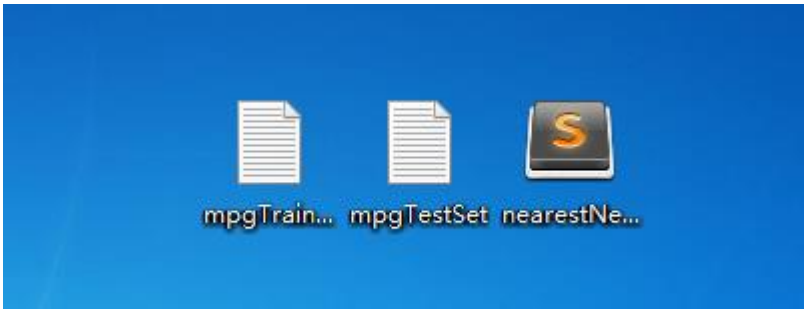
推荐系统作业三

It is fun playing with data sets and trying different methods. I obtained the Iris and MPG data sets from the UCI Machine Learning Repository (archive.ics.uci.edu/ml). I encourage you to go there, download a data set or two, convert the data to match data format, and see how well our classifier does.

班级：商务 1402 班

姓名：梁天华、李卫林

1. MPG 数据集下载地址：http://archive.ics.uci.edu/ml/datasets/auto+mpg
2. 文件如下：



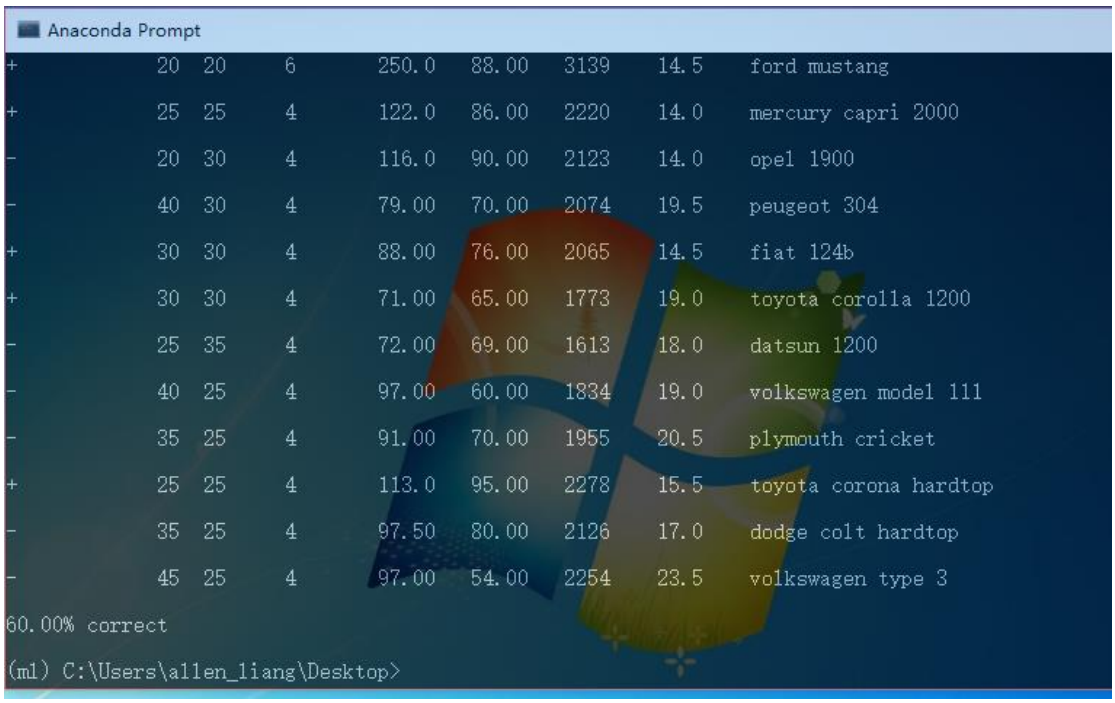
3.分类器代码如下：

```
class Classifier:
    def __init__(self, filename):
        self.medianAndDeviation = []
        # reading the data in from the file
        f = open(filename)
        lines = f.readlines()
        f.close()
        self.format = lines[0].strip().split('\t')
        self.data = []
        for line in lines[1:]:
            fields = line.strip().split('\t')
            ignore = []
            vector = []
            for i in range(len(fields)):
                if self.format[i] == 'num':
                    vector.append(float(fields[i]))
                elif self.format[i] == 'comment':
                    ignore.append(fields[i])
                elif self.format[i] == 'class':
                    classification = fields[i]
            self.data.append((classification, vector, ignore))
        self.rawData = list(self.data)
        # get length of instance vector
        self.vlen = len(self.data[0][1])
        # now normalize the data
        for i in range(self.vlen):
            self.normalizeColumn(i)

    def getMedian(self, alist):
        """return median of alist"""
        if alist == []:
```

```
            return []
        blist = sorted(alist)
        length = len(alist)
        if length % 2 == 1:
            # length of list is odd so return middle element
            return blist[int(((length + 1) / 2) -    1)]
        else:
            # length of list is even so compute midpoint
            v1 = blist[int(length / 2)]
            v2 =blist[(int(length / 2) - 1)]
            return (v1 + v2) / 2.0

    def getAbsoluteStandardDeviation(self, alist, median):
        """given alist and median return absolute standard deviation"""
        sum = 0
        for item in alist:
            sum += abs(item - median)
        return sum / len(alist)

    def normalizeColumn(self, columnNumber):
        """given a column number, normalize that column in self.data"""
        # first extract values to list
        col = [v[1][columnNumber] for v in self.data]
        median = self.getMedian(col)
        asd = self.getAbsoluteStandardDeviation(col, median)
        #print("Median: %f      ASD = %f" % (median, asd))
        self.medianAndDeviation.append((median, asd))
        for v in self.data:
            v[1][columnNumber] = (v[1][columnNumber] - median) / asd

    def normalizeVector(self, v):
        """We have stored the median and asd for each column.
        We now use them to normalize vector v"""
        vector = list(v)
        for i in range(len(vector)):
            (median, asd) = self.medianAndDeviation[i]
            vector[i] = (vector[i] - median) / asd
        return vector

    def Xupt_Distance(self, vector1, vector2):
        """Computes the Manhattan distance."""
        from math import sqrt
        #return sum(map(lambda v1, v2: sqrt(abs(v1 - v2)), vector1, vector2))
        xupt = 0.5
```

```
            return sum(map(lambda v1, v2: pow(abs(v1 - v2),xupt), vector1, vector2))

        def nearestNeighbor(self, itemVector):
            """return nearest neighbor to itemVector"""
            return min([ (self.Xupt_Distance(itemVector, item[1]), item)
                                for item in self.data])

        def classify(self, itemVector):
            """Return class we think item Vector is in"""
            return(self.nearestNeighbor(self.normalizeVector(itemVector))[1][0])

    def test(training_filename, test_filename):
        """Test the classifier on a test set of data"""
        classifier = Classifier(training_filename)
        f = open(test_filename)
        lines = f.readlines()
        f.close()
        numCorrect = 0.0
        for line in lines:
            data = line.strip().split('\t')
            vector = []
            classInColumn = -1
            for i in range(len(classifier.format)):
                    if classifier.format[i] == 'num':
                        vector.append(float(data[i]))
                    elif classifier.format[i] == 'class':
                        classInColumn = i
            theClass= classifier.classify(vector)
            prefix = '-'
            if theClass == data[classInColumn]:
                # it is correct
                numCorrect += 1
                prefix = '+'
            print("%s    %12s    %s" % (prefix, theClass, line))
        print("%4.2f%% correct" % (numCorrect * 100/ len(lines)))

    test("mpgTrainingSet.txt", "mpgTestSet.txt")
```

4.预测准确率如下：



预测准确率为 60%，比课本的 56% 提高了 4%

	classifier built		
data set	using no normalization	using the formula on previous page	using Modified Standard Score
Athletes	80.00%	60.00%	80.00%
Iris	100.00%	83.33%	93.33%
MPG	32.00%	36.00%	56.00%