def init_centroids(self, data):
        return [[data[x][0], data[x][1]] for x in random.sample(range(0, len(data)),
self.clusters)]
    def split_data(self, data):
        return [[tupl[0], tupl[1], self.indexes[tupl[4]]] for tupl in data]
    def rearrange_centroids(self, data, centroids):
```

```
        bins = [[0]*3 for x in range(3)]
        rearranged_centroids = [0]*3
        for tupl in data:
            distances = [self.distance(tupl, centroid) for centroid in centroids]
            binn = distances.index(min(distances))
            bins[binn][tupl[2]] += 1
        for i in range(3):
            binn = bins[i]
            rearranged_centroids[binn.index(max(binn))] = centroids[i]
        return rearranged_centroids
    def show_clusters(self, data, centroids):
        for tupl in data:
            distances = [self.distance(tupl, centroid) for centroid in centroids]
            klass = distances.index(min(distances))
            plt.plot(tupl[0], tupl[1], self.colors[klass], markersize = 3)
        for centroid in centroids:
            plt.plot(centroid[0], centroid[1],
self.centroid_colors[centroids.index(centroid)], markersize = 12)
        plt.xlabel('Sepal Width')
        plt.ylabel('Sepal Length')
        plt.show()
    def analyse_clusters(self, data, centroids):
        for tupl in data:
            plt.plot(tupl[0], tupl[1], self.colors[tupl[2]], markersize = 3)
        for centroid in centroids:
            plt.plot(centroid[0], centroid[1], 'kx',markersize = 12)
        #plt.title('Iris Sepal Width vs Length')
        plt.xlabel('Sepal Width')
        plt.ylabel('Sepal Length')
        plt.show()
    def analyse_data(self, data):
        first = [True]*3
        for tupl in data:
            if first[tupl[2]]:
                plt.plot(tupl[0], tupl[1], self.colors[tupl[2]], markersize = 3, label =
self.labels[tupl[2]])
                first[tupl[2]] = False
            else:
                plt.plot(tupl[0], tupl[1], self.colors[tupl[2]], markersize = 3)
        plt.title('Iris Sepal Width vs Length')
        plt.xlabel('Sepal Width')
        plt.ylabel('Sepal Length')
        plt.legend()
        plt.show()
    def load_data(self):
        return pd.read_csv("Iris\iris.csv").values
runProgram = main()
```

5.3 分类结果



通过它们的萼片长度和萼片宽度绘制图形，可以看到实际的簇之间的关系。Iris-setosa可与其他两个类别线性分离。然而，Iris-virginica和Iris-versicolor不是线性可分的，尽管它们居中的位置存在明显的差异。我实现了K-means聚类算法，基于原始数据，很明显我们应该使用3个集群，随机选择它们以具有与原始数据点中的3个相同的坐标，它们聚集得相当好。

6. 算法四——DNN

6.1 算法简介

人工神经网络模型具有强大的学习能力、适应能力、计算效率，可以良好地模拟出输入空间到输出空间的非线性映射关系，在很多应用领域已经取得了令人瞩目的成果。全连接神经网络理论部分的精髓在于前向传播和反向传播，全连接神经网络模型一般包含输入层、若干个隐藏层、输出层，每层包含数目不等的节点。前向传播是指在给定训练数据和模型参数的情况下，通过输入层的数据层层传播至输出层，与真实值对比并计算出误差的过程。如果误差不达要求，则进入反向传播过程，这个过程主要是基于梯度下降法的，目的是在给定训练数据和损失函数的前提下，修改连接各节点的边上的权值使损失函数达到最小。神经网络模型的反向传播机制可以类比于飞机的发动机，地位十分重要，是神经网络模型具有学习能力的关键。

6.2 算法实现

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
import urllib
import numpy as np
import tensorflow as tf
# Data sets
IRIS_TRAINING = "iris_training.csv"
IRIS_TRAINING_URL = "http://download.tensorflow.org/data/iris_training.csv"
IRIS_TEST = "iris_test.csv"
IRIS_TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
def main():
  # If the training and test sets aren't stored locally, download them.
  if not os.path.exists(IRIS_TRAINING):
    raw = urllib.urlopen(IRIS_TRAINING_URL).read()
    with open(IRIS_TRAINING, "w") as f:
      f.write(raw)
  if not os.path.exists(IRIS_TEST):
    raw = urllib.urlopen(IRIS_TEST_URL).read()
    with open(IRIS_TEST, "w") as f:
      f.write(raw)
  # Load datasets.
  training_set = tf.contrib.learn.datasets.base.load_csv_with_header(
      filename=IRIS_TRAINING,
      target_dtype=np.int,
      features_dtype=np.float32)
  test_set = tf.contrib.learn.datasets.base.load_csv_with_header(
      filename=IRIS_TEST,
      target_dtype=np.int,
      features_dtype=np.float32)
  # Specify that all features have real-value data
  feature_columns = [tf.contrib.layers.real_valued_column("", dimension=4)]
  # Build 3 layer DNN with 10, 20, 10 units respectively.
  classifier =
tf.contrib.learn.DNNClassifier(feature_columns=feature_columns,hidden_units=[10, 20,
10],n_classes=3,model_dir="/tmp/iris_model")
  # Define the training inputs
  def get_train_inputs():
    x = tf.constant(training_set.data)
    y = tf.constant(training_set.target)
    return x, y
  # Fit model.
  classifier.fit(input_fn=get_train_inputs, steps=2000)
  # Define the test inputs
  def get_test_inputs():
    x = tf.constant(test_set.data)
    y = tf.constant(test_set.target)
    return x, y
  # Evaluate accuracy.
```

```
  accuracy_score = classifier.evaluate(input_fn=get_test_inputs,steps=1)["accuracy"]
  print("\nTest Accuracy: {0:f}\n".format(accuracy_score))
  # Classify two new flower samples.
  def new_samples():
    return np.array(
      [[6.4, 3.2, 4.5, 1.5],
       [5.8, 3.1, 5.0, 1.7]], dtype=np.float32)
  predictions = list(classifier.predict(input_fn=new_samples))
  print(
      "New Samples, Class Predictions:    {}\n"
      .format(predictions))
if __name__ == "__main__":
    main()
```

6.3分类结果

Test Accuracy: 0.966667