应用机器学习方法
建立网络侧变量与终端用户视频体验的模型

2017 年 4 月 29 日

陈旭鹏　2014012882

王复英　2015011175

马聿伯　2015012295

摘要

由于题目本身已经规定了变量，并给出了较多的数据，因此我们不需要再做具体问题的抽象，而可以将研究重点放在寻找最优的函数模型上。为了研究三个网络侧变量和两个用户体验评价变量之间的函数关系，我们总共尝试了三大类模型：广义线性回归模型、SVR 以及神经网络模型。运用各种模型，我们分别寻找了网络侧变量与用户体验变量之间的关系，并设置了 MAE 与r^2两个评价指标来评估模型的精度。在上述方法中，只有广义线性模型可以得到显式的函数关系式，但是精度并不令人满意。在学习并使用了 SVR 与神经网络两类较为强大的机器学习与深度模型后，我们得到了较好的 MAE 与r^2。尽管这两种方法这不能写出显式的函数表达式，但却可以满足“输入网络侧自变量，预测用户体验因变量”的要求。在进一步的改进中，我们使用了交叉验证法来极为严格地评估我们的模型的好坏，从而避免了认为选取训练集与预测集的主观性。在这一过程中，我们利用 python、MATLAB 、sklearn 库、Keras、matplotlib 等编程与计算软件实现了模型的建立、求解与图像绘制工作。

我们的尝试是递进式的，从最基本的广义线性模型，到用 MATLAB 实现效果更好但计算效率很低的 SVR，到使用 sklearn 的 SVR 更快速地训练及调参以获得更好的模型，到使用强大的深度学习方法获得更好的 MAE。我们通过此次课题学到了大量的机器学习的理论知识与工具运用方法。在文章的最后我们也提到了很多可以改进的地方，并且提出了一些可行的改进方法，以后有时间还可以进一步优化我们的结果。

关键词：**机器学习，广义线性模型，支持向量回归，深度学习，python，MATLAB**

目录

一.问题背景与问题分析

1.1 问题背景

随着无线宽带网络的升级，以及智能终端的普及，越来越多的用户选择在移动智能终端上应用客户端 APP 观看网络视频，这是一种基于 TCP 的视频传输及播放。看网络视频影响用户体验的两个关键指标是初始缓冲等待时间和在视频播放过程中的卡顿缓冲时间，我们可以用初始缓冲时延和卡顿时长占比（卡顿时长占比 = 卡顿时长 / 视频播放时长）来定量评价用户体验。研究表明影响初始缓冲时延和卡顿时长占比的主要因素有初始缓冲峰值速率、播放阶段平均下载速率、端到端环回时间（E2E RTT），以及视频参数。然而这些因素和初始缓冲时延和卡顿时长占比之间的关系并不明确。我们需要研究用户体验评价变量（初始缓冲时延，卡顿时长占比）与网络侧变量（初始缓冲峰值速率，播放阶段平均下载速率，E2E RTT)之间的函数关系。

1.2 问题分析

由于本题直接给出了各种参量的明确含义，建模要求简洁明了（即通过这些数据建立用户体验评价变量与网络侧变量的函数关系），因此我们认为本题已经不需要具体情境的数学抽象，而可以直接从求解方法的角度上进行思考。

考虑到本题海量的数据以及这些数据之间纷繁复杂的关系，利用传统方法进行回归分析无论是效率还是精度上都无法满足我们的要求。因此我们尝试通过机器学习的方法来完成本题的求解。我们搜集了大量与机器学习有关的理论知识与算法样例，建立了不同的模型来寻找网络侧变量与用户体验评价变量之间的关系。在下文中，我们将通过不同模型对新的未知样本进行预测，并对这些模型从效率与准确度等多个方面进行评估、比较，以期进行改进。

二.问题分析

1.数据处理

题中附件的数据多种多样,但是根据题意,我们只需分析三个网络侧变量(初始缓冲峰值速率、播放平均下载速率、E2E RTT）和两个用户体验评价变量（初始缓冲时延、卡顿时长占比）之间的函数关系即可。剩下的若干数据仅作为参考性指标，在分析函数关系中将被我们有目的的忽略，这样可以减少工作量，将有限的建模时间与资源更集中地投入到本题的核心之中。

2. 数据导入

后面的模型求解中，我们需要想将数据导入 Python 中（或 matlab）中才能用它进行计算，下面先给出 Python 导入数据的过程，在后面的模型求解中，我们默认数据已经导入。

```
import sklearn                #导入 sklearn 机器学习库
import scipy                  #导入 scipy 科学计算库
import scipy.io as scio
dataFile = '×××/data.mat'
data = scio.loadmat(dataFile)
dataset = data["Data"]
# X，y 为训练集；x_val，y_val 为测试集
X = dataset[0:10000, 1:4]          #第 1-10000 行，2-4 列
y = dataset[0:10000, 4:5]          #第 1-10000 行，第 5 列
# y = dataset[0:10000, 5:6]        计算第二个因变量的函数关系第，第 6 列
x_val = dataset[10000:11000, 1:4]
```

```
y_val = dataset[10000:11000, 4:5]
# y = dataset[0:10000, 5:6]        计算第二个因变量的函数关系第，第 6 列
```

2.评价指标

这里我们首先引入评价我们模型精确度的指标：MAE 和r^2：

(1) .**MAE （Mean Absolute Error,平均绝对误差）**

平均绝对误差是所有单个观测值与算术平均值的偏差的绝对值的平均。与平均误差相比，平均绝对误差由于离差被绝对值化，不会出现正负相抵消的情况，因而，平均绝对误差能更好地反映预测值误差的实际情况。

$$\mathbf{MAE} = \frac{\sum_{i=1}^{n} |f(x_i) - y_i|}{n}$$

(2).r^2 (coefficient,相关系数)

相关系数是反映变量之间线性关系密切程度的统计指标。我们想要描述预测结果和真实结果的关系，在理想情况下（预测全部完全正确），二者满足 y=x 的函数关系。我们要计算预测值与真实值的r^2，（当预测值与真实值完全吻合时相关系数为 1），来描述测试结果与真实结果的相关程度。

这里也给出 Python 计算这两个评价指标的方法：

```
import sklearn
#分别导入计算 MAE 和r^2的模块
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
#y_val 表示测试集的真实值，y_pre 表示通过模型预测出来的结果
mae = mean_absolute_error(y_val,y_pre)    #计算平均绝对误差
```

```
r2 = r2_score(y_val,y_pre)                    #计算相关系数
```

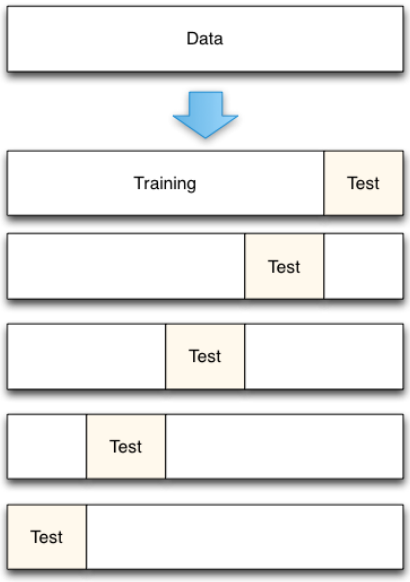
另外，为了更直观的显示我们的预测数据与真实数据的差别，对每个模型的计算结果，我们都会做一幅图来描述预测结果和真实结果的关系，在理想情况下（预测全部完全正确），该图为 y=x 的图像，但是实际情况下总会存在一定的偏差。下面给出 python 中绘图的方法：

```
import matplotlib.pyplot as plt        #导入绘图工具库
from matplotlib.lines import Line2D
figure, ax = plt.subplots()
ax.set_xlim(left=0, right=1500)          #设置横坐标范围
# ax.set_xlim(left=0, right=1)
ax.set_ylim(bottom=0, top=1500)       #设置纵坐标范围
#ax.set_ylim(bottom=0, top=1)
plt.xlabel(u"true_value")
plt.ylabel(u"predict_value")
ax.annotate('r^2=×××', xy = (200,1200), xytext=(200, 1200))
ax.annotate('mae=×××', xy = (200,1000), xytext=(200, 1000))
line1 = [(0, 0), (1500, 1500)]
(line1_xs, line1_ys) = zip(*line1)
ax.add_line(Line2D(line1_xs, line1_ys, linewidth=1, color='blue'))
p1 = plt.scatter(y_val,y_pre, marker = 'o', color = 'm')
plt.plot()
plt.show()
```

4.交叉验证（cross validation）

我们在进行人为的规定训练样本与测试样本进行各模型的回归之后，又了解了交叉验证的方法。交叉验证的方法保证了我们的取样是在完整集合中均匀抽取的，因此可以更严格准确地评价模型的结果。

我们使用的是 K-折交叉验证（K-CV），将原始数据分成 K 组（一般是均分），将每个子集数据分别做一次验证集，其余的 K-1 组子集数据作为训练集，这样会得到 K 个模型，用这 K 个模型最终的验证集的分类准确率的平均数作为此 K-CV 下分类器的性能指标。K-CV 可以有效的避免过学习以及欠学习状态的发生，最后得到的结果也比较具有说服性。我们选择 K=10，即把总样本分为十份进行测试。



我们在每个模型的训练与测试中会先使用前 10000 行作为训练集，第 10001 至 11000 行作为测试集，计算 MAE 与r^2，然后再进行交叉验证，并在交叉验证的严格标准下计算 MAE，此时的 MAE 应该是最为公正客观的。

Python 中实现交叉验证的方法如下：

```
import sklearn

from sklearn import cross_validation

scores=cross_validation.cross_val_score(clf,X,y,cv=10,scoring='mean_absolute_

error')                #计算交叉验证得到的平均绝对误差

mae_scores = scores.mean()

#mae_scores = -scores.mean()        取绝对值
```

三.符号系统

编号	符号	说明	备注
1	MAE	绝对平均误差	经过训练的模型预测的目标值与真实值之间的差的绝对值的平均
2	r^2	相关系数	反映变量之间线性关系密切程度的统计指标
3	y	目标值	
4	$\hat{y}$	预测值	通过回归分析计算得出的值
5	w	系数向量	$w=(w_0,w_1,\ldots,w_p)$，SVR 之中恰可表示超平面的法向量
6	X	样本矩阵	矩阵的每一行表示一个样本值，每一列表示某个自变量
7	D	训练样本集	
8	r	距离	样本空间中任意点到超平面的距离
9	γ	间隔	两个异类支持向量到超平面的距离之和
10	ε	偏差上限	支持向量回归假设我们能容忍 f(x)与 y 之间最多有ε的偏差
11	C	正则化常数	
12	L_p	正则化范数	
13	ε_i	松弛变量	
14	$\mu_i,\hat{\mu_i},\alpha_i,\hat{\alpha_i}$	拉格朗日乘子	$\mu_i\geq 0,\hat{\mu_i}\geq 0,\alpha_i\geq 0,\hat{\alpha_i}\geq 0$
15	z_i	映射函数	
16	$K(x_1,x_2)$	核函数	
17	$\phi(x_i)$	变换函数	

四. 广义线性模型下的回归分析

我们首先想到尝试用一些比较基础的回归分析模型对数据拟合，探索变量间的函数关系。在广义线性模型下的回归分析中，目标值 y 是输入变量 x 的线性组合。

$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$，其中$\hat{y}$是预测值。

在这个模型中,我们定义向量$w = (w_0, w_1, \ldots, w_p)$为系数向量。我们将在下文中尝试运用若干广义线性模型进行计算与求解。

4.1.普通最小二乘法

【模型原理】

首先我们尝试使用最基本的最小二乘法模型进行求解。最小二乘法的数学形式可表达为: $\min_w ||Xw - y||_2{}^2$。矩阵 X 包含了样本中这三个自变量的有关信息，用系数 :$w = (w_1, \ldots, w_p)$来拟合一个线性模型，使得数据集实际观测数据和预测数据（估计值）之间距离平方和最小。

【模型求解】

我们使用 Python 语言来求解，通过调用 fit 函数来拟合 X,y(X 为输入，y 为输出).并且会把拟合的线性模型的系数向量 w 存储到成员变量中。

```
#最小二乘法
import sklearn
from sklearn import linear_model
#建立线性回归模型
clf = linear_model.LinearRegression()
clf.fit (X, y)                    #用 X，y 样本训练模型
```

clf.coef_	#计算本模型参数向量
clf.intercept_	#计算常数项
y_pre = clf.predict(x_val)	#利用训练好的模型对第 10001-11000 行做预测

最终我们得到的函数关系为：

a.网络侧变量与初始缓冲时延之间的关系：

$$y = 5094.9 - 0.0194x_1 + 19.48x_2 - 0.678x_3$$

b.网络侧变量与卡顿占比之间的关系：

$$y = 1.69 - 3.95 \times 10^{-7}x_1 + 1.03 \times 10^{-3}x_2 - 2.9 \times 10^{-4}x_3$$

【模型评价】

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE：322.232280491

r^2：0.578651388042

Cross validation MAE：431.163320024

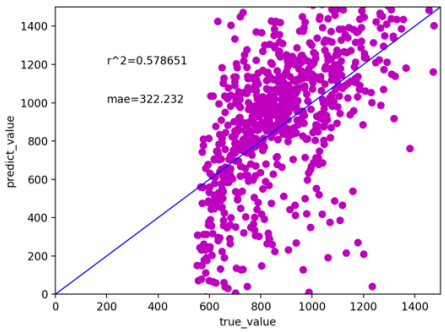
b.考察网络侧变量与卡顿占比之间的关系：

MAE：0.0931312468332

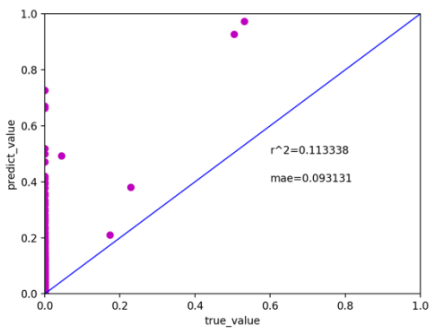
r^2：0.113337856494

Cross validation MAE：0.126683570118

图形如下：



网络侧变量与初始缓冲时延



网络侧变量与卡顿占比

2. LASSO 回归

【模型原理】

由于最小二乘法得到的结果与我们的预期存在一定的差距，因此我们尝试利用其他的广义线性回归模型进行分析。LASSO 回归通过构造一个一阶惩罚函数获得一个精炼的模型，通过最终确定一些变量的系数为 0，解释力很强。

【模型求解】

我们使用 Python 语言来求解，通过调用 fit 函数来拟合 X,y(X 为输入，y 为输出).并且会把拟合的线性模型的系数向量 w 存储到成员变量中。

```
#LASSO 回归
import sklearn
from sklearn import linear_model
clf = linear_model.Lasso(alpha = 0.1)
clf.fit (X, y)                    #用 X，y 样本训练模型
clf.coef_                         #计算本模型参数向量
clf.intercept_                    #计算常数项
y_pre = clf.predict(x_val)     #利用训练好的模型对第 10001-11000 行做预测
```

最终我们得到的函数关系为：

a.网络侧变量与初始缓冲时延之间的关系：

$$y = 5094.9 - 0.0194x_1 + 19.48x_2 - 0.678x_3$$

b.网络侧变量与卡顿占比之间的关系：

$$y = 1.70 - 4.45 \times 10^{-7}x_1 + 7.26 \times 10^{-4}x_2 - 2.93 \times 10^{-4}x_3$$

【模型评价】

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE：322.232280491

r^2：0.5786508096

Cross validation MAE：431.04454179

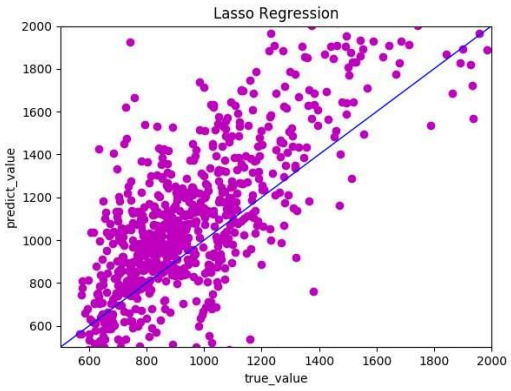
b.考察网络侧变量与卡顿占比之间的关系：

MAE：0.0934592022871

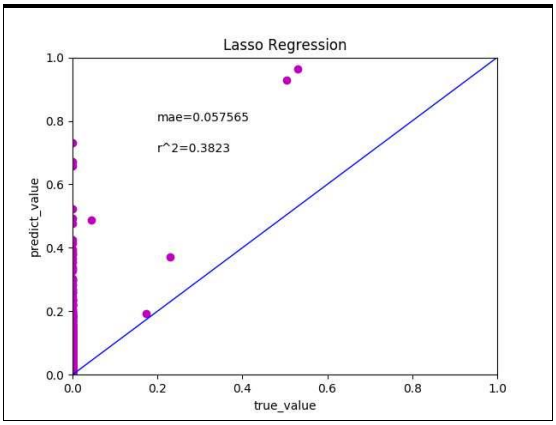
r^2：0.10813569654981559

Cross validation MAE：0.126683570118

图形如下：



网络侧变量与初始缓冲时延



网络侧变量与卡顿占比

3.岭回归

【模型原理】

岭回归是一种专用于共线性数据分析的有偏估计回归方法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、

更可靠的回归方法，对病态数据的拟合要强于最小二乘法。

【模型求解】

我们使用 Python 语言来求解，通过调用 fit 函数来拟合 X,y(X 为输入，y 为输出).并且会把拟合的线性模型的系数向量 w 存储到成员变量中。

```
#岭回归
import sklearn
from sklearn import linear_model
clf = linear_model.Ridge(alpha = 0.5)
clf.fit (X, y)                    #用 X，y 样本训练模型
clf.coef_                         #计算本模型参数向量
clf.intercept_                    #计算常数项
y_pre = clf.predict(x_val)     #利用训练好的模型对第 10001-11000 行做预测
```

最终我们得到的函数关系为：

a.网络侧变量与初始缓冲时延之间的关系：

$$y = 5094.9 - 0.0194x_1 + 19.48x_2 - 0.678x_3$$

b.网络侧变量与卡顿占比之间的关系：

$$y = 1.69 - 3.95 \times 10^{-7}x_1 + 1.03 \times 10^{-3}x_2 - 2.94 \times 10^{-4}x_3$$

【模型评价】

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE：322.232287203

r^2：0.578651382276125

Cross validation MAE：431.04454179

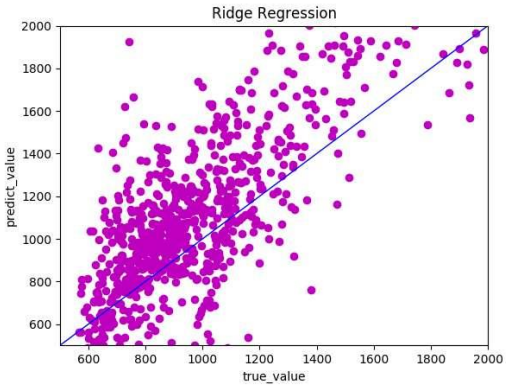
b.考察网络侧变量与卡顿占比之间的关系：

MAE：0.0931312469872

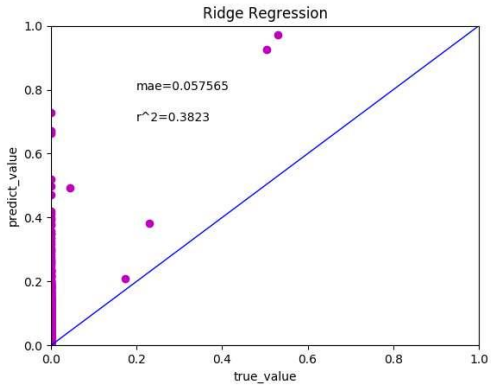
r^2：0.1133378541227859

Cross validation MAE：0.126683570118

图形如下：



网络侧变量与初始缓冲时延



网络侧变量与卡顿占比

五.SVR 模型下的回归分析

在上文中我们使用了数个广义线性模型进行回归分析，尽管方法简便、求解迅速并且可以得到确切的函数关系表达式，但 MAE 与r^2的具体数值告诉我们以上模型的近似效果均差强人意。为此我们考虑利用新的手段对模型进行求解。为了在提高精确度的同时防止过拟合情形的发生，我们摒弃了“函数关系即函数关系式”的固有观念，期冀于通过 SVR（support vector regression,支持向量回归）方法拟合这种一一对应的映射关系。尽管该方法不能给出我们确切的函数表达式，但通过以下的分析计算可以看出，它的拟合效果要显著优于广义线性回归模型。SVR 方法是本次建模的核心，因此有必要将它的基本思想做一个试探性的阐述.

5.1 模型原理说明

支持向量机(Support Vector Machine)(SVM)是 Cortes 和 Vapnik 于 1995 年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。

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，Accuracy）和学习能力（即无错误地识别任意样本的能力）之间寻求最佳折衷，以期获得最好的泛化能力。在本题中，我们注意到题目的数据样本数量足够大，但是样本特征数有限，根据我们的建模需要，我们只需要寻找第二至四列的变量与第五列或者与第六列变量的函数关系。SVM 将会通过核函数“增加变量”，并且通过惩罚变量（松弛变量）很好的解决过拟合问题，从而得到令我们满意的模型。SVM 既可以解决分类问题，又可以解决回归问题（SVR）。

5.2 基本概念介绍与模型阐述

5.2.1.间隔与支持向量

给定训练样本集 D={$(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)$},$y_i \in \{-1, 1\}$，分类问题最基本的想法就是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。在样本空间中，划分超平面可通过如下线性方程来描述：

$$w^T x + b = 0$$

其中 w=$(w_1; w_2; \ldots; w_d)$为法向量，决定了超平面的方向；b 为位移项，决定了超平面和原点的距离。显然，划分超平面可以被法向量 w 和位移 b 确定，下面我们将其记为（w, b）。样本空间中任意点 x 到超平面(w,b)的距离可写为：

$$r = \frac{|w^T x + b|}{||w||}$$

假设超平面（w, b）能将样本正确分类，即对于（x_i, y_i）ϵD，有

$$w^T x_i + b \geq +1, y_i = +1;$$
$$w^T x_i + b \leq -1, y_i = -1.$$

距离超平面最近的这几个训练样本点使上式中的等号成立，它们被称为“支持向量”，两个异类支持向量到超平面的距离之和为：

$$\gamma = \frac{2}{||w||},$$

它被称为“间隔”。

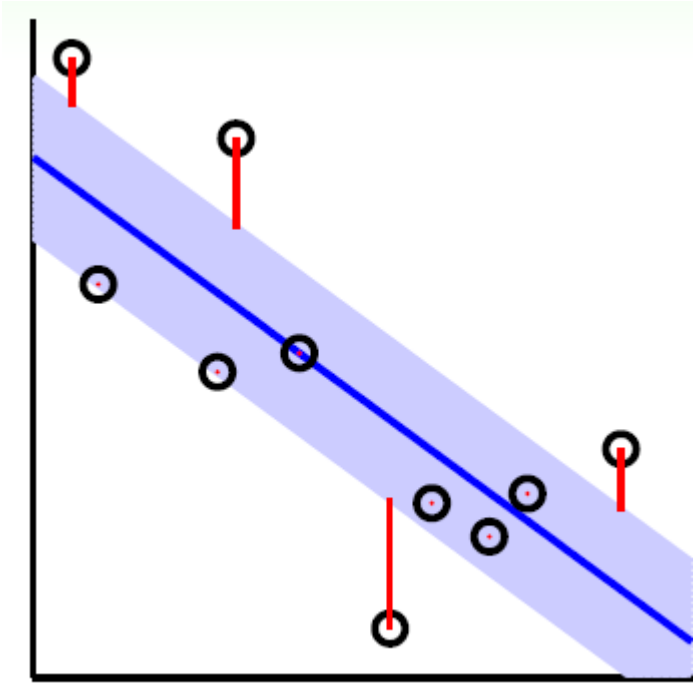
5.2.2 支持向量回归简介

我们试图寻找一个线性回归方程（函数 y=g(x)）去拟合所有的样本点，它寻求的最优超平面总方差最小。给定训练样本：D={$(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)$},$y_i \in$ R,希望得到得到一形如 f(x)=$w^T x + b$的回归模型，使得 f(x)与 y 尽可能接近，w 和 b 是待确定的模型参数。

支持向量回归也分为线性回归和非线性回归两种，但不是统计学中的线性或者非线性回归了，而是根据是否需要嵌入到高维空间来划分的。

5.2.3 支持向量回归（SVR）的损失函数度量

对样本（x,y），传统回归模型通常基于模型直接输出 f(x)与真实输出 y 之间的差别来计算损失，支持向量回归假设我们能容忍 f(x)与 y 之间最多有ε的偏差，即仅当 f(x)与 y 之间的差别绝对值大于ε时才计算损失。



支持向量回归示意图.蓝色显示出ε-间隔带，落入其中的样本不计算损失.

如上图所示，相当于以 f(x)为中心，建立了一个宽度为 2ε的间隔带，若训练样本落入此间隔带内，则被认为是预测正确的。

$$\mathrm{err}(x_i, y_i) = \begin{cases} 0 \ , |y_i - w^T x - b| \le \epsilon \\ |y_i - w^T x - b| - \varepsilon, |y_i - w^T x - b| \ge \epsilon \end{cases}$$

5.2.4　SVR 问题的目标函数与约束条件

欲找到具有“最大间隔”的超平面，使得训练样本尽可能落入此间隔带内，即要找能满足约束的参数 w 和 b，使得γ最大，即：

$$\max_{w,b} \frac{2}{||w||}$$

$$\text{s.t.} \ | \ y_i - w^T x - b| \le \epsilon, \quad i = 1,2,\dots,m.$$

为了最大化间隔，仅需最大化$||w||^{-1}$，这等价于最小化$||w||^2$，于是上式等价于：

$$\min_{w,b} \frac{||w||^2}{2}$$

$$\text{s.t.} \ | \ y_i - w^T x - b| \le \epsilon, \quad i = 1,2,\dots,m.$$

5.2.5 软间隔与正则化

在实际问题中，我们发现不存在一个超平面使得回归的点都在超平面确定的间隔带以内，即使实现了这样的回归，我们也很难断定这不是过拟合。

如果我们要求所有样本点都在我们划定的间隔内，这称为“硬间隔”，而“软间隔”则是允许某些样本不满足约束：

$$|\ y_i - w^T x - b| \leq \epsilon$$

当然，在最大化间隔的同时，不满足约束的样本应该尽可能少。此时常常对模型的参数做一定的限制，使得模型偏好更简单的参数，这就叫“正则化”。

正则化可理解为一种“惩罚函数法”，即对不希望得到的结果施以惩罚，从而使得优化过程趋向于希望目标。从贝叶斯估计的角度看，正则化项可认为是提供了模型的先验概率。

我们在目标函数后面增加正则化项后可得：

$$\min_{w,b} \frac{||w||^2}{2} + C\sum_{i=1}^{m} l_\epsilon(f(x_i) - y_i)$$

其中 C>0,成为正则化常数。L_p范数是常用的正则化项，其中L_2范数$||w||_2$倾向于 w 的分量的取值尽量均衡，即非零分量的个数尽量稠密；而L_0范数$||w||_0$和L_1范数$||w||_1$则倾向于 w 的分量的取值尽量均衡，即非零分量尽可能的少。

5.2.6 目标函数的对偶形式

引入松弛变量$\varepsilon_i \geq 0$，可将上式重写为：

$$\min_{w,b,\ \varepsilon_i,\hat{\varepsilon_i}} \frac{||w||^2}{2} + C\sum_{i=1}^{m}(\varepsilon_i + \hat{\varepsilon_i})$$

$$\text{s.t. f}(x_i) - y_i \leq \epsilon + \varepsilon_i,$$

$$y_i - f(x_i) \leq \epsilon + \hat{\varepsilon_i},$$

$$\varepsilon_i \geq 0, \hat{\varepsilon_i} \geq 0, i = 1,2,\dots,m$$

引入拉格朗日乘子$\mu_i \geq 0, \hat{\mu_i} \geq 0, \alpha_i \geq 0, \hat{\alpha_i} \geq 0$,由拉格朗日乘子法得到拉格朗日函数

L(w, b, $\alpha, \hat{\alpha}, \varepsilon, \hat{\varepsilon}, \mu, \hat{\mu}$)= $\frac{||w||^2}{2}$ + $C\sum_{i=1}^{m}(\varepsilon_i + \hat{\varepsilon_i})$ - $\sum_{i=1}^{m}\mu_i\varepsilon_i$ - $\sum_{i=1}^{m}\hat{\mu_i}\hat{\varepsilon_i}$+$\sum_{i=1}^{m}\alpha_i(f(x_i) - y_i - \epsilon - \varepsilon_i)$+$\sum_{i=1}^{m}\hat{\alpha_i}(y_i - f(x_i) - \epsilon - \hat{\varepsilon_i})$

带入 f(x)=$w^T x + b$,再令 L(w, b, $\alpha, \hat{\alpha}, \varepsilon, \hat{\varepsilon}, \mu, \hat{\mu}$)对 $w, b, \varepsilon_i, \hat{\varepsilon_i}$的偏导为 0，得到 SVR 问题的对偶问题：

$$\max_{\alpha,\ \hat{\alpha}} \sum_{i=1}^{m} y_i(\hat{\alpha_i} - \alpha_i) - \epsilon(\hat{\alpha_i} + \alpha_i) - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\hat{\alpha_i} - \alpha_i)(\hat{\alpha_j} - \alpha_j)x_i^T x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{m}(\hat{\alpha_i} - \alpha_i) = 0$$

$$0 \leq \alpha_i, \hat{\alpha_i} \leq C$$

上述过程中需满足 KKT 条件，即要求：

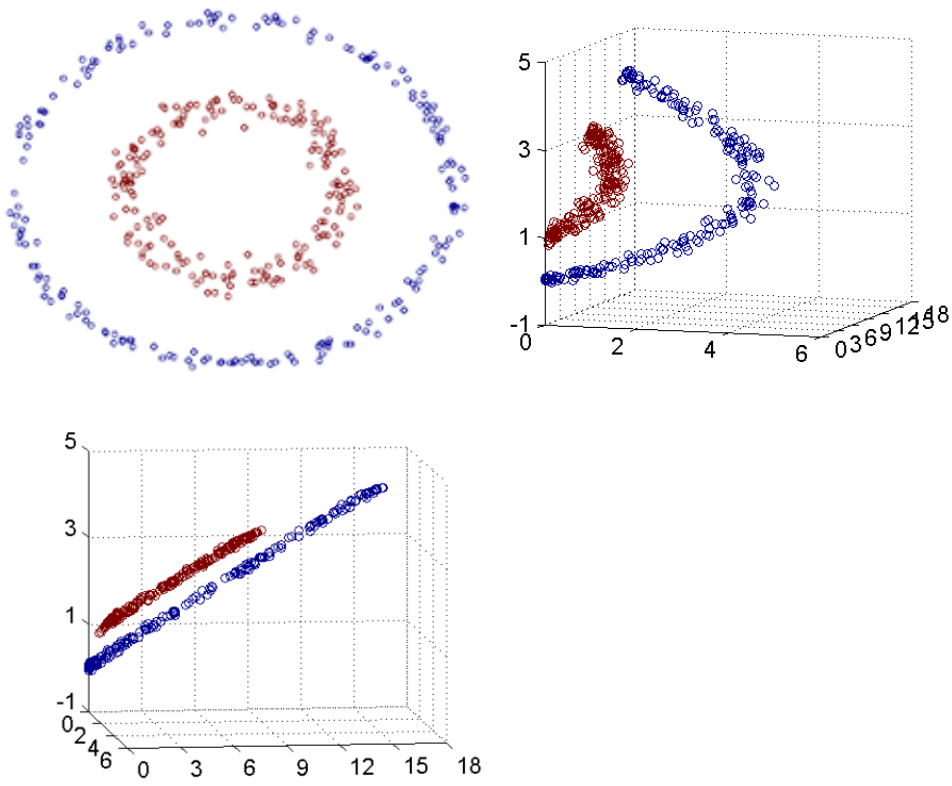
$$\begin{cases} \alpha_i(f(x_i) - y_i - \epsilon - \varepsilon_i) = 0 \\ \hat{\alpha_i}(y_i - f(x_i) - \epsilon - \hat{\varepsilon_i}) = 0 \\ \alpha_i\hat{\alpha_i} = 0, \varepsilon_i\hat{\varepsilon_i} = 0, \\ (C - \alpha_i)\varepsilon_i = 0, (C - \hat{\alpha_i})\hat{\varepsilon_i} = 0 \end{cases}$$

由上式易知，仅当样本(x_i, y_i)不落入ϵ-间隔带，相应的α_i和$\hat{\alpha_i}$才能去非零值。另外，α_i和$\hat{\alpha_i}$中至少有一个为零。

5.2.7 kernel 函数及选择

我们的目标是寻找一个超平面使得距离各个向量的欧氏距离最短。由于样本的线性不可分问题，我们可以通过升高样本向量的维度（从三维到更高维度）使得样本线性可分。

从下面的例子可以解释 SVR 的做法：



两个数据集在二维平面上线性不可分，但是当我们把这两个类似于椭圆形的点映射到一个高维空间后，映射函数为：$z_1 = x_1^2, z_2 = x_2^2, z_3 = x_2$,自此两个样本可以通过一个三维空间中的平面线性可分。

从哲学思想上说：世界上本来没有两个完全一样的物体，对于所有的两个物体，我们可以通过增加维度来让他们最终有所区别，当维度增加到无限维的时候，一定可以让任意的两个物体可分了。

我们发现在低维用线性回归解决此问题并不令人满意，因此我们希望做的就是将样本的特征向量升维，从而将一个低维的非线性问题转化为一个高维的线性问题，利用核函数可以方便地进行内积的运算，从而解决我们的回归问题，核函数可以显著地降低计算的复杂度，甚至把不可能的计算变为可能。

核函数有如下的性质：

$$K(x_1, x_2) = \phi(x_1) * \phi(x_2)$$

其中ϕ(x)是对 x 做变换的函数，有些变换会将样本映射到更高维的空间，如

果这个高维空间内x_1与x_2是线性可分的，那么我们就做了一次成功的变换。核函数是二元函数，其输出与两个向量变换之后的内积相等。

我们前面已经知道了 SVR 原始形式的对偶问题：

$$\max_{\alpha,\ \hat{\alpha}} \sum_{i=1}^{m} y_i\left(\hat{\alpha_i} - \alpha_i\right) - \epsilon(\hat{\alpha_i} + \alpha_i) - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\hat{\alpha_i} - \alpha_i)(\hat{\alpha_j} - \alpha_j)x_i^T x_j$$

很明显，未知量α的个数与样本的个数是相等的，那么这个对偶问题计算的时间复杂度是与训练样本的个数正相关的。将样本先变换到高维空间，然后再求在高维空间内的内积，这样的变换还是需要很多计算资源。由于核函数的作用是“核函数是二元函数，输入是变换之前的两个向量，其输出与两个向量变换之后的内积相等”，所以我们可以用$\kappa(x_i, x_j)$来替代：$\max_{\alpha,\ \hat{\alpha}} \sum_{i=1}^{m} y_i\left(\hat{\alpha_i} - \alpha_i\right) - \epsilon(\hat{\alpha_i} + \alpha_i) - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\hat{\alpha_i} - \alpha_i)(\hat{\alpha_j} - \alpha_j)x_i^T x_j$中的$x_i^T x_j$从而避免了显式的特征变换。使用核技巧之后，学习是隐式地在特征空间进行的，不需要显式地定义特征空间和映射函数。而$\kappa(x_i, x_j)$就是将线性空间映射到高维空间的核函数。核函数有线性核、多项式核、径向基内核 RBF（高斯核）、sigmoid 核等。

RBF 核函数可以将一个样本映射到一个无穷维空间，再计算样本向量两两的内积，能力强大，可以求解复杂的边界，符合我们的需求，而且线性核函数是 RBF 的一个特例，也就是说如果考虑使用 RBF，那么就没有必要考虑线性核函数了。RBF 与多项式核函数相比，需要确定的参数更少（只有一个参数），数值变化范围小，从而减少函数的复杂度，会减少数值的计算困难。

RBF 函数表达式：$\kappa\left(x_i, x_j\right) = \exp(-\frac{(x_i - x_j)^2}{2\sigma^2})$

在具体的回归分析时，用核生成的变量是在 reproducing kernel Hilbert space 里的点，这些点都是无穷维的 vector，即我们把回归问题等价为了把所有点都放在这个无穷维空间做 regression。而当需要预测新样本时，我们同样不需要进行

显式的运算。SVR 学习出来的权值ω为支撑向量的线性组合，即$w = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)$，所以求解$f(x) = w^T \phi(x) + b$时我们就可以使用核技巧，从而避免显式变换，即$f(x) = w^T \phi(x) + b = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b$

这样我们做到了不再使用变换ϕ显式的计算高维向量，而是使用 Kernel；还不再储存 Z 空间中的w，而是存储表示这个w的支持向量和这个线性组合的系数α_n。也就是我们用 MATLAB 做最终的计算中的x_1.

总而言之，核函数帮助我们实现了样本向量之间的分隔（通过升高维度的方式），并且可以较快地利用 kernel trick 计算高维空间中向量的内积。

5.3 模型求解

①MATLAB 实现

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE:263.0858

r^2:0.6254

b.考察网络侧变量与卡顿占比之间的关系

MAE：0.0074133

r^2：0.9244

（由于 matlab 的算法实现速度较慢，我们没有足够的时间对其进行交叉验证）

MATLAB 代码解释：以获得第 2-4 列自变量与第 5 列因变量的关系为例。我们选择的评价标准为最小均方差 MSE，即预测值与样本真实值的方差的均值。

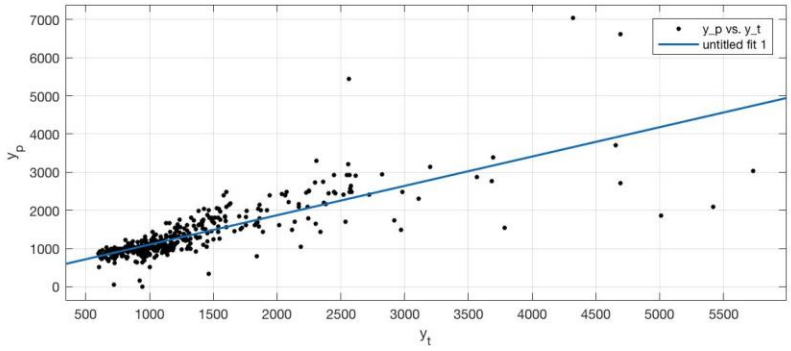
```
clc    clear all    close all
%数据导入与读取
%{
```

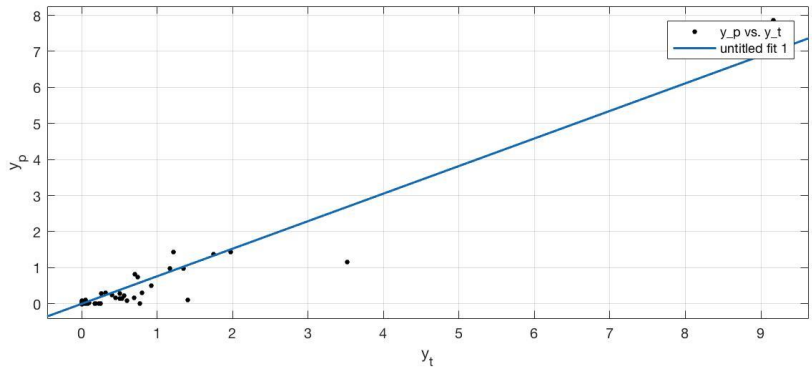
```
filename='SpeedVideoDataforModeling.xlsx';
Data = xlsread(filename);
save data.mat Data
%}
load data.mat

%这里需要使用一个SLEP_package.此包可以帮助我们实现梯度下降，计算

每次梯度下降的cost function.

addpath(genpath(pwd))
%%%%%%%%%%%%%%%%%%%%%%%%Funtion1: input 2-4, output
5%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is a continuous output

%所使用的cost function如下，其中加入了L1、L2正则化项，因为我们使用

高斯kernel计算ω，因此此时的cost function中用κ与α的线性组合表示ω。正

则化项的正则化常数为rho_1、rho_2.

% min    1/2 || K alpha - y||^2 + 1/2 * rho_1 * ||alpha||_2^2 + rho_2 *
||alpha||_1

读取前一万行样本作为训练集

y=Data(1:10000,5);
X=Data(1:10000,2:4);

12001到12400行样本作为测试集

X_t=Data(12001:12400,2:4);
y_t=Data(12001:12400,5);
%----------------------- Set optional items -----------------------
opts=[];
% Starting point

%  梯度下降的起始点可以随机选则，这里选取0做起点

opts.init=0;            % starting from a zero point
% termination criterion
opts.tFlag=5;           % run .maxIter iterations
opts.maxIter=100;

% maximum number of iterations，即一共做一百次梯度下降

%  每次iteration的cost function会被输出到funval表格中，可以看到cost

funtion是在逐步下降的。
```

```
% normalization
opts.nFlag=0;           % without normalization
% regularization
%opts.rFlag=1;           % the input parameter 'rho' is a ratio in (0, 1)
opts.rsL2=0;        % the squared two norm term
%因为只有一个变量，因此我们只需要L1正则化项，L1正则化项即可以保证
sparse，又可以防止过拟合（overfitting），符合我们的要求。
opts.mFlag=1;           % treating it as compositive function
opts.lFlag=1;          % Nemirovski's line search
%rho=0.01;                  % the regularization parameter
                           % it is a ratio between (0,1), if .rFlag=1
%----------------------training ----------------------------------------%
%通过改变i，j来改变kernel与正则化项的参数，从而寻找最优参数。
for i=1:20
    for j=1:10
        for k=1:1
            sigma=2^(i-4);
            rho=2^(j-13);
          opts.rsL2=0;
          %定义高斯核函数
          K=exp(-(dist(X,X').^2)*(sigma^-2)/2);
          %利用高斯核函数计算高维向量的两两之间的内积。因为我们输
入了一万个样本，因此计算两两之间的内积会得出一个10000*10000的对角
矩阵，该矩阵对角线元素为1，即一个向量与它本身的内积为1（因为向量与
自身的距离为0，代入高斯核函数得到值1），这个矩阵被输出为K
          %K=dist(X,X');
%调取SLEP_package中的LeastR函数计算带正则化项的损失函数
          [x1, funVal1, ValueL1]= LeastR(K, y, rho, opts);
          %令alpha(:,i)=x1，x1即为所求参数;
          %计算预测集的K，同样利用高斯核函数，计算预测集的四百个样
本向量与原训练集一万个样本向量的内积，生成400*10000的矩阵K_t
              K_t=exp(-(dist(X_t,X').^2)*(sigma^-2)/2);
```

```
            %K_t=dist(X_t,X');
            %计算预测集的y
            y_p=K_t*x1;
            %计算预测数据与真实情况的绝对值误差
            error=sqrt((y_t-y_p).^2);
            %输出20*10的MSE矩
               MSE(i,j)=mean(error);
        end end end
[min_val,min_ind]=min(min(MSE));
%alpha_best=alpha(:,min_ind);
```

在输出在输出的 20*10 的 MSE 矩阵中，我们找到最小的 MSE=263.5562 及对应的 i=14，j=6，将 i，j 固定，即固定了高斯核函数的 sigma 与带有 L1 正则化项的 cost function 中的 rho。重新运行代码，即可得到我们寻找的函数关系即参数，对于未来的预测我们输入新的自变量，用上述代码计算即可得到所求的用户体验值。同理，只需修改载入数据的列数，即可计算第 2-4 列与第 6 列函数关系，并进行预测。





②.Python 实现

```
import sklearn
import numpy as np
from sklearn import svm
from sklearn.svm import SVR
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=5e-8)
y_rbf = svr_rbf.fit(X, np.ravel(y)).predict(x_val)
```

【模型评价】

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE：188.104

r^2：0.8022

Cross validation MAE：263.80388

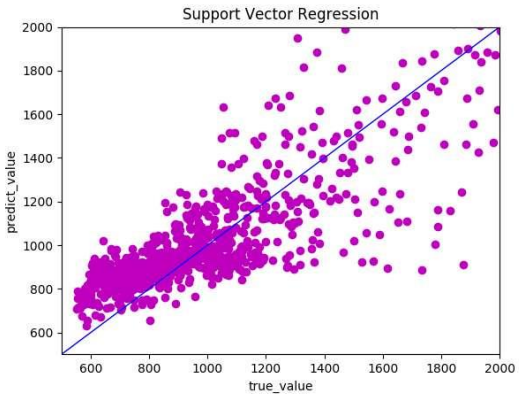
b.考察网络侧变量与卡顿占比之间的关系：

MAE：0.05756495

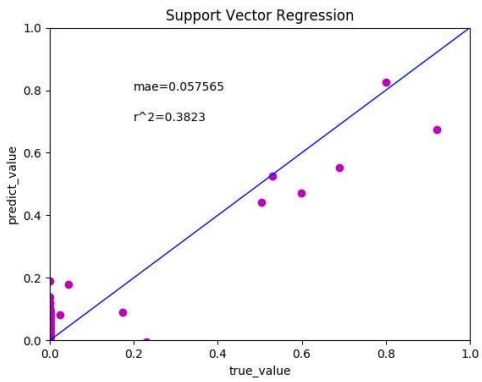
r^2：0.3823

Cross validation MAE：0.070464

图形如下：



网络侧变量与初始缓冲时延



网络侧变量与卡顿占比

六.神经网络模型下的回归分析

在使用了广义线性模型与支持向量回归之后，我们希望进一步寻找更加准确且高效的模型，在经过搜集资料与学习了解之后，我们认为利用神经网络的方法也许可以帮助我们实现这一目标。

6.1 模型原理说明

神经网络一般指的是“神经网络学习”，是机器学习与神经网络两个学科的交叉部分。所谓神经网络，目前用得最广泛的一个定义是“神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所做出的交互反应”。神经网络中最基本的单元是神经元模型（neuron）。在生物神经网络的原始机制中，每个神经元通常都有多个树突（dendrite），一个轴突（axon）和一个细胞体（cell body），树突短而多分支，轴突长而只有一个；在功能上，树突用于传入其它神经元传递的神经冲动，而轴突用于将神经冲动传出到其它神经元，当树突或细胞体传入的神经冲动使得神经元兴奋时，该神经元就会通过轴突向其它神经元传递兴奋。一直沿用至今的“M-P 神经元模型”正是对这一结构进行了抽象，也称“阈值逻辑单元“，其中树突对应于输入部分，每个神经元收到 n 个其他神经元传递过来的输入信号，这些信号通过带权重的连接传递给细胞体，这些权重又称为连接权（connection weight）。细胞体分为两部分，前一部分计算总输入值（即输入信号的加权和，或者说累积电平），后一部分先计算总输入值与该神经元阈值的差值，然后通过激活函数（activation function）的处理，产生输出从轴突传送给其它神经元。

激活函数是神经网络的核心思想之一，最简单的神经元结构叫做感知机（perceptron），由两层神经元组成的一个简单模型，但只有输出层是 M-P 神经

元，即只有输出层神经元进行激活函数处理，也称为功能神经元（functional neuron）；输入层只是接受外界信号（样本属性）并传递给输出层（输入层的神经元个数等于样本的属性数目），而没有激活函数。这样一来，感知机与之前线性模型中的逻辑斯谛回归的思想基本是一样的，都是通过对属性加权与另一个常数求和，再使用 sigmoid 函数将这个输出值压缩到 0-1 之间，从而解决分类问题。

由于感知机模型只有一层功能神经元，因此其功能十分有限，只能处理线性可分的问题，对于这类问题，感知机的学习过程一定会收敛（converge），因此总是可以求出适当的权值。但是对于像书上提到的异或问题，只通过一层功能神经元往往不能解决，因此要解决非线性可分问题，需要考虑使用多层功能神经元，即神经网络。在神经网络中，输入层与输出层之间的层称为隐含层或隐层（hidden layer），隐层和输出层的神经元都是具有激活函数的功能神经元。只需包含一个隐层便可以称为多层神经网络。

非线性激活函数是神经网络的精髓所在，如果不用激励函数（相当于激励函数是 f(x) = x），则每一层输出都是上层输入的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，也就是最原始的感知机（Perceptron）。在激活函数的选择上，我们使用了 Relu 函数，Relu 函数比更早的 sigmoid 函数以及 tanh 函数相比有一些优点：

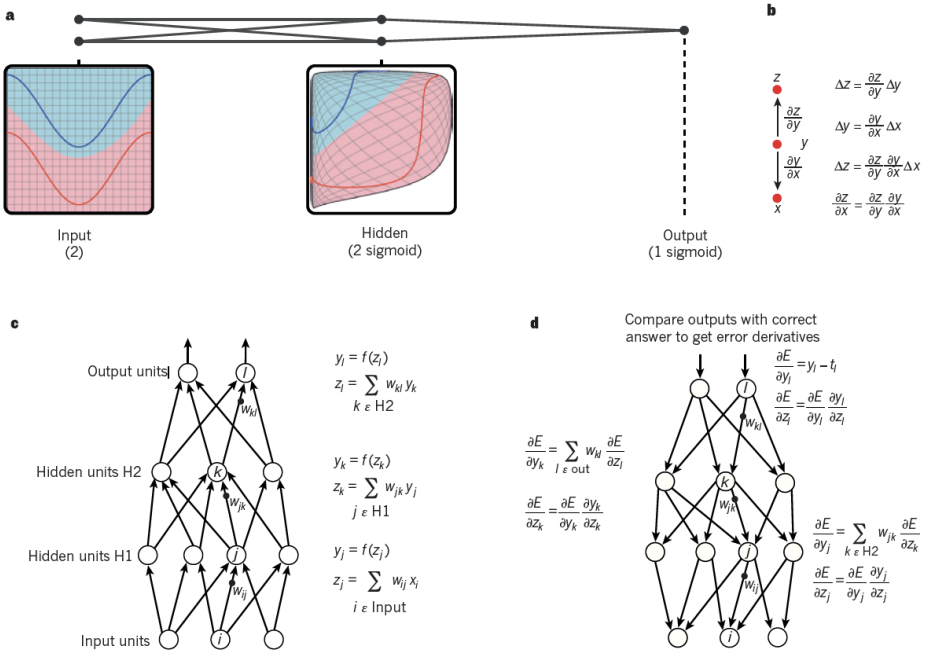
Ⅰ sigmoid 函数在计算反向传播时的求导涉及除法，计算量大，Relu 函数计算量小，节约时间。

Ⅱ sigmoid 函数在深层网络反向传播时容易出现梯度消失的现象，无法完成训练

III Relu 会使一部分神经元的输出为 0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。

6.2 深度学习

理论上，参数越多，模型复杂度就越高，容量（capability）就越大，从而能完成更复杂的学习任务。深度学习（deep learning）正是一种极其复杂而强大的模型。增加复杂度的办法包括：增加隐层的数目，增加隐层神经元的数目。前者更有效一些，因为它不仅增加了功能神经元的数量，还增加了激活函数嵌套的层数。但是对于多隐层神经网络，经典算法如标准 BP 算法往往会在误差逆传播时发散（diverge），无法收敛达到稳定状态。



由于我们只是初步涉及深度学习的知识，因此我们选择搭建一个比较基础的深度网络。我们使用了基于 TensorFlow 的高层库 Keras 作为搭建模型的库。使用 Keras 提供的序贯模型（sequential model）进行搭建。该神经网络比较简单，

层数有五十层，每层作为一个独立的结构，再线性连接在一起。首先我们和前面一样用前一万行数据对模型进行训练，然后用接下来的 1000 行数据进行测试。下面是以拟合网络侧变量与初始缓冲时延数据为例的 python 代码：

```
#导入需要的库和模块
import scipy
import keras
import numpy
from keras import layers
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from keras import models
import sklearn
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline


# load dataset 和前面一样，不再展示
#搭建神经网络模型
# define base mode
model = Sequential()
```

```
model.add(Dense(50, input_dim=3, init='normal', activation='relu'))
#输入的 dimension 为 3 ， 使用 Relu 激活函数
model.add(Dense(1, init='normal')) #可以增加网络层数
#compile model 需要将网络编译后才可以训练
model.compile(loss='mean_squared_error', optimizer='adam',metrics=['mae'])
model.summary()
#打印出网络的相关信息


#train the model
#用样本训练模型，共循环 10000 次，每次送入 1000 个样本
model.fit(X, Y, batch_size=1000, epochs=10000, verbose=1, callbacks=None,
validation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
sample_weight=None, initial_epoch=0)
#保存模型
import h5py
from keras.models import load_model
#model.save('my_model.h5')
#model.save('my_model2.h5')
model = load_model('my_model.h5')
#test the model
model.evaluate(x_val, y_val,batch_size=1000, verbose=1,
sample_weight=None)
```

```
y_pre = model.predict(x_val, batch_size=1000, verbose=0)

from sklearn.metrics import mean_absolute_error

print(mean_absolute_error(y_val,y_pre))

from sklearn.metrics import r2_score

print(r2_score(y_val,y_pre))



import matplotlib.pyplot as plt

from matplotlib.lines import Line2D

figure, ax = plt.subplots()

ax.set_xlim(left=0, right=1.)

ax.set_ylim(bottom=0, top=1)

plt.xlabel(u"true_value")

plt.ylabel(u"predict_value")

ax.annotate('r^2=0.078511', xy = (0.6,0.5), xytext=(0.6,0.5))

ax.annotate('mae=0.102126', xy = (0.6,0.4), xytext=(0.6,0.4))

line1 = [(0, 0), (1, 1)]

(line1_xs, line1_ys) = zip(*line1)

ax.add_line(Line2D(line1_xs, line1_ys, linewidth=1, color='blue'))

p1 = plt.scatter(y_val,y_pre, marker = 'o', color = 'm')

plt.plot()

plt.show()
```

```
# evaluate model with standardized dataset
#define
def baseline_model():
        model = Sequential()
        model.add(Dense(50, input_dim=3, init='normal',
    activation='relu'))  #dim=3
        model.add(Dense(1, init='normal')) #
        # Compile model
        model.compile(loss='mean_squared_error',
    optimizer='adam')
        return model
estimator = KerasRegressor(build_fn=baseline_model, nb_epoch=10000,
batch_size=1000, verbose=0)  #数据少可全尺寸
# fix random seed for reproducibility
import numpy
seed = 7
numpy.random.seed(seed)
X_1 = dataset[:, 1:4]
Y_1 = dataset[:, 5:6]
# use 10-fold cross validation to evaluate this baseline model
kfold = KFold(n_splits=10, random_state=seed)
```

```
mae=cross_val_score(estimator, X_1, Y_1,
cv=kfold,scoring='mean_absolute_error')
mae_scores = -mae
print('cross_val mase:',mae_scores.mean())
```

6.3 模型求解与评价

a.考察网络侧变量与初始缓冲时延之间的关系：

MAE：154.453

r^2：0.7873

Cross validation MAE：411.540

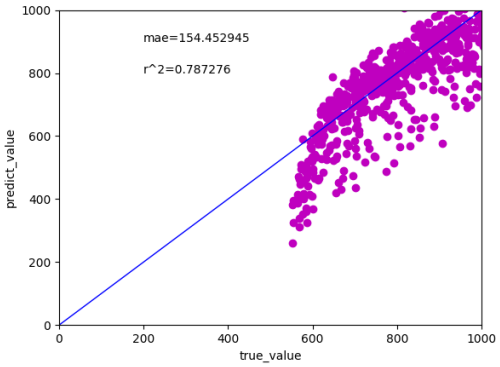
b.考察网络侧变量与卡顿占比之间的关系：

MAE：0.06126048

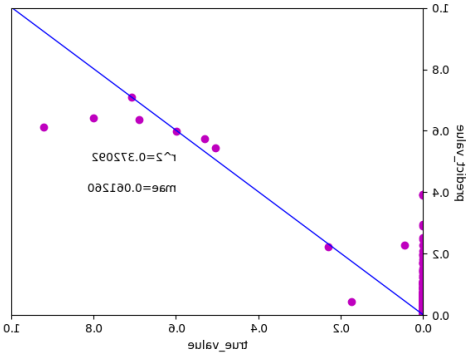
r^2：0.3721

Cross validation MAE：0.19372028

图形如下：



网络侧变量与初始缓冲时延



网络侧变量与卡顿占比

七. 模型评价与改进

7.1 模型评价

在本次建模过程中，我们使用了五个子模型（三个广义线性回归模型、一个SVR 回归模型和一个神经网络回归模型）来寻找网络侧变量与用户体验变量之间的函数关系。以下两表分别是这些子模型初始缓冲时延与卡顿占比的评价参量。

	MAE	r^2	Cross validation MAE
最小二乘法	322.23	0.57865	431.16
LASSO 回归	322.23	0.57865	431.04
岭回归	322.23	0.57865	431.04
SVR(matlab)	263.0858	0.6254	暂缺
SVR(python)	188.10	0.80221	263.80
神经网络	154.453	0.7873	411.540

	MAE	r^2	Cross validation MAE
最小二乘法	0.093459	0.10814	0.12668
LASSO 回归	0.093131	0.11334	0.11473
岭回归	0.093131	0.11334	0.11509
SVR(matlab)	0.0074133	0.9244	暂缺
SVR(python)	0.05756	0.38231	0.07046
神经网络	0.06126048	0.3721	0.19372028

结合表中数据和我们建模过程中的体会，可对模型做出如下评价：

Ⅰ 用传统的回归分析计算用时较短，但是结果没有 SVR 准确与可靠。

Ⅱ 对模型进行验证时，人为的划定训练集与测试集并不足够可靠，因此我们可以引入交叉验证的方法（cross validation），随机划分数据集，对每份样本分别进行验证。

Ⅲ SVR 方法更为复杂，计算时间较长，但是准确度更高，更为可靠。通过应用高斯核函数，使得计算高维向量变得简易可行，参数只有一个，调节较方便。但是也有缺乏物理意义，不计算出具体的ω，比线性计算慢的缺点。在应对核函数造成的过拟合现象，我们使用两个正则化项来纠正，两个正则化也可以互相平衡，

减少参数的个数。

III 使用深度学习进行预测，并没有具体的函数关系式，但是可以进行非常快速的计算，精确度也较高。由于我们只是初步涉及深度学习的知识，因此我们选择搭建一个比较基础的深度网络。我们使用了基于 TensorFlow 的高层库 Keras 作为搭建模型的库。使用 Keras 提供的序贯模型（sequential model）进行搭建。该神经网络比较简单，层数有五十层，每层作为一个独立的结构，再线性连接在一起。

7.2 模型改进：

I 因为时间匆忙，我们还没有来得及调整超参数以及网络结构，可以看到神经网络在指定数据集上的结果比较理想，但是应用 cross validation 后 MAE 值较大，如果以后还有机会可以进一步调整，但是鉴于深层网络的调节并没有一套清晰简明的方法可以遵循，因此可能需要花费一定的时间。此外，此题因为数据量有限，也不一定适合用比较成熟的深度学习算法如 CNN 或者 LSTM 求解。

II 对于 SVR 方法的参数可以进一步优化，对 i 和 j 可以进行更精确的界定，但是由于计算时间的限制，短时间内的优化难以实现。对于核函数的选择也可以尝试其他函数，但是由于时间限制我们选择了较为适合本题的函数，如果时间足够还可以尝试其他的函数。

III 对于深度学习方法，可以进一步调节参数，调节神经网络结构，这个过程并没有准确的方法可以遵循，如果时间足够也可以进一步调节。

八. 参考文献

[1] Jun Liu and Jieping Ye, Efficient Euclidean Projections in Linear Time, ICML 2009.

[2] Jun Liu and Jieping Ye, Sparse Learning with Efficient Euclidean Projections onto the L1 Ball, Technical Report ASU, 2008.

[3] 周志华,《机器学习》，北京：清华大学出版社，2016，121—139

[4] 李航,《统计学习方法》，北京：清华大学出版社，2012，95—123

[5]Lecun, Deep Learning, NATURE, MAY 28 2015, 436—444