交通数据挖掘课程报告

题目：应用机器学习识别手机传感器数据交通方式

姓　　名：王　倩　妮

学　　号：2015112956

班　　级：　交通 2015-02 班

任课教师：杨飞、曹鹏、熊耀华

2017 年 12 月 20 日

目录

一、问题描述

训练数据文件夹中是已标定好的出行个体在出行过程中的时间（s）、经纬度（度）、速度（m/s）和对应交通方式（1-步行，2-自行车，3-公交车，4-小汽车）。

请查阅相关文献资料，选取合适的特征属性，利用有监督机器学习算法（如人工神经网络、支持向量机、决策树、朴素贝叶斯等）训练出有效的交通方式分类器，最后对测试数据文件夹中每个出行个体的交通方式进行测试分类，识别出出行方式模式【如步行-小汽车-步行（1-4-1）、步行-自行车-公交车-步行（1-2-3-1）等】，以及交通方式发生切换的时间点。

二、数据挖掘方法选定

为解决本问题，首先需要选定较为合适的数据挖掘方法，因此本部分首先介绍目前机器学习的几种分类方式，而后针对本问题选择较为合适的数据挖掘方法。

2.1 机器学习方法分类

- 按学习方式分类

✓ 有监督学习：有监督学习是从标签化训练数据集中推断出函数的机器学习任务。

✓ 无监督学习：无监督学习是根据类别未知(没有被标记)的训练样本解决模式识别中的问题。

✓ 半监督学习：半监督学习使用大量的未标记数据，以及同时使用标记数据，来进行模式识别工作。当使用半监督学习时，将会要求尽量少的人员来从事工作，同时，又能够带来比较高的准确性。

- 按问题归属分类

✓ 回归：有监督的一种，确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。

✓ 分类：与回归问题本质类似，将回归后的结果离散化。

✓ 聚类：无监督的一种，是依靠某些属性自动聚集成一个一个簇。簇与簇之间无交集，且各个簇的并集即为整个样本整体的方法。

✓ 降维：降维指采用某种映射方法,将原高维空间中的数据点映射到低维度的空间中。

如下图所示，是 Python 中的机器学习模块 scikit-learn 提供给使用者的方法选择步骤，在方法选择过程中，需要结合自己面对的实际问题、现有的资源选择合适的机器学习方法。

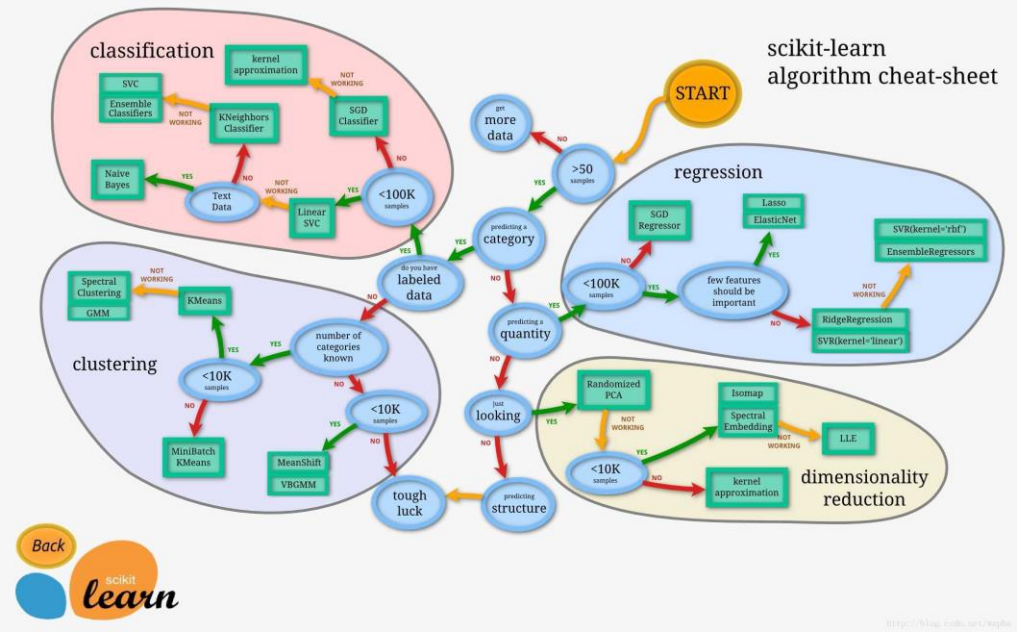


图 1 scikit-learn algorithm cheat-sheet

分析本问题现有数据集资源与预期目标，可以确定本问题是一个有监督的分类问题。在接下来的算法选择中，主要选取用于此类问题的方法。

2.2 方法选定

有监督的分类问题常用方法主要有以下几种：

- ✓ Logistics 回归：
- ✓ K 近邻算法：KNN 方法是分类中最为简便的方法，思想为：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。
- ✓ 决策树：决策树学习是一种逼近离散值目标函数的方法,在这种方法中学习到的函数被表示为一棵决策树。决策树一般都是自上而下的来生成的。每个决策或事件（即自然状态）都可能引出两个或多个事件，导致不同的结果。
- ✓ 朴素贝叶斯：朴素贝叶斯法是基于贝叶斯定理与特征条件独立假设的分类方法。常用于文本分类。
- ✓ 支持向量机：支持向量机方法是建立在统计学习理论的 VC 维理论和结构风险最小原理基础上的，根据有限的样本信息在模型的复杂性（即对特定训练样本的学习精度）和学习能力（即无错误地识别任意样本的能力）之间寻求最佳折中，以求获得最好的推广能力。在解决小样本、非线性及高维模式识别中表现出许多特有的优势。
- ✓ 神经网络：人工神经网络是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

在本问题实践的过程中，考虑本问题涉及到时间序列，因此起初打算采用神经网络中循环神经网络(RNN)的方法，由于遗忘记忆门参数不易设置，导致 RNN 并不能有效

解决此问题。

在尝试循环神经网络方法无果后，选择采用基本的人工神经网(NN)尝试解决，使用两种方法确定神经网络输入与输出。构建神经网络并训练后计算正确率，若全样本训练，全样本计算，随着训练步骤的增加，正确率收敛在 70%左右；但部分数据训练，另一部分数据进行检测后发现准确率最终收敛在 45%左右。在调整参数后结果并未有大幅度改变，因此接下来换用了支持向量机方法。

采用支持向量机方法，并在输入输出等方面进行了改进，随机选取 80%数据用于训练，其余 20%用于测试，发现全局准确率可达 90%以上，因此确定 SVM 为解决此问题的较好方法。

接下来，首先介绍神经网络方法的操作步骤及初步结果，再介绍支持向量机方法的操作步骤与结果，并用此方法得到的模型用于预测，解决此问题。

三、原始数据集介绍

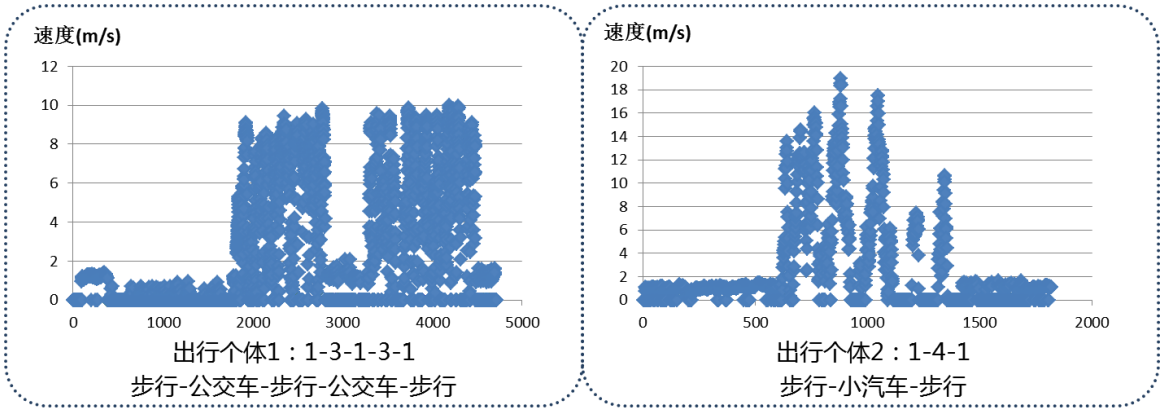
观察原始数据，原始数据分为训练数据集与测试数据集。

- ✓ 训练数据集包含共 6 个出行个体、17130 条数据的个体编号、时间、经度、纬度、速度与经人工标定的交通方式。
- ✓ 测试数据包含共 44 个出行个体的个体编号、时间、经度、纬度、速度数据。

其中交通方式的对应规则为：1-步行，2-自行车，3-公交车，4-小汽车。数据采集频率为 1 次/s。

由于中文在数据处理上的不便性，因此，将以上数据项标题转化为“id_number”、“time”、“longitude”、“latitude”、“speed”、“means”，并用于接下来模型的训练与预测。

绘制 6 个训练样本的速度随记录数目的变化图像，如下图所示。在其中发现规律，并为模型输入输出的确定提供依据。



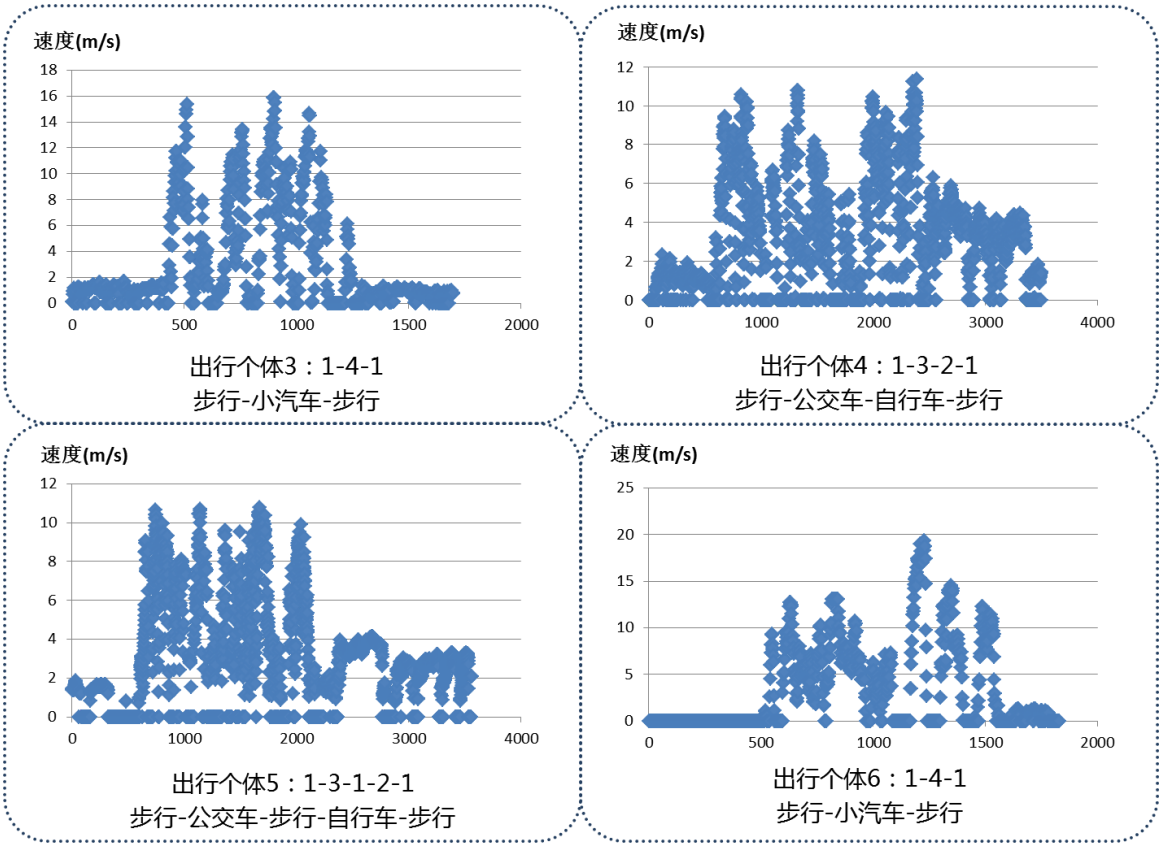


图 2 训练数据集速度变化散点图

四、神经网络方法

4.1 神经网络方法搭建环境

- 分类器整体编程语言：Python
- 神经网络框架：TensorFlow
- 矩阵运算：Numpy
- 绘图：Matplotlib
- IDE：Jupyter Notebook

4.2 数据预处理与模型输入输出

应用神经网络的方法，确定模型的输入与输出尤为重要。首先，观察第三部分所得的原始数据，我们可以发现以下结果：

1) 步行速度平稳且处于较低水平
2) 自行车速度高于步行速度
3) 公交车与小汽车速度较高
4) 公交车与小汽车的波动性存在差异

因此，我们可以产生以下两条初步结论：

1) 交通方式在短时间内不会频繁变化（不存在频繁跳动）

2) 交通方式与速度、速度变化有关

考虑以上因素，由于数据前后相关性的存在，所以不能直接输入原始数据，而是应该在考虑相关性的基础上处理速度数据。而数据处理的基本思想为：分组思想。在这种思想的指导下，进行了两种尝试。

在两种尝试下分别得到了两种输入、输出形式，在神经网络接下来的部分均通过两种方法实现。

- **尝试一：**

- 分组方法：每个 SCALE 长度的数据分为一组。17130 条数据分为(17130/SCALE)组。
- 输入数据量： (17130/SCALE)条数据，即每组总结为 1 条数据。
- 输入内容：
  - 每个 SCALE 内的平均速度 $\bar{v}$
  - 每个 SCALE 内的最大速度 v_{max}
  - 每个 SCALE 内的速度标准差 $std(v)$
- 输出数据量：(17130/SCALE)条数据，即每组总结为 1 条数据
- 输出内容：转化为 one-hot 模式的每个 SCALE 对应的交通方式

[[1,0,0,0],
[0,1,0,0],
[0,0,1,0],
[0,0,0,1]]

- 输入输出内容确定步骤：
  - 确定 SCALE=60（每分钟进行统计）
  - STEP1：选取文件，得到速度列、出行方式列
  - STEP2：计算此文件能够分为几个组，每组长度为 SCALE
  - STEP3：计算训练数据中每个组的神经网络输入
  - STEP4：计算训练数据中每个组的神经网络输出（占比最大）
  - STEP5：转化神经网络输出为 one-hot 形式
  - STEP6：得到用于神经网络训练使用的数据 data_x 与 data_y

- **尝试二：**

- 分组方法：滚动分组思想，每条数据与之前(SCALE)与后(SCALE)分为一组，组长 2SCALE。
- 输入数据量: 17130 条数据
- 输入内容：
  - 滚动每 2SCALE 内的平均速度 $\bar{v}$
  - 滚动每 2SCALE 内的最大速度 v_{max}
  - 滚动每 2SCALE 内的速度标准差 $std(v)$
- 输出数据量：17130 条数据
- 输出内容：转化为 one-hot 模式的每个 SCALE 对应的交通方式
- 输入输出内容确定步骤：
  - 确定 SCALE=60（滚动组长=120）

- STEP1：选取文件，得到速度列、出行方式列
- STEP2：按滚动规则，计算训练数据中每条数据的神经网络输入[平均速度，最大速度，速度标准差]
- STEP3：直接输入每条数据对应的出行方式
- STEP4：转化神经网络输出为 one-hot 形式
- STEP5：得到用于神经网络训练使用的数据 data_x 与 data_y

4.3 神经网络方法工作原理与构建参数

4.3.1 神经网络方法工作原理

神经网络是一种应用类似于大脑神经突触联接的结构进行信息处理的数学模型。

人工神经网络无需事先确定输入输出之间映射关系的数学方程，仅通过自身的训练，学习某种规则，在给定输入值时得到最接近期望输出值的结果。作为一种智能信息处理系统，人工神经网络实现其功能的核心是算法。BP 神经网络是一种按误差反向传播(简称误差反传)训练的多层前馈网络，其算法称为 BP 算法，它的基本思想是梯度下降法，利用梯度搜索技术，以期使网络的实际输出值和期望输出值的误差均方差为最小。

基本 BP 算法包括信号的前向传播和误差的反向传播两个过程。即计算误差输出时按从输入到输出的方向进行，而调整权值和阈值则从输出到输入的方向进行。正向传播时，输入信号通过隐含层作用于输出节点，经过非线性变换，产生输出信号，若实际输出与期望输出不相符，则转入误差的反向传播过程。误差反传是将输出误差通过隐含层向输入层逐层反传，并将误差分摊给各层所有单元，以从各层获得的误差信号作为调整各单元权值的依据。通过调整输入节点与隐层节点的联接强度和隐层节点与输出节点的联接强度以及阈值，使误差沿梯度方向下降，经过反复学习训练，确定与最小误差相对应的网络参数(权值和阈值)，训练即告停止。此时经过训练的神经网络即能对类似样本的输入信息，自行处理输出误差最小的经过非线形转换的信息。

4.3.2 神经网络参数选择

- **尝试一：**

- 神经层数：4 层
- 每层的神经元数：10 个元
- 激活函数选取：softmax 函数
- 梯度下降优化器选取：GradientDescentOptimizer
- 学习率：0.001
- 全局/随机学习：全局
- 循环次数：10000 次

- **尝试二：**

- 神经层数：4 层
- 每层的神经元数：10 个元
- 激活函数选取：softmax 函数
- 梯度下降优化器选取：GradientDescentOptimizer
- 学习率：0.001
- 全局/随机学习：全局
- 循环次数：25000 次

4.4 模型训练结果

● 尝试一：

记录随着神经网络训练步骤的增加，Loss 值与 Accuracy 值的变化，运用尝试一的方式分组方式训练 10000 步，最终得到准确率为：0.766784。

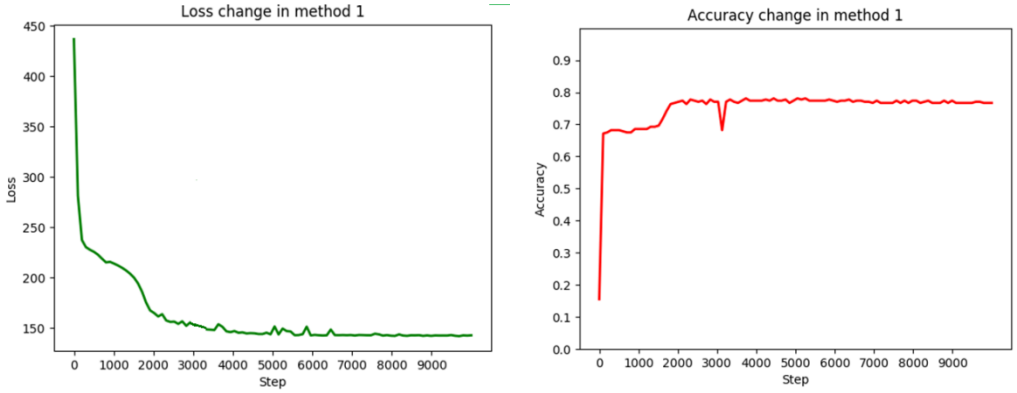


图 3 神经网络尝试一 Loss 与 Accuracy 变化

● 尝试二：

记录随着神经网络训练步骤的增加，Loss 值与 Accuracy 值的变化，运用尝试二的方式分组方式训练 25000 步，最终得到准确率为：0.692002。

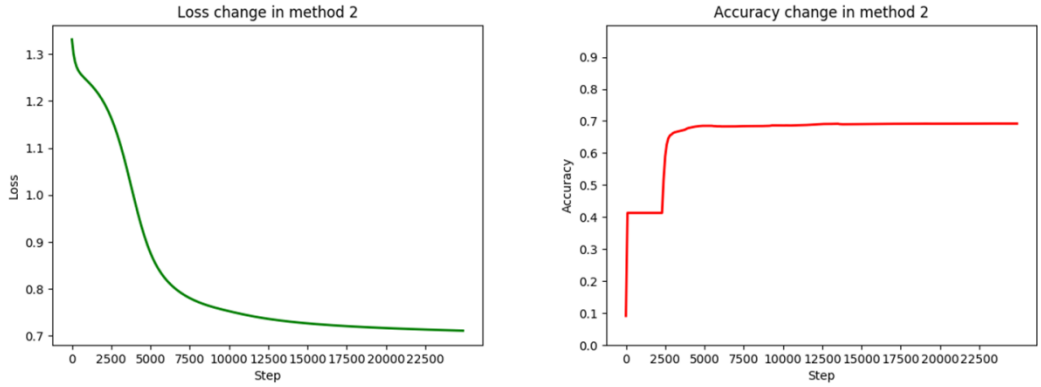


图 4 神经网络尝试二 Loss 与 Accuracy 变化

4.5 TensorFlow 操作步骤

- STEP1：定义全局变量

- 输入数据尺度
- 输出数据尺度
- 学习效率

➢ STEP2：定义添加层函数

```
def add_layer(input_data,in_size,out_size,activation_function=None):
    Weight=tf.Variable(tf.random_normal([in_size,out_size]))
    biases=tf.Variable(tf.zeros([1,out_size])+0.1)
    wx_b=tf.matmul(input_data,Weight)+biases
    if activation_function==None:
        outputs=wx_b
    else:
        outputs=activation_function(wx_b)
    return outputs
```

➢ STEP3：定义输入、输出数据的 placeholder

```
X = tf.placeholder(tf.float32, [None, X_IN_SIZE])
Y = tf.placeholder(tf.float32, [None, N_CLASS])
```

➢ STEP4：添加层（4 层网络，每层网络宽度为 10）

```
LAYER_UNIT=10
hidden_layer1=add_layer(X,X_IN_SIZE,LAYER_UNIT,activation_function=tf.nn.softmax)
hidden_layer2=add_layer(hidden_layer1,LAYER_UNIT,LAYER_UNIT,activation_function=tf.nn.softmax)
hidden_layer3=add_layer(hidden_layer2,LAYER_UNIT,LAYER_UNIT,activation_function=tf.nn.softmax)
y=add_layer(hidden_layer3,LAYER_UNIT,N_CLASS,activation_function=tf.nn.softmax)
```

➢ STEP5：定义 Loss(又称为 cross_entropy or cost)

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(Y* tf.log(y),reduction_indices=[1]))
```

➢ STEP6：定义梯度下降优化器

```
train_step = tf.train.GradientDescentOptimizer(LR).minimize(cross_entropy)
```

➢ STEP7：定义神经网络分类准确率

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction , "float"))
```

➢ STEP8：保存模型

```
saver = tf.train.Saver()
save_path = saver.save(sess, "my_net/save_net.ckpt")
```

➢ STEP9：完成计算图的定义并运算

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for i in range(25000):
    _,loss_=sess.run([train_step,cross_entropy], feed_dict={X: data_x, Y: data_y})
    accuracy_=sess.run(accuracy, feed_dict={X: data_x, Y: data_y})
    if i % 100== 0:
        print ('trainstep','%s'%i,'loss=',loss_,'|accuracy=',accuracy_)
```

4.6 模型用于预测

若神经网络模型训练成功且准确率高，则可以采用以下步骤进行预测。但实际过程中由于神经网络方法准确率较低，因此并未进行预测步骤。

- STEP1：选取测试文件，得到速度列、出行方式列
- STEP2：按滚动规则，计算训练数据中每条数据的神经网络输入[平均速度，最大速度，速度标准差]
- STEP3：得到输入数据 data_x
- STEP4：带入 saver 中保存的训练好的神经网络进行预测，并将输出结果按概率高低转化为出行方式。
- STEP5：使用平滑手段进行平滑，去除跳变的数据，保证连续性/每隔一定长度取众数，保证连续性
- STEP6：依据每个测试数据对应的交通方式，得出交通方式切换时间点

4.7 需注意的问题与改进方向

4.7.1 注意问题

1) 同样的输入与输出，神经网络的效果与参数的设定有很大关系。（神经网络层数、神经层的宽度、学习效率、激活函数选择、梯度下降优化器选择、损失函数度量方式……）
2) 一定要对于 tensor 的维度有着清楚的认知。
3) 学习效率过大可能会出现 loss=nan 的状况，此时应当以 10 的数量级递减。

4.7.2 改进方向

神经网络的调参工作需要花费大量时间，若希望得到较高的准确率，还需进一步进行参数的调整以使网络达到更好的分类效果。

五、支持向量机方法

5.1 支持向量机方法搭建环境

- 分类器整体编程语言：Python
- SVM 借助第三方库：scikit-learn.svm.SVC
- 矩阵运算：Numpy
- 绘图：Matplotlib
- 数据处理：Pandas
- IDE：Jupyter Notebook

5.2 数据预处理与模型输入输出

5.2.1 输入与输出

与神经网络方法的输入有一定差距，在支持向量机方法中，我选择滚动长度

SCALE=60,即每个数据的前 60 个数据与后 60 个数据进行运算，每个出行个体的前 SCALE 与后 SCALE 个数据分别处理。并增加了数据输入项。

每条用于训练的数据将输入一下 5 项内容：

- 原始速度
- 原始速度时间序列下的一次指数平滑值，阻尼=0.5
- 滚动每 2SCALE 内的平均速度 $\bar{v}$
- 滚动每 2SCALE 内的最大速度 v_{max}
- 滚动每 2SCALE 内的速度标准差 $std(v)$

由于支持向量机自身方法的优势，无需将输出转为 one-hot 的形式，因此输出的内容为：

- 原始标定的出行方式（以数字代替）[1],[2],[3],[4]

6 个出行个体的全部 5 项数据汇总为 data_x，所有标定的出行方式汇总为 data_y。

5.2.2 训练集与验证集

在以上 6 个出行个体构成的全部样本中，随机选取 80%作为训练数据，这些训练数据输入与输出帮助确定 SVM 构建的方式。剩余 20%为验证数据，用这部分数据代入训练好的模型，得出预测结果，用预测结果与输出结果（原始标定）进行比对，计算准确率。随机选取训练集与验证集，比直接指定更为科学、合理。

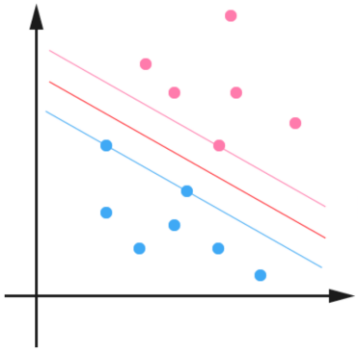
5.3 支持向量机方法工作原理与参数构建

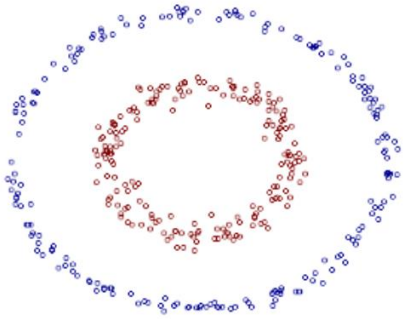
5.3.1 支持向量机原理

支持向量机是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，即支持向量机的学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

通俗来讲，支持向量机就是通过一个 n-1 维超平面将存在于 n 维空间中的两类数据分开，并使两类数据到这个超平面的距离达到最达。对于不能直接用线分开的两类数据，通常的做法便是使用核函数将低维空间的点映射到高维空间，再寻找超平面分隔两类数据。

对于本问题来讲，存在四类出行方式，但支持向量机是一种二元分类器，在实际处理解决问题的过程中需要进行多次分类达到分成四类的目的。借助 Python 机器学习库 scikit-learn 可以解决这些复杂的操作。





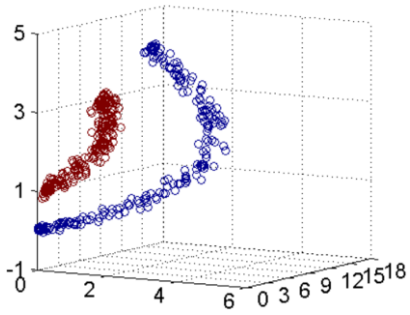


图 5 二维空间内线性分割案例　图 6 无法用 n-1=1 维超平面直接分割则需要核函数映射到高维空间

5.3.2 模型选用参数

在支持向量机方法运用的过程中，采用默认参数据即可达到较高准确率，调整部分参数后发现结果并没有明显变化，因此没有继续进行参数调整。SVC 运用及默认参数如下列代码所示：

```
clf = SVC()
clf.fit(data_xnew, data_ynew.ravel())
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

其中各参数含义此处不再赘述，详见 scikit-learn 官方说明文件。

5.4 模型训练验证结果

5.4.1 准确率定义

- 全局准确率：20%验证集中的预测结果与原始标定的数据对比所得的正确个数除以 20%验证集的总数据条数。
- 各方式准确率：依照原始标定结果划分为 4 个集合，分别计算四个集合中识别准确的个数除以集合数据条目总数。

5.4.2 验证结果

依照上述步骤，进行模型构架与训练后，得到的准确率结果如下所示：

- 全局准确率 0.9246935201401051
- 步行准确率： 0.9376313945339874
- 自行车准确率： 0.9604938271604938
- 公交车准确率： 0.9682835820895522
- 小汽车准确率： 0.7720306513409961

由以上结果可见使用 SVM 方法得到的准确率很高，运行多次验证均在 92%-94%之间，但各方式识别准确率仍有一定差距，小汽车出行的识别准确率较低，多次运行均在

76%-80%之间。

5.5 模型用于预测

测试集共有 44 个出行样本，输入模型的数据需经与训练数据同样的处理方法，整合为“原始速度、一次平滑速度读、平均值、最大值、标准差”的形式。

而后代入训练好的 SVM 模型中预测，由于 scikit-learn 机器学习库的存在，此步骤实现较为容易，因此此处不再赘述。结果输出为 excel 格式。

换乘时间的确定需首先观察预测结果，对预测结果进行进一步处理，并以一定规则定义“换乘时间”，而后编程实现换乘时间的输出。

5.6 结果分析

对测试集前 10 个出行个体直接通过 SVM 模型预测，并将出行方式结果绘制成如下图所示的图像。

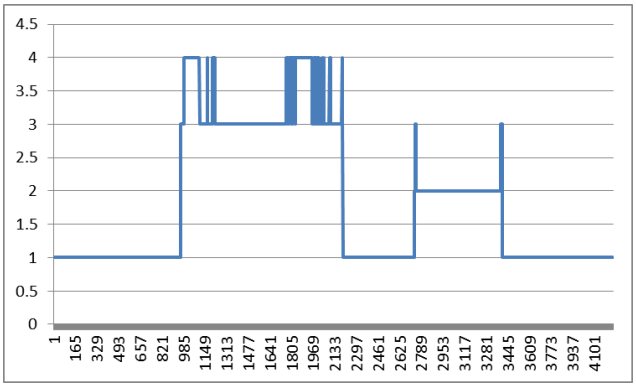


图 7 测试集个体 1 出行方式变化预测结果图

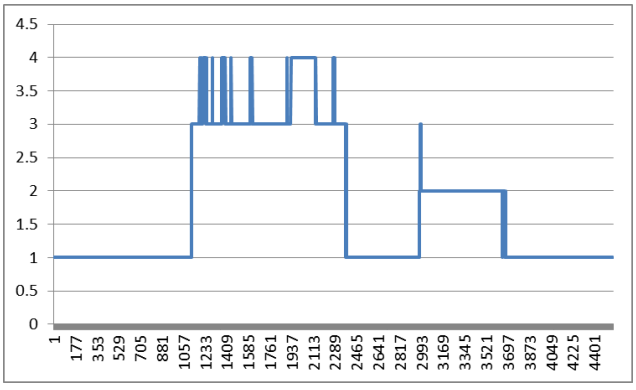


图 8 测试集个体 2 出行方式变化预测结果图

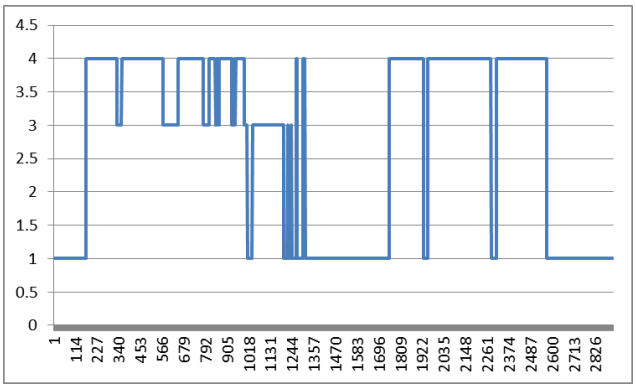


图 9 测试集个体 3 出行方式变化预测结果图

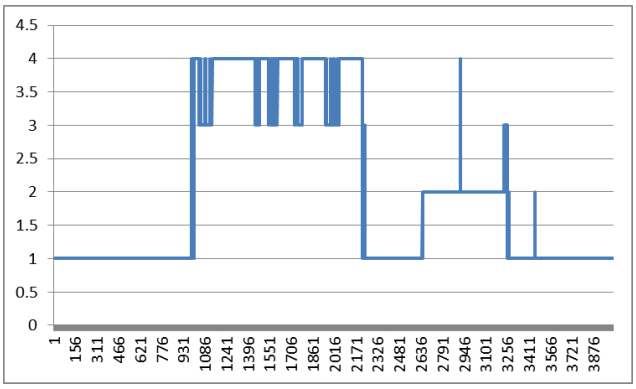


图 10 测试集个体 4 出行方式变化预测结果图

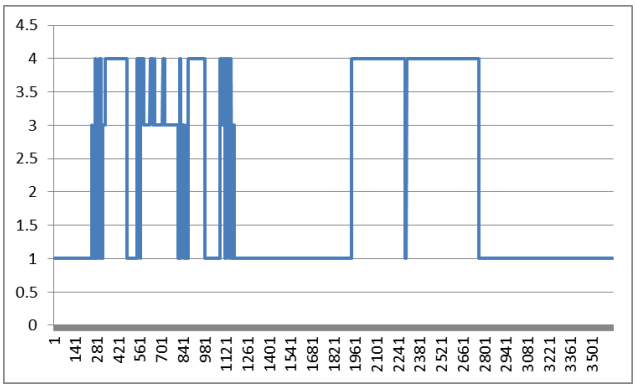


图 11 测试集个体 5 出行方式变化预测结果图

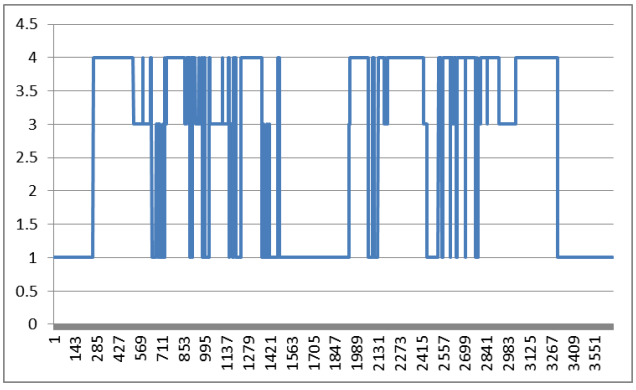


图 12 测试集个体 6 出行方式变化预测结果图

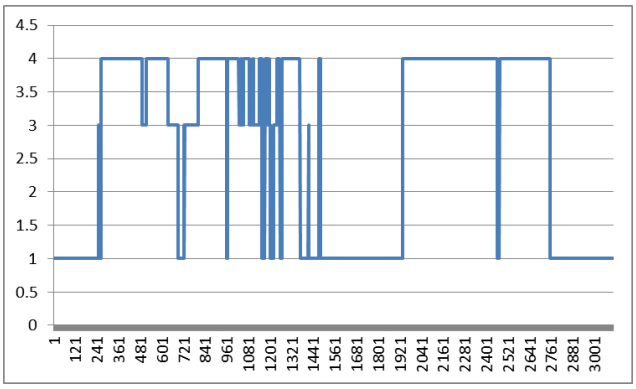


图 13 测试集个体 7 出行方式变化预测结果图

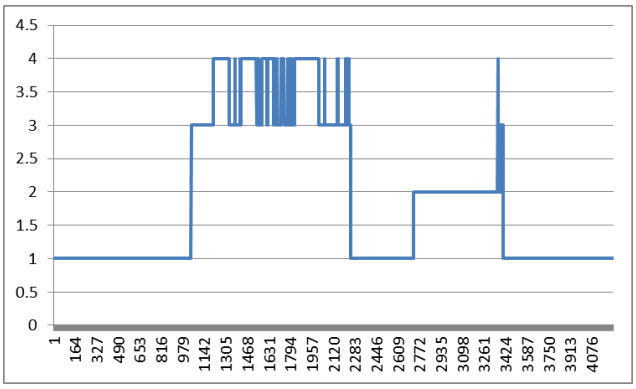


图 14 测试集个体 8 出行方式变化预测结果图

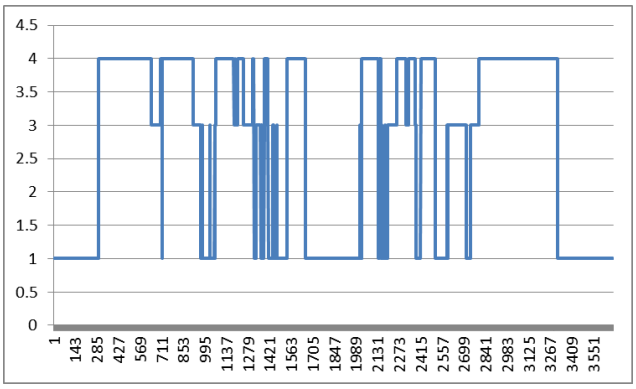


图 15 测试集个体 9 出行方式变化预测结果图

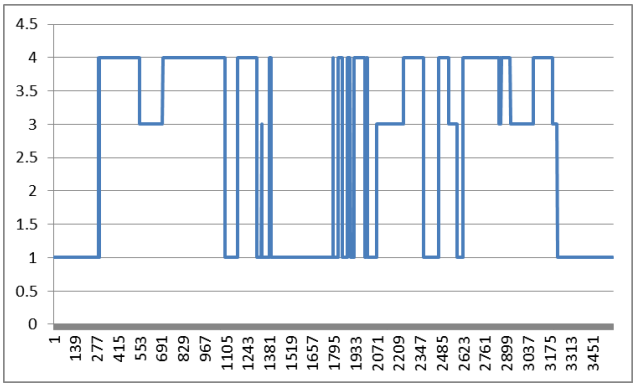


图 16 测试集个体 10 出行方式变化预测结果图

将所有预测结果进行进一步众数/平滑处理，预测出行方式，并识别方式切换时间，得到的部分出行方式及切换时间如下表所示：

表 1 部分测试数据结果分析

出行个体编号	预测出行方式	切换时间
1	1-3-1-2-1	964,2195,2738,3406
2	1-3-1-2-1	1126,2378,2980,3676
3	1-4-1-4-1	169,1312,1751,2573
4	1-4-1-2-1	992,2215,2647,3271,
5	1-4-1-4-1	269,1177,1940,2765
6	1-4-1-4-1	260,1369,1944,3308
7	1-4-1-4-1	262,1368,1938,2754
8	1-3-1-2-1	1042,2248,2729,3406
9	1-4-1-4-1	294,1659,2025,3315
10	1-4-1-4-1	288,1302,1821,3224

六、数据挖掘课程心得体会

一学期的数据挖掘课程接近尾声，一学期以来，三位老师向我们介绍了各自研究领域的数据挖掘工作，让我们了解到大数据背景下交通工程行业的从业者进行的相关工作。曹鹏老师主要结合滴滴智慧信号灯向我们讲解其中的方法，熊耀华老师主要介绍了神经网络内容，杨飞老师结合手机传感器数据的案例并在授课的过程中不断激发同学们思考，发表观点。最终的数据挖掘课程作业，也锻炼了我们的“自主学习-应用实践”能力，体会到方法探索过程的不易，通过亲身实践，我们也破除了对于机器学习、数据挖掘的神秘感，并切实体会到它应用的可能性。

一学期的课程调动起我对于机器学习、数据挖掘等内容的兴趣，作为如今交通行业的热门发展方向，在未来的学习过程之中，我还会学习更多的内容，并注重实践，利用这些方法切实解决一些交通问题。

附 录

神经网络尝试一代码：

```
###神经网络输入：每 SCALE 长度为一组，经计算一组内的[速度平均值、速度最大值、速度标准差]整合为一条输入数据
###神经网络输出：预测该数据点为 4 种交通方式的概率，如[0.9,0.02,0.03,0.05]
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter


#  用于将多维 list 整合成为 1 维 list
def flat(l): #  用于将多维 list 整合成为 1 维 list
    for k in l:
        if not isinstance(k, (list, tuple)):
            yield k
        else:
            yield from flat(k)


SCALE = 60#定义 SCALE 长度，每个组长度为 SCALE，之后将该组数据整合为 1 条输入数据
data_x=[]
data_y=[]


##定义文件读入函数
def choose_file(filenumber):
    traindata = pd.read_excel('train%s.xlsx'%filenumber)
    tempspeed=[]
    tempmeans=[]
    tempspeed.append(list(traindata['speed']))
    tempmeans.append(list(traindata['means']))
    tempspeed = list(flat(tempspeed))
    tempmeans = list(flat(tempmeans))
    groupnumber = len(tempspeed)//SCALE
```

```
    for i in range(groupnumber):
        speed = np.array(tempspeed[i*SCALE : (i+1)*SCALE])
        means = np.array(tempmeans[i*SCALE : (i+1)*SCALE])
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        data_x.append(list([mean_speed,max_speed,std_speed]))
        nn=0
        max_item=0
        for j in range(1,5):
            if Counter(means)[j]>nn:
                nn=Counter(means)[j]
                max_item=j
        data_y.append(list([max_item]))
    return data_x,data_y#返回用于训练模型的 x 和 y


##读入 6 个出行个体的训练文件
for i in range(1,7):
    choose_file(i)


# 将输出结果转化为 one-hot 的形式
data_y = list(flat(data_y))
for i in range(len(data_y)):
    if data_y[i] == 1:
        data_y[i] = [1,0,0,0]
    elif data_y[i] == 2:
        data_y[i] = [0,1,0,0]
    elif data_y[i] == 3:
        data_y[i] = [0,0,1,0]
    else:
        data_y[i] = [0,0,0,1]

##定义部分神经网络参数
X_IN_SIZE=3
N_CLASS = 4
LR=0.001

##定义神经网络的结构
```

```
def init_weights(shape):
    return tf.Variable(tf.random_normal(shape))
def init_biases(shape):
    return tf.Variable(tf.zeros(shape)+0.1)
def model(X, w_layer_1, w_layer_2, w_layer_3,b_layer_1,b_layer_2,b_layer_3):
    hidden_1 = tf.nn.softmax((tf.matmul(X, w_layer_1))+b_layer_1)
    hidden_2 = tf.nn.softmax((tf.matmul(hidden_1, w_layer_2))+b_layer_2)
    return tf.nn.softmax((tf.matmul(hidden_2, w_layer_3))+b_layer_3)


##定义 Placeholder
X = tf.placeholder(tf.float32, [None, X_IN_SIZE])
Y = tf.placeholder(tf.float32, [None, N_CLASS])

##建立神经网络
LAYER_UNIT=10
w_layer_1 = init_weights([X_IN_SIZE,LAYER_UNIT])
w_layer_2 = init_weights([LAYER_UNIT, LAYER_UNIT])
w_layer_3 = init_weights([LAYER_UNIT, N_CLASS])
b_layer_1 = init_biases([1,LAYER_UNIT])
b_layer_2 = init_biases([1,LAYER_UNIT])
b_layer_3 = init_biases([1,N_CLASS])



##神经网络训练后的输出
y = model(X, w_layer_1, w_layer_2, w_layer_3,b_layer_1,b_layer_2,b_layer_3)
##定义损失函数（交叉熵）以及梯度下降优化器
cross_entropy = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(y)))##tf.sum 没 有 加
reduction_indices 所以输出的 loss 与方法二有较大差别
train_step = tf.train.GradientDescentOptimizer(LR).minimize(cross_entropy)


##运行计算图
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction , "float"))
```

```
for i in range(10000):
    _,loss_=sess.run([train_step,cross_entropy], feed_dict={X: data_x, Y: data_y})
    if i % 100== 0:
        print ('trainstep','%s'%i,'loss=',loss_,'|accuracy=',sess.run(accuracy, feed_dict={X: data_x, Y: data_y}))
```

神经网络尝试二代码：

```
###神经网络输入：便利所有数据点，该数据点前后个 SCALE 长度内的[速度平均值、速度最大值、速度标准差]
###神经网络输出：预测该数据点为 4 种交通方式的概率，如[0.9,0.02,0.03,0.05]
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

##定义用于将多维 list 整合成为 1 维 list 的函数
def flat(l):
    for k in l:
        if not isinstance(k, (list, tuple)):
            yield k
        else:
            yield from flat(k)

SCALE = 60#定义 SCALE 长度，滚动每段数据实际为 2SCALE 长度
data_x=[]
data_y=[]

##定义文件读入函数
def choose_file(filenumber):
    traindata = pd.read_excel('train%s.xlsx'%filenumber)
    tempspeed=[]
    tempmeans=[]
    tempspeed.append(list(traindata['speed']))
    tempmeans.append(list(traindata['means']))
    tempspeed = list(flat(tempspeed))
    tempmeans = list(flat(tempmeans))
    for i in range(SCALE):#每个文件前 SCALE 个数据的读入
        speed=np.array(tempspeed[0 : i+SCALE])
```

```
            mean_speed=speed.mean()
            max_speed=speed.max()
            std_speed=speed.std()
            data_x.append(list([mean_speed,max_speed,std_speed]))
            data_y.append(list([tempmeans[i]]))
        for i in range(SCALE,(len(tempspeed)-SCALE)):#每个文件可以正常选择 SCALE 的数据的读入
            speed = np.array(tempspeed[i-SCALE : i+SCALE])
            mean_speed=speed.mean()
            max_speed=speed.max()
            std_speed=speed.std()
            data_x.append(list([mean_speed,max_speed,std_speed]))
            data_y.append(list([tempmeans[i]]))
        for i in range((len(tempspeed)-SCALE),len(tempspeed)):#每个文件尾部的 SCALE 个数据的读入
            speed=np.array(tempspeed[i-SCALE:(len(tempspeed))])
            mean_speed=speed.mean()
            max_speed=speed.max()
            std_speed=speed.std()
            data_x.append(list([mean_speed,max_speed,std_speed]))
            data_y.append(list([tempmeans[i]]))
        return data_x,data_y#返回用于训练模型的 x 和 y

##读入 6 个出行个体的训练文件
for i in range(1,7):
    choose_file(i)


##将输出结果转化为 one-hot 的形式
data_y = list(flat(data_y))
for i in range(len(data_y)):
    if data_y[i] == 1:
        data_y[i] = [1,0,0,0]
    elif data_y[i] == 2:
        data_y[i] = [0,1,0,0]
    elif data_y[i] == 3:
        data_y[i] = [0,0,1,0]
    else:
        data_y[i] = [0,0,0,1]
```

```
##定义部分神经网络参数
X_IN_SIZE=3
N_CLASS = 4
LR=0.01


##定义添加层函数
def add_layer(input_data,in_size,out_size,activation_function=None):
    Weight=tf.Variable(tf.random_normal([in_size,out_size]))
    biases=tf.Variable(tf.zeros([1,out_size])+0.1)
    wx_b=tf.matmul(input_data,Weight)+biases
    if activation_function==None:
        outputs=wx_b
    else:
        outputs=activation_function(wx_b)
    return outputs


##定义 Placeholder
X = tf.placeholder(tf.float32, [None, X_IN_SIZE])
Y = tf.placeholder(tf.float32, [None, N_CLASS])


##构建神经网络计算图结构 4 层神经层，每层 10 个神经元
LAYER_UNIT=10
hidden_layer1=add_layer(X,X_IN_SIZE,LAYER_UNIT,activation_function=tf.nn.softmax)
hidden_layer2=add_layer(hidden_layer1,LAYER_UNIT,LAYER_UNIT,activation_function=t
f.nn.softmax)
hidden_layer3=add_layer(hidden_layer2,LAYER_UNIT,LAYER_UNIT,activation_function=t
f.nn.softmax)
y=add_layer(hidden_layer3,LAYER_UNIT,N_CLASS,activation_function=tf.nn.softmax)


##定义损失函数（交叉熵）和梯度下降优化器：不知道这样定义是否正确，不知道
reduction_indices=[1]是否修改、有何作用。
#cross_entropy = -tf.reduce_mean(Y * tf.log(tf.clip_by_value(y, 1e-10, 1.0)))
#cross_entropy =-tf.reduce_mean(tf.reduce_sum(Y * tf.log(tf.clip_by_value(y, 1e-10, 1.0))))
#cross_entropy =-tf.reduce_mean(tf.reduce_sum(Y * tf.log(y)))
```

```
#cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=y,labels=Y)#这种方式训练后的 loss 每个 step 会有很多个数据
cross_entropy = tf.reduce_mean(-tf.reduce_sum(Y* tf.log(y),reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(LR).minimize(cross_entropy)


##保存与运行计算图
saver = tf.train.Saver()
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
save_path = saver.save(sess, "my_net/save_net.ckpt")
print("Save to path: ", save_path)
##定义准确率
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction , "float"))
for i in range(25000):
    _,loss_=sess.run([train_step,cross_entropy], feed_dict={X: data_x, Y: data_y})
    accuracy_=sess.run(accuracy, feed_dict={X: data_x, Y: data_y})
    if i % 100== 0:
        print ('trainstep','%s'%i,'loss=',loss_,'|accuracy=',accuracy_)
print(sess.run(y,feed_dict={X: data_x}))
```

支持向量机代码：

```
#-*- coding=utf-8 -*-
##一次平滑用于预测
from sklearn.svm import SVC
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter


##定义用于将多维 list 整合成为 1 维 list 的函数
def flat(l):
    for k in l:
        if not isinstance(k, (list, tuple)):
            yield k
        else:
```

```
            yield from flat(k)


SCALE = 60#定义 SCALE 长度，滚动每段数据实际为 2SCALE 长度
data_x=[]
data_y=[]#训练集

Data_x=[]
Data_y=[]#测试集
##定义文件读入函数
def choose_file(filenumber):
    traindata = pd.read_excel('train%s.xlsx'%filenumber)
    tempspeed=[]
    tempfirstplant=[]
    tempmeans=[]
    tempspeed.append(list(traindata['speed']))
    tempmeans.append(list(traindata['means']))
    tempfirstplant.append(list(traindata['firstplant']))
    tempspeed = list(flat(tempspeed))
    tempmeans = list(flat(tempmeans))
    tempfirstplant=list(flat(tempfirstplant))
    for i in range(SCALE):#每个文件前 SCALE 个数据的读入
        speed=np.array(tempspeed[0 : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
        data_y.append(list([tempmeans[i]]))
    for i in range(SCALE,(len(tempspeed)-SCALE)):#每个文件可以正常选择 SCALE 的
数据的读入
        speed = np.array(tempspeed[i-SCALE : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
```

```
            data_y.append(list([tempmeans[i]]))
        for i in range((len(tempspeed)-SCALE),len(tempspeed)):#每个文件尾部的 SCALE 个数据的读入
            speed=np.array(tempspeed[i-SCALE:(len(tempspeed))])
            speed_in=tempspeed[i]
            firstplant=tempfirstplant[i]
            mean_speed=speed.mean()
            max_speed=speed.max()
            std_speed=speed.std()
            data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
            data_y.append(list([tempmeans[i]]))
        return data_x,data_y#返回用于训练模型的 x 和 y

##读入 5 个出行个体的训练文件
l=[1,2,3,4,5,6]
for i in l:
    choose_file(i)
data_y=np.array(data_y)

import  random
a=range((len(data_x)))
b=random.sample(a,int(0.8*len(data_x)))
c=list(set(a).difference(set(b)))
data_xnew=[]
data_ynew=[]
Data_xnew=[]
Data_ynew=[]
for i in b:
            data_xnew.append(data_x[i])
            data_ynew.append(data_y[i])
for i in c:
            Data_xnew.append(data_x[i])
            Data_ynew.append(data_y[i])
data_ynew=np.array(data_ynew)

clf = SVC()
clf.fit(data_xnew, data_ynew.ravel())
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
```

```
        tol=0.001, verbose=False)
y=clf.predict(Data_xnew)
count=0
for i in range (len(y)):
        if y[i]==Data_ynew[i]:
                count+=1
accuracy=count/(len(y))
print('全局准确率',accuracy)

count1=0
count2=0
count3=0
count4=0

bingo1=0
bingo2=0
bingo3=0
bingo4=0
for i in range (len(Data_ynew)):
        if    Data_ynew[i]==1:
                count1+=1
                if y[i]==Data_ynew[i]:
                        bingo1+=1
        elif Data_ynew[i]==2:
                count2+=1
                if y[i]==Data_ynew[i]:
                        bingo2+=1
        elif Data_ynew[i]==3:
                count3+=1
                if y[i]==Data_ynew[i]:
                        bingo3+=1
        elif Data_ynew[i]==4:
                count4+=1
                if y[i]==Data_ynew[i]:
                        bingo4+=1
        else:
                pass
accuracy1=bingo1/count1
accuracy2=bingo2/count2
accuracy3=bingo3/count3
```

```
accuracy4=bingo4/count4
print("步行准确率：",accuracy1)
print("自行车准确率：",accuracy2)
print("公交车准确率：",accuracy3)
print("小汽车准确率：",accuracy4)

def predict(filenumber):
    test_x=[]
    testdata =pd.read_excel("test(%s).xlsx"%(filenumber))
    tempspeed=[]
    tempfirstplant=[]
    tempspeed.append(list(testdata['speed']))
    tempfirstplant.append(list(testdata['firstplant']))
    tempspeed = list(flat(tempspeed))
    tempfirstplant=list(flat(tempfirstplant))
    for i in range(SCALE):#每个文件前 SCALE 个数据的读入
        speed=np.array(tempspeed[0 : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        test_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
    for i in range(SCALE,(len(tempspeed)-SCALE)):#每个文件可以正常选择 SCALE 的数据的读入
        speed = np.array(tempspeed[i-SCALE : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        test_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
    for i in range((len(tempspeed)-SCALE),len(tempspeed)):#每个文件尾部的 SCALE 个数据的读入
        speed=np.array(tempspeed[i-SCALE:(len(tempspeed))])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
```

```
            std_speed=speed.std()
            test_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
        predictmeans=clf.predict(test_x)
        testdata['predictmeans']=predictmeans
        testdata.to_excel('testresult%s.xlsx'%filenumber)

for i in range(1,16):
        predict(i)
```